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

13 March 2006

University Supervisor: Prof. Vassil Alexandrov (University of Reading)
Placement Supervisor: Dr. Adil Hasan
(CCLRC Rutherford Appleton Laboratory)

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 a network and, as a result, to make the storage infrastructure appear transparent for the end user.

This dissertation presents the development of multiple highly configurable and independent monitoring tools, which operate within a network to support and improve the administration and debugging 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 are 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 using remote procedure calls in conjunction with the Extensible Markup Language (XML). Special attention was paid to 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. This dissertation provides inside knowledge about design and implementation issues, as well as issues faced problems during the development, and the corresponding solutions.

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 for 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.

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
CD	**C**ompact **D**isc
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
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
XDR	E**x**ternal **D**ata **R**epresentation
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 virtual networks. 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 computational grids, access to distributed computing resources, data grids, access to large amounts of distributed data, and equipment grids where the grid is used to control certain technology remotely [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 a specific piece of 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 problems mentioned above. 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 chapter a brief overview about the dissertation structure and used conventions are given. Further, the project description as well as an introduction to the SRB data grid management system are presented.

Each chapter begins with a little summary about what the reader is going to find on the following pages.

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 as well as existing software products are analysed.

Possible solutions, specifications and class diagrams, including a brief description of all functions and member variables, are presented in **Chapter 4**.

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

Results gained and tests made are illustrated in **Chapter 6**.

Chapter 7 summarises this project and finally, achievements and future prospects are presented in **Chapter 8**.

The 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 difficulties to debug or supervise a running system have been observed. Errors, *e.g.* due to problems with a server connecting to a remote SRB master or if the meta data catalog (MCAT) server is 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 is a highly configurable software package which involves the SRB log file analysis. According to the results of the log file structure 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)
  - additional file handling (configuration files etc.)

- 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 processes 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 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 systems within a network. In addition data grid offers data resources. By using data grids, it is possible to access data which might be distributed on several computer systems. The grid can be designed in such a way, that the user does not know where exactly the data is located. This possible transparency is called data virtualisation.

The Storage Resource Broker was developed as a project of the San Diego Supercomputer Center (SDSC).

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 a distributed data grid network [5] including:

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

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 (with connection to a MCAT) to access any data stored in the system.

- 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 automatically redirects 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 the user to copy and migrate data to a new system without affecting access.

Figure 1.1 gives a brief overview of the SRB architecture.

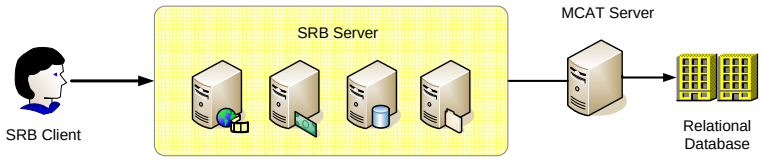

FIGURE 1.1: SRB Architecture

The core is the SRB server, which accepts enquiries from the SRB client. The server knows all meta data about users, resources and datasets. Basically, meta data is a piece of data which contains information about another piece of data like content, type, location etc. This structure enables the user to quickly gain a basic summarised knowledge about data he might be interested in. Through the meta data structure, the original data is well organised and a search can be performed very quickly.

Two different kinds of SRB servers exist:

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

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 over the connection to the SRB agent, who handles all the client enquiries.

Each server maintains one log file. All running SRB server processes alter this log file if an event happens. In the course of this dissertation a monitoring system will be developed. This includes the evaluation of the SRB server log file, since the SRB system does not provide any possibility to carry out this task. 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 also points out XML handling and certain XML restrictions. Furthermore, Python, the programming language used in this project, is introduced.

2.1 Basic Network Principles

A network can be described as a pool of different and individual electronic 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 called 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 describes 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) originally developed.

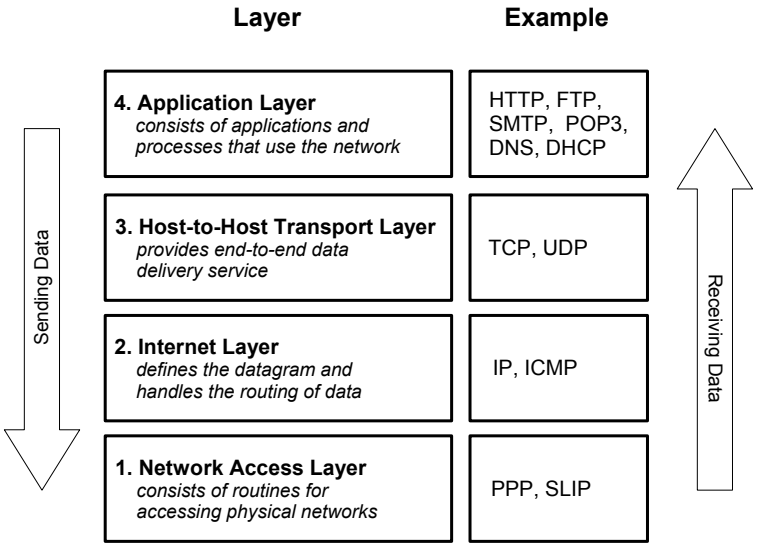

FIGURE 2.1: TCP/IP Protocol Stack based on the DoD-Model

Each layer provides its own structure and conventions. If data has to be sent, it is passed 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 evaluates and 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 ($ACK = x + 1$). A SYN packet with another sequence number y is then sent 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 an end packet (FIN) is sent. Again, the reception of the packet is acknowledged (ACK).

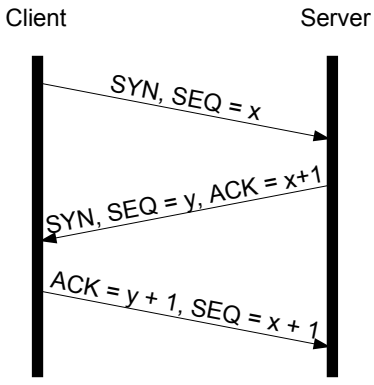

FIGURE 2.2: TCP Handshake

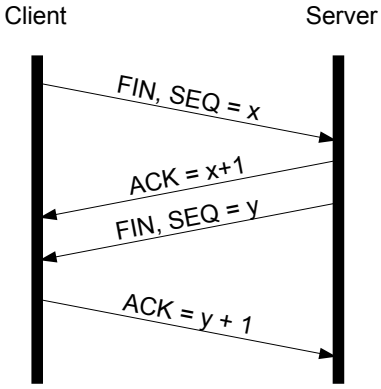

FIGURE 2.3: TCP Connection Termination

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. If no acknowledgement was received or a timeout occurs, the packets are resent. 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 in 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 being 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 no continuous 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 a 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 an 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 mail. 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 sent to an SMTP client. The client determines the SMTP server using other existing technologies. The mail is then sent 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 systems can be derived from 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 represents an architecture of distributed intelligence and is cross-platform compatible. It is possible to run client and server applications on different operating systems. In contrast to 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, a secure connection provides data confidentiality. Nobody should be able to eavesdrop on the information transmitted. Another important point is data integrity. The system should be able to detect an alternation in the content of a data packet. 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 available, for example

- Symmetric Key Encryption

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

Symmetric key algorithms use 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 in connection with password protection, because a certain input always creates the same output. If a secret key is combined with the production 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].

Cryptography 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 fill the available timeframe. 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

Today, the most widely spread security protocol is the 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 described on page 11 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 meet the requirements mentioned above the SSL protocol consists of several protocols as illustrated in Figure 2.4 [10].

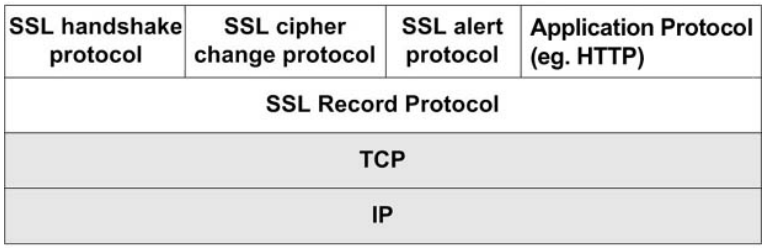

FIGURE 2.4: SSL Protocol Stack

The Application Data Protocol is responsible for the data transfer between the application and SSL. To establish an 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.

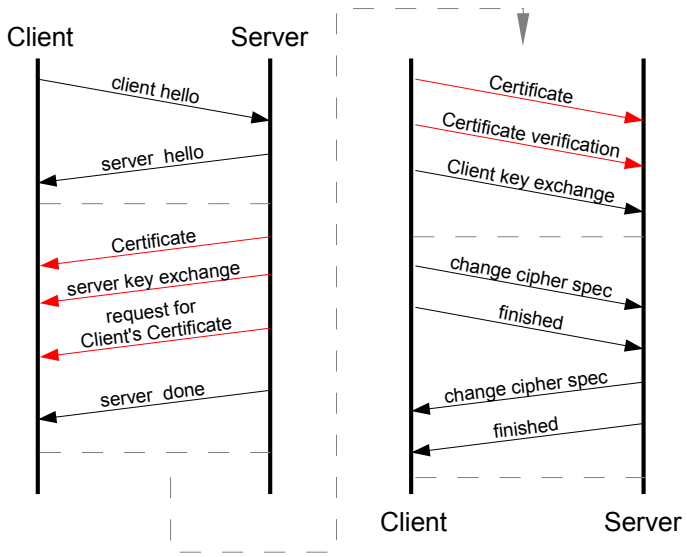

FIGURE 2.5: Possible SSL Handshake (red = optional)

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 a single 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 `server done` 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 to encrypt the data. 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 its 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 is completed, the data will be 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 on the basis of SSL 3.0. Under RFC 2246, they released a new Transport Layer Security (TLS) protocol version 1.0 in 1999. TSL implements the same features as SSL and additionally contains more interoperability and expandability towards applications. Additional RFC's and extensions have been published by the IETF in conjunction to TSL. Presently, the development of TLS version 1.1 by the IETF is in progress. A draft is available so far.

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

2.4 XML

This project requires a platform independent exchange data system. A very flexible way of data exchange is offered 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 also provides 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.

It 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 the 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 -->
    <!ELEMENT error_number (#PCDATA)>
    <!-- error_number contains the error_number text and nothing else -->
    <!ELEMENT error_string (#PCDATA)>
    <!-- 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 XML "version" used 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 report to 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 an XML document. The DTD separates the data from the data definition. DTDs 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 carry the actual data. An 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 to 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 are 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 #xD (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 two different kinds of parsers exist. Parsers which validate the source code (requiring a DTD) and parsers which does not execute validation.

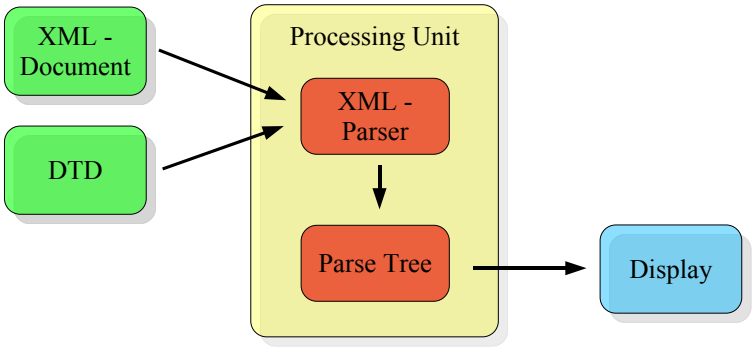

FIGURE 2.6: Possible XML Processing

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 with the Document Object Model (DOM) according to the W3C. The two application programming interfaces (APIs) most commonly 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 integrated 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 not a 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 commonly 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 over 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 to be a 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 memory management system. Garbage collection is a method of freeing unused memory and other system resources automatically. 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 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 which can be very helpful, but misused with a 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 this choice is 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 for the project, *e.g.* to execute the log file parsing. Strategies are examined to establish network communication efficiently. 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. Examining this kind of data can be difficult, especially as each log file may have a different structure. Additional knowledge may be needed to interpret the data and not all the information is important. But how does one determine which data is worth looking at?

This project demands a log file parser which

- can identify any defined errors
- can be dynamically configured
- is efficient to use
- is accessible and manageable with Python
- can run on a UNIX system
- is free of use (if extra software package)

Internet research discovered that many different log parsers exist. 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 generators 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 are involved.

Instead of trying to adjust existing software solutions the decision was made to develop a 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 compactly. 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 between the required client and server applications is realised by using 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. Sockets
   Sockets are the basis for communication through a network and can be described

[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

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 the 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 protocols and services are defined which facilitate the creation of distributed applications in heterogeneous environments. CORBA is language independent and uses an 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 a 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). From the ORB the data is sent to the ORB on the remote system where a skeleton is called. A skeleton is a piece of code as well; in this case the skeleton does 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 local procedure calls.

The requirement of using Python 2.2.3 limits the choices. Python provides 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 an efficient way. Within the RPC package, sockets are used and the socket implementation is stable and reliable.

3.3 Daemon

Several applications of this project will be run as background processes. Therefore those applications should be turned into a daemon processes. 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 an 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, this is not possible. Hence a directory change might be useful.

5. `umask`
   `unmask` sets the file creation mask for a process. The file creation mask defines which rights are **not** to be assigned to a 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 respectively. The information about the new session 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.

The project focuses 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

```
1 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
- a 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 took 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 Listing 3.2.

LISTING 3.2: SRB Log File Entries

```
1 getAndQueHostName: gethostbyname error for mda-18.sdsc.edu ,errno = 22
2 LocalHostName:  zebedee.local, localhost, 130.246.42.39, 192.168.0.2, 127.0.0.1,
      192.168.0.2, Port Num: 5544.
3 Local storage vault conf:
4 storSysType: 0, vaultPath: /Users/hasan/work/SRB/Vault
5 Local Zone :
6 ZoneName = AdilZ  HostName = zebedee.local  PortNum = 5544
7 Remote Zone :
8 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 number of events occurring 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 above 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 one way to utilise security features as described in Section 2.3.

OpenSSL consists of a cryptography library and an SSL toolkit and is derived 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 published. 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.(1,2,3,4), OpenSSL 0.9.7 and SWIG 1.3.2.(1,2,3). 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 takes the declarations found in C/C++ header files and uses them to generate 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 good documentation and examples were provided by the developer. Furthermore, the handling was understandable and efficient. Therefore, the decision was made to use M2Crypto instead of pyOpenSSL, because the documentation is insufficient and no examples were 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

These 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 supposed 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 was chosen. Pysqlite is a database interface for SQLite and is freely available. Due to compatibility the version pysqlite 1.0.1 was 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.

Graphs or diagrams are required to display error statistics. Firstly, individual errors have to be displayed with the corresponding occurrence in total. A bar chart diagram can realise this. Secondly, a diagram is necessary where a certain error can be displayed as function of time. Suitable would be a line diagram.

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.

The usage of the `matplotlib` module would allow to generate sophisticated graphs and diagrams. *The `matplotlib` is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms* [21]. This library can be compilied to use the TK toolkit as a GUI backend and requires Python 2.2 or higher [21]. Unfortunately, additional software packages such as `Numeric` or `numarray` and `freetype` are required as well [21]. This led to the decision to use the Tkinter interface only, since the project requirements state against unnecessary additional software modules. Supported is the decision made by the fact, that merely certain defined graph styles, in particular bar charts and line diagrams, are needed.

4 Design

In this chapter all design issues concerning the software development are presented and explained. First a few general aspects about the new monitoring system are given. 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 includes some software specifications as well.

4.1 General Aspects

All applications are written as console applications. 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 regarding handling a console application is expected.

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

According to the project description, two main applications are needed. First of all a server, which is handling the log file parsing. Secondly, a client has to be developed which collects the parsing results from the server and handles the storage and display of the parsed data as well as the notification. According to the analysis in Chapter 3, it is adequate to base design of these two applications on a client-server-architecture.

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

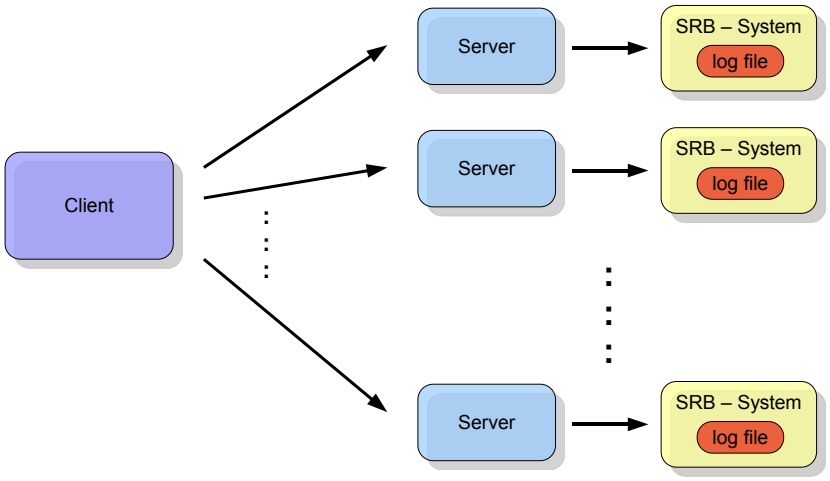

FIGURE 4.1: Client-Server-Relation (1:n)

As decided in the analysis (Chapter 3), the communication is done with an adjusted version of the Python standard library module `SimpleXMLRPCServer`. The integration of a password protection 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 interferences should be avoided.

Once the server is started, it keeps track of all log file changes. A separate tool is developed to additionally integrate older log files which are stored as compressed files (*.gz). Since this integration is done only once, this process is sourced out to another module, which uses the same parsing technology as the server.

Often, a `utils_*` class can be found in the adjacent class diagrams. This is **not** a class, it represents a script which contains functions, which are needed by multiple other classes. The class diagrams shown 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.

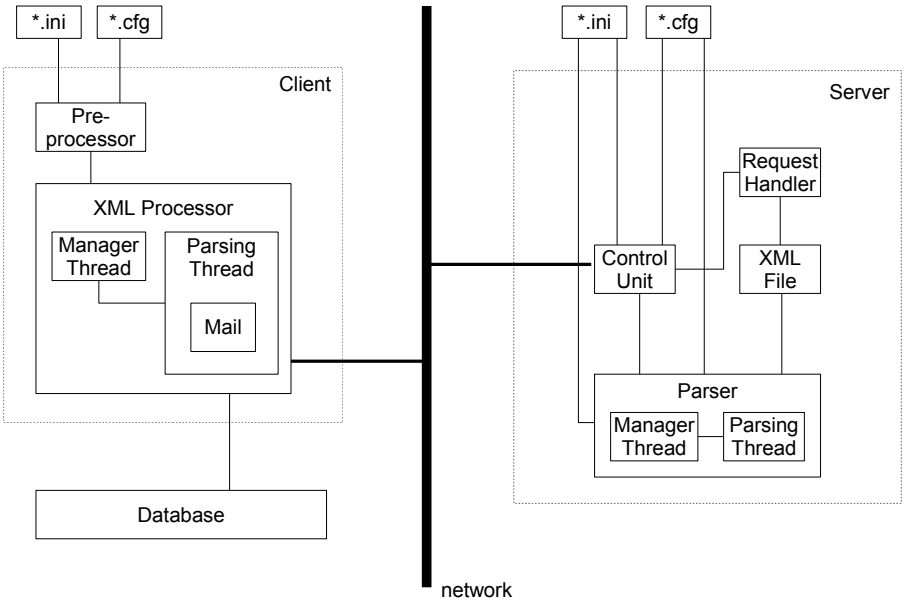

FIGURE 4.2: Basic Client-Server Design

The server's control unit is responsible for verifying the user input. Furthermore, this unit starts the parser with the correct configuration and accepts incoming requests. The requests are passed on to 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
  - SRB compressed log files
  - 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 certain resources are not required by multiple threads simultaneously. Thread synchronisation is done with mutex mechanisms. If such mechanisms are used, a system of deadlock avoidance has to be established.

The parsing module analyses 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 identified as being of interest the following values are extracted:

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

The date and time are 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 proceeds with the next line in the log file. The expression “error string” refers to the whole log file entry line. The line number defines 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 to be able to access the XML file 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 tries 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 runs 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 as well. 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.

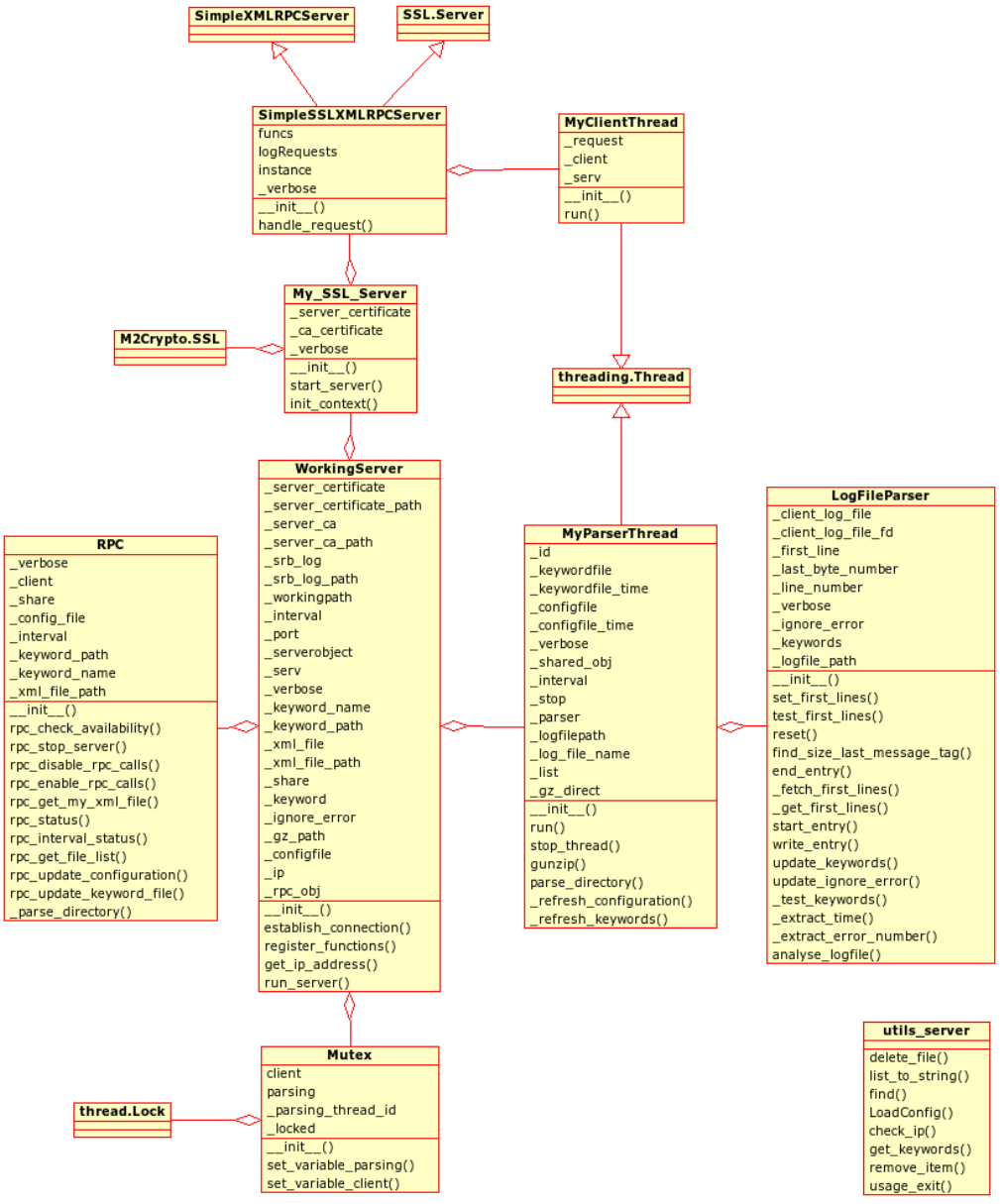

FIGURE 4.3: Server Class Diagram

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

- a mutex object (`Mutex`)
- RPC functions (`RPC`)
- a parsing thread (`MyParserThread`)
- a secure server (`My_SSL_Server`)

and initialises all necessary objects. The required data is held 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

Continued on next page

Table 4.2 Member Variables - *continued from previous page*

Variable	Type	Explanation
`_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
`_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 handles the parsing and runs as a thread. The 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

Continued on next page

Table 4.3 Member Variables - *continued from previous page*

Variable	Type	Explanation
_configfile_time	STRING	last modified time of configuration file
_verbose	INTEGER	defines printing of debug messages
_shared_object	MUTEX	object of class Mutex
_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 file 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 periodic parsing is organised. Firstly, the configuration file and keyword file are checked for modifications. 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 generated 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 itself. 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
`_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` saves the first fifteen lines and `_fetch_first_lines` only reads these lines from 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` resets the member variables `_line_number` and `_last_byte_number` after a log file rotation. 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 detected with `_extract_error_number`. The most important function is `analyse_logfile`. A flow chart of the most important program loop is displayed in Figure 4.4.

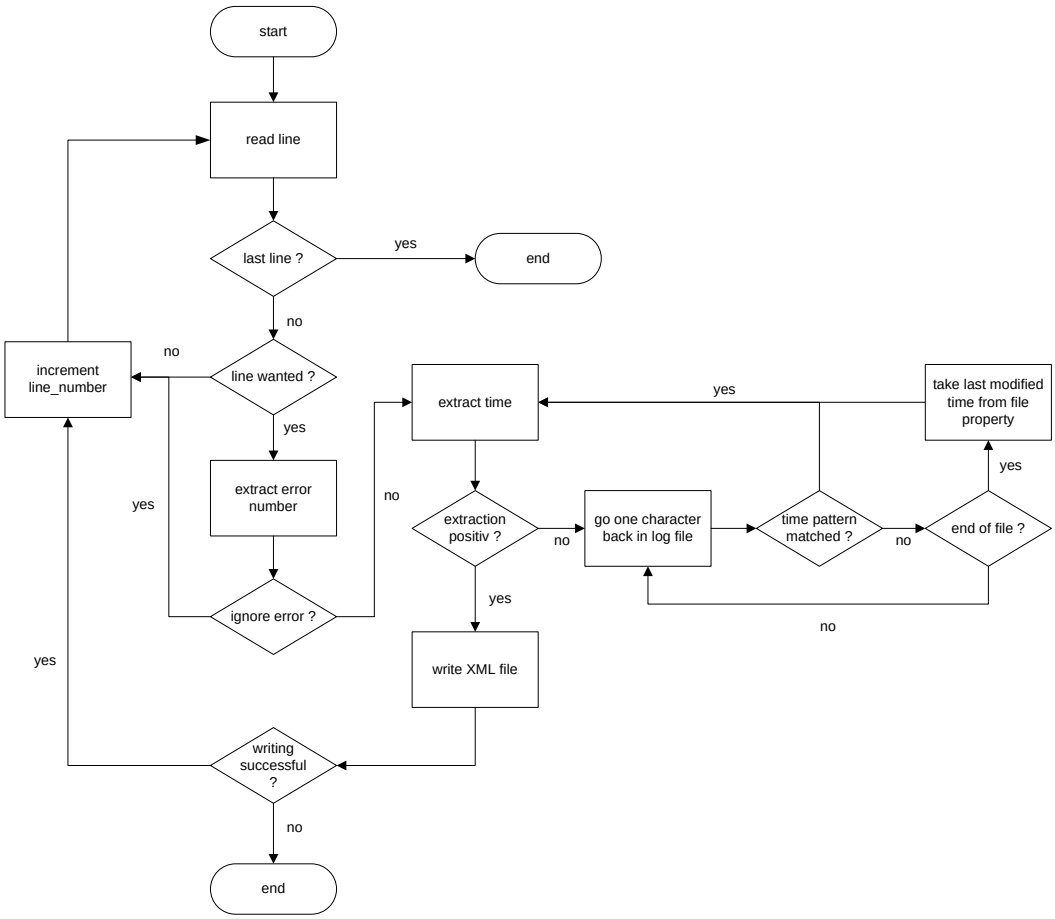

FIGURE 4.4: Flow Chart `analyse_logfile`

The parser reads the log file line by line. If the end of the file is reached the parsing process is terminated as well as if problems occur while 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 inform are possible. According to the mode the corresponding file is modified or the required information is extracted. During the file modification the file is partly deleted and after exchanging or deleting the required value, rewritten. Figure 4.5 depicts a top level flow chart diagram as an overview of the such a function.

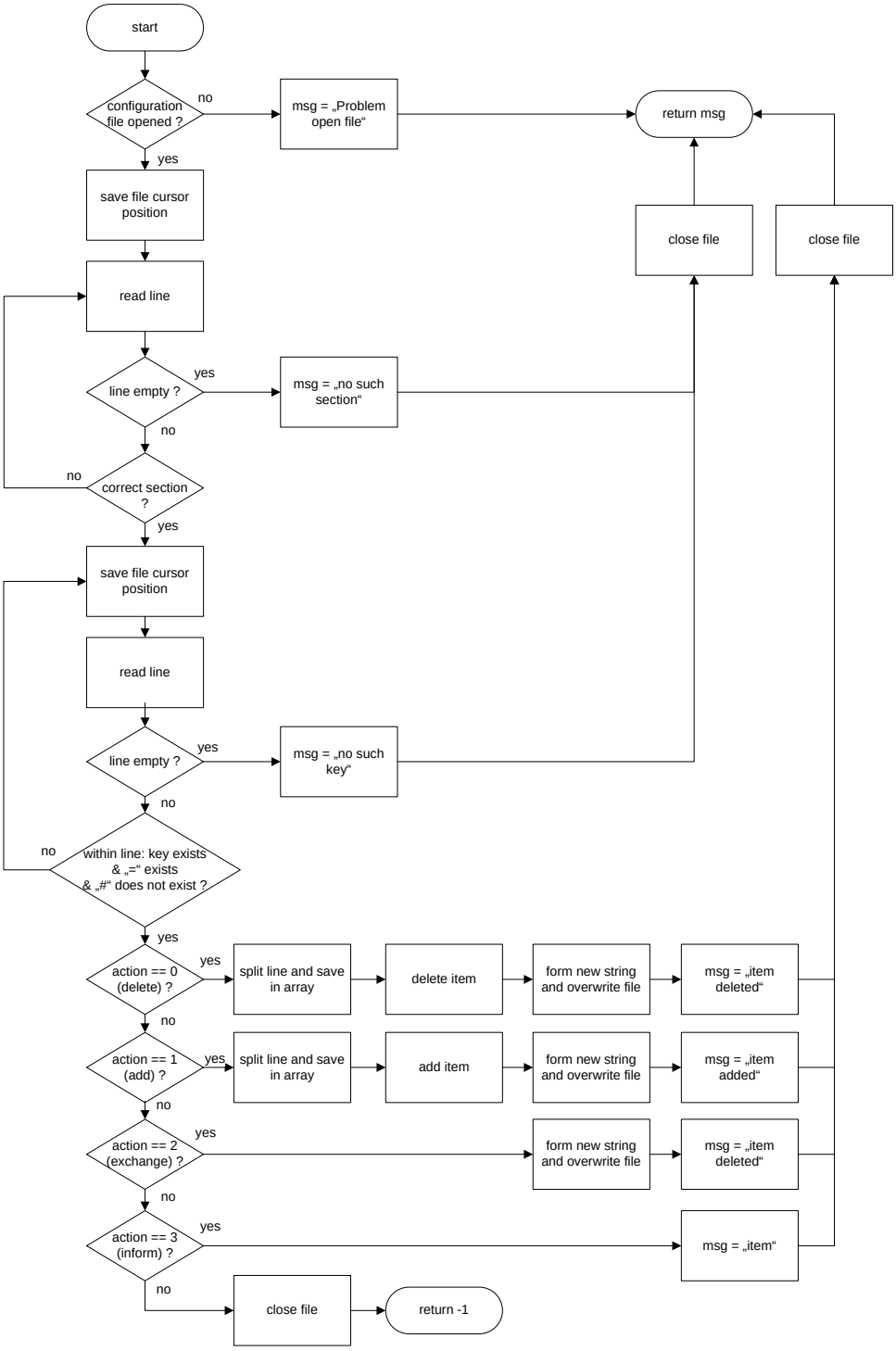

FIGURE 4.5: Flow Chart `rpc_update_configuration`

The configuration file is read line by line. As soon as the concerning section and keyword are found the values are processed according to the defined mode. The file is modified and an appropriate message returned. A message is returned in any case, also if the required section or keyword are not found.

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 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:

`BaseServer`
⇓
`SocketServer.TCPServer`
⇓
`SimpleXMLRPCServer`
⇓
`SimpleSSLXMLRPCServer`

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 variables 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

Continued on next page

Table 4.8 Member Variables - *continued from previous page*

Variable	Type	Explanation
`_client`	INTEGER	indicates if client is busy
`_parsing_thread_id`	INTEGER	identification number of parsing thread
`_locked`	THREADING.LOCK	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, followed by the XML processor which works independently after its started. The processor is controlled by the manager thread. The actual connection to the server is established by the parsing thread, which also takes care of processing the XML file (storing the preprocessed information in a database) and the email notification procedure.

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 the
  - 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. In the case of successfully file transfer, 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 via 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 every XML entry is stored in the database sequentially. 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 number 999999 is assigned to the error message. New error numbers are automatically inserted in the database. Double entries are avoided by checking the database beforehand if the entry already exists. Such circumstances 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 additional error numbers. The keywords, listed in a keyword file, contain all those keywords, where the recipient is not interested in notification. 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 is deleted. This is followed by creating a mail using the mail content file and sending it via SMTP. The procedure is realised by using the module `smtplib` from Python's standard library. If the content file consists of more than 5000 entries, only the total amount of error occurrences and the time range when the errors occurred are sent. This restriction prevents an undefined size of the final mail. Email provider have usally restricted the size of a single mail. By introducing the previously mentioned restriction the size of a mail is kept below 1 megabyte and does not interfere with any provider. A single mail is sent for each server monitored and each XML file fetching process. 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 saved is not an option, since the Python scripts (source code) are stored as plain text and so 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 this purpose 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.

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

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.

4.3.1 Client Class Diagram Design

The class diagram of the client application is shown in Figure 4.6.

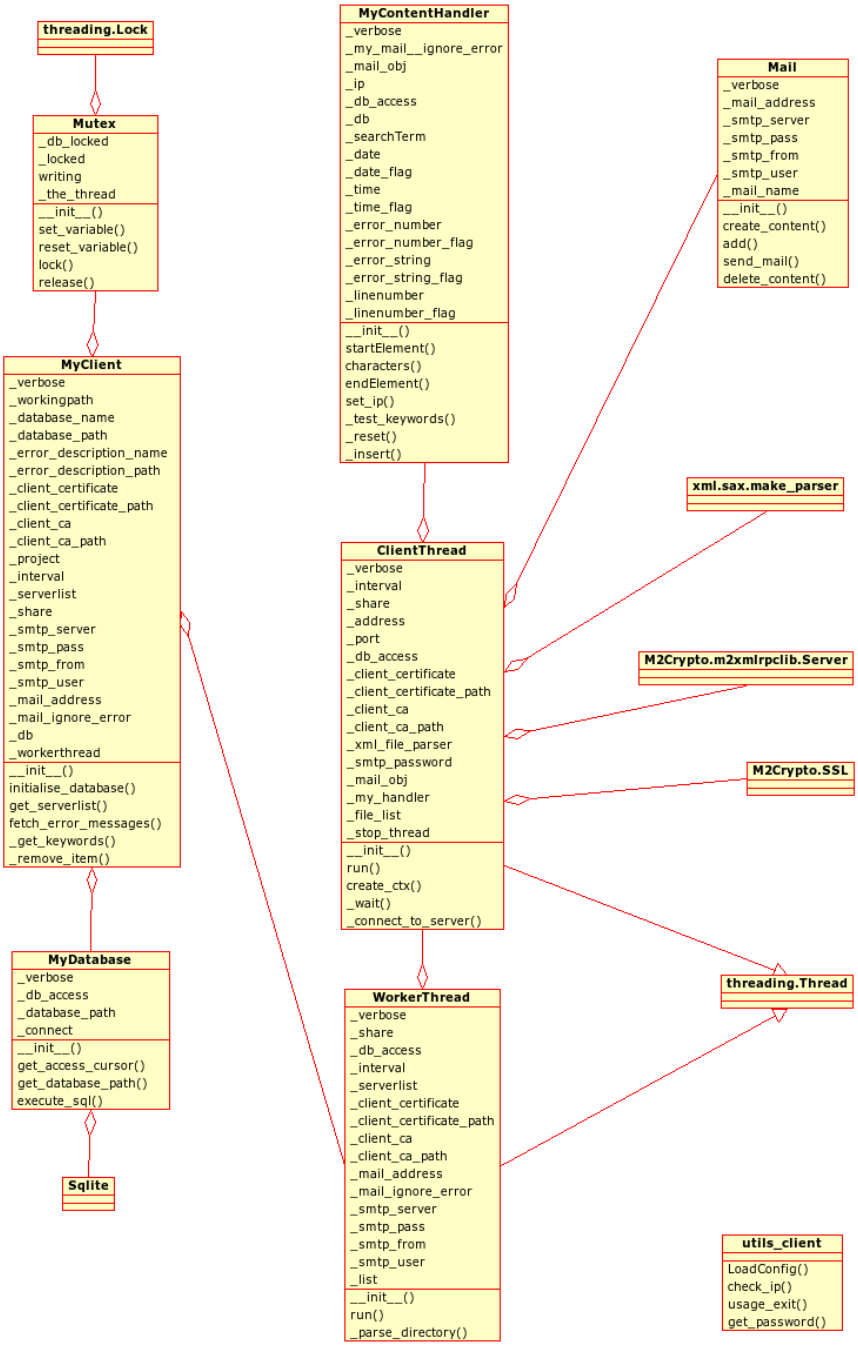

FIGURE 4.6: Client Class Diagram

The manager class `MyClient` verifies 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
`_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`

Continued on next page

Table 4.10 Member Variables - *continued from previous page*

Variable	Type	Explanation
`_workerthread`	WORKER-THREAD	object of class `WorkerThread`

The constructor `__init__` initialises the member variables whereas the function `initialise_ database` initialises 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.

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

Continued on next page

Table 4.11 Member Variables - *continued from previous page*

Variable	Type	Explanation
_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, handles the actual XML fetching and processing in conjunction 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

Continued on next page

Table 4.12 Member Variables - *continued from previous page*

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

The constructor `__init__` initialises the member variables. The XML parser requires a content handler which is provided by `MyContentHandler`. The necessary SSL context to connect with the server is supplied by `create_ctx`. The function `_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 has become availalbe (busy waiting). Within `run` the whole XML file fetching and processing procedure is executed.

The fetching consists of three parts. Figure 4.7 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 are available, the actual fetching takes place (part III).

All these parts contain routines for scenarios such as the remote procedure calls are disabled, the server is busy and the server is not reachable.

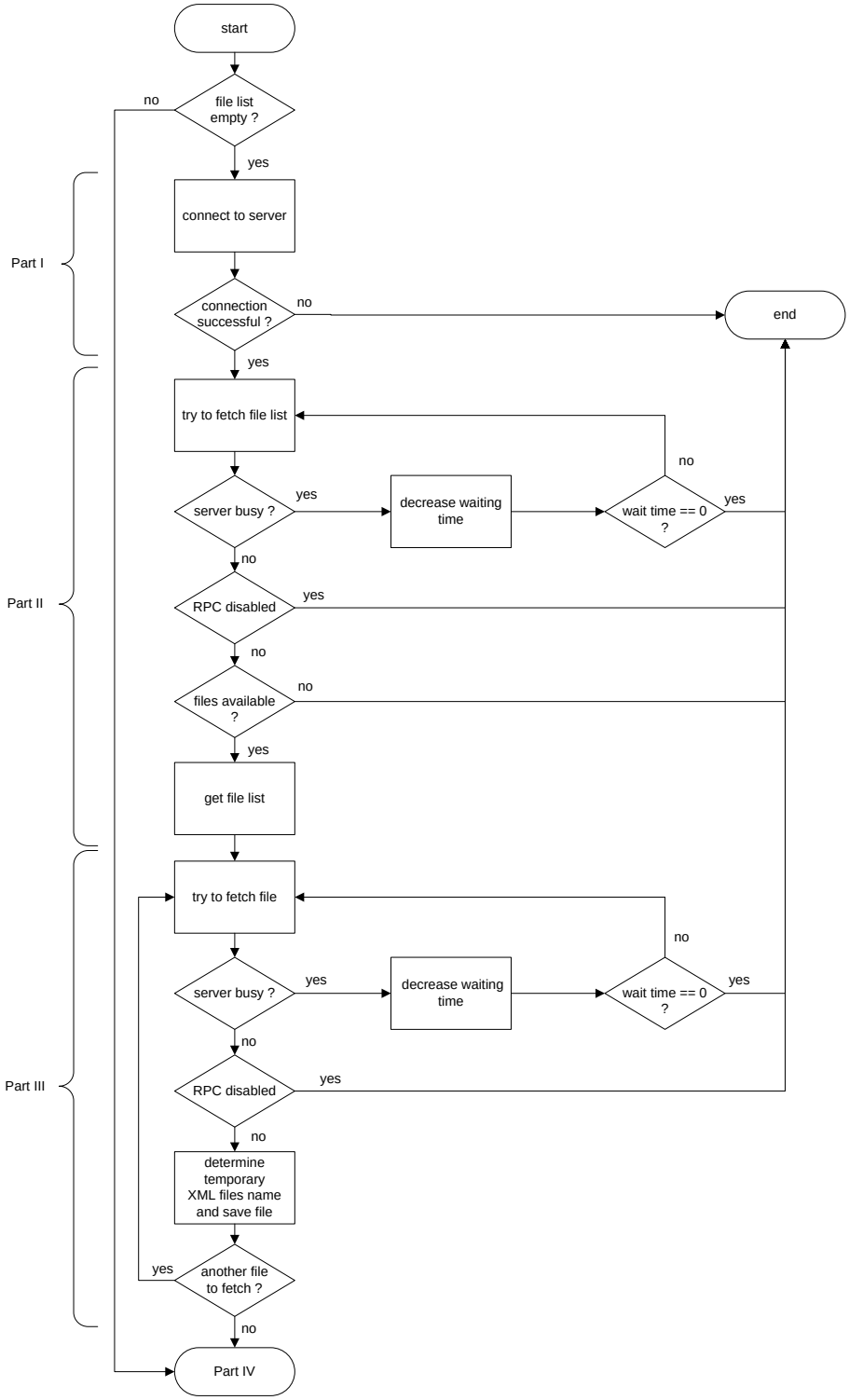

FIGURE 4.7: Flow Chart `ClientThread - run()` Part I - III

Now the XML processing is executed (part IV). The temporary saved files contain the IP address from the producing server. Figure 4.8 shows a top level flow chart diagram of part IV.

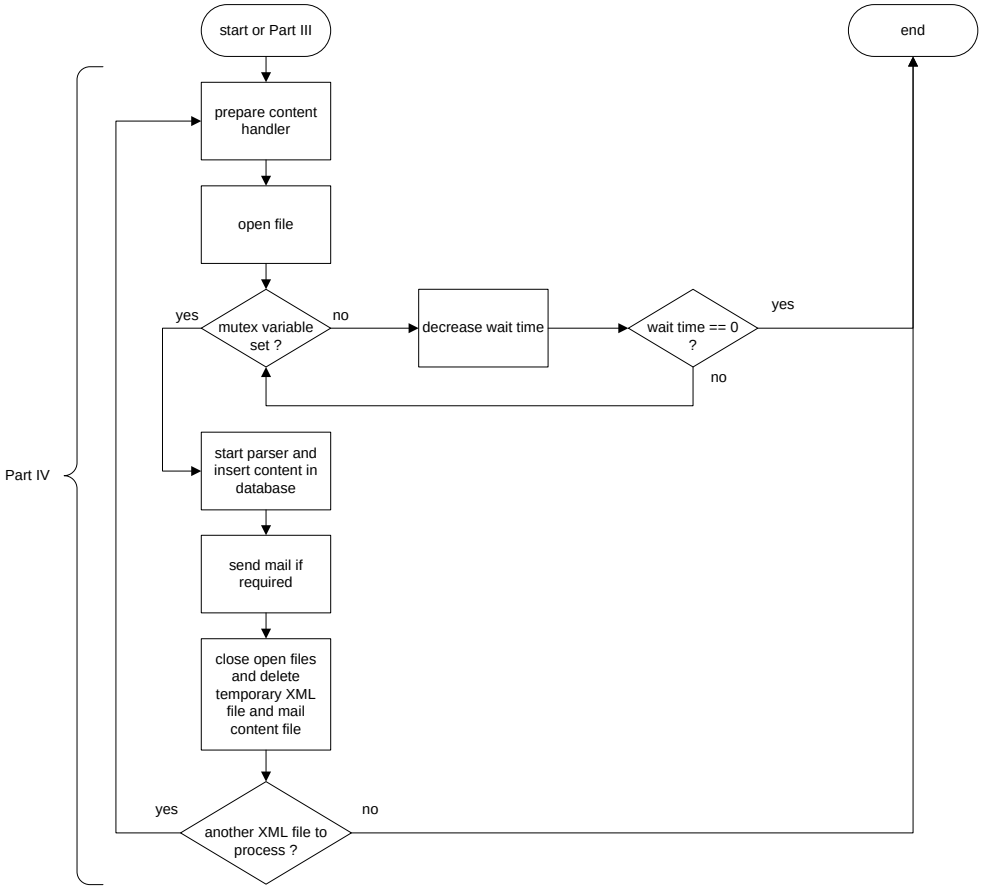

FIGURE 4.8: Flow Chart `ClientThread - run()` Part IV

Firstly, 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.

The XML parser module has to obtain the information regarding how to manage the content of the XML file. This provides `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 take place, *e.g.* double entry check. For the mail content creation 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 variables are listed 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 4.9 gives an overview about the constructor structure.

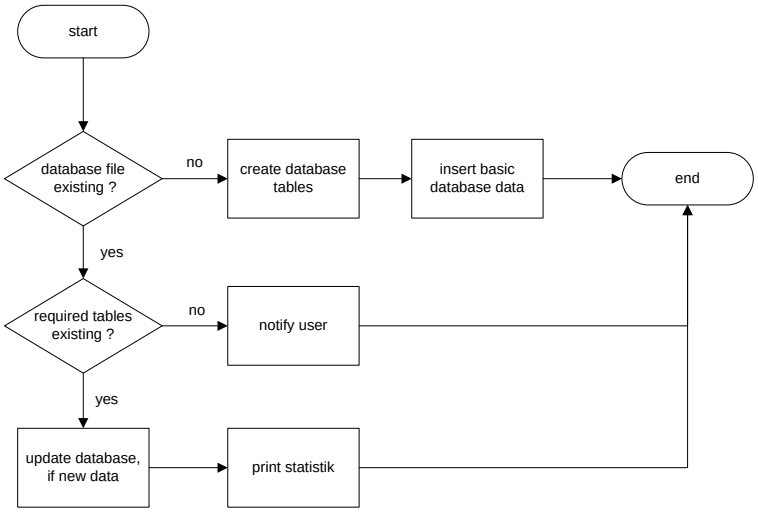

FIGURE 4.9: Flow Chart Constructor `MyDatabase`

The database is created and basic data, such as errors from the error description file, are inserted 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. With the `execute_sql` function all SQL commands are executed.

Tools for sending a mail are provided by the class `Mail`. Table 4.15 displays the member variables.

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 generated. The function `add` is used to append information to the content file. The content file is deleted with `delete_content`. The mail is formed and 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` sets 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 properties of the monitored server 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 information 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 the 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 4.10:

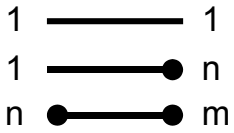

FIGURE 4.10: Cardinality within ERM

Figure 4.11 shows an extended Entity Relationship Model for the required database. The extended ERM defines precisely the range of possible values (min, max).

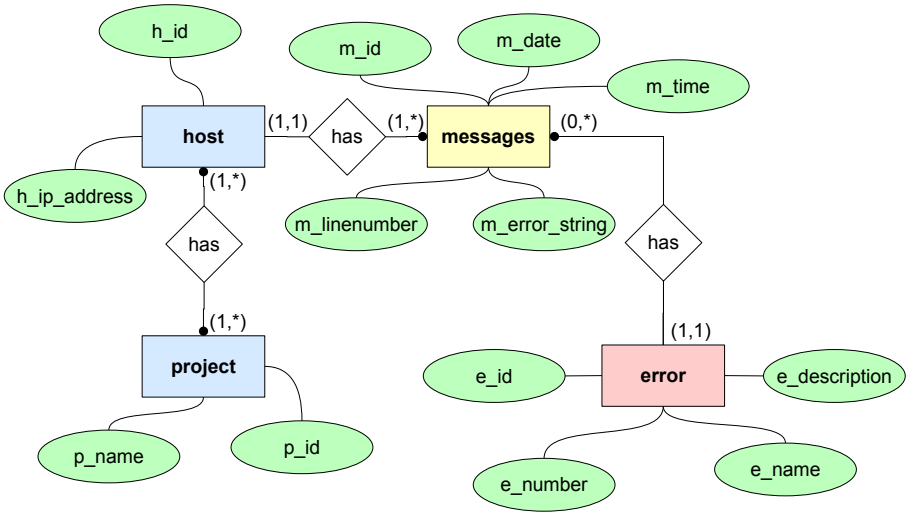

FIGURE 4.11: Entity Relationship Model

Based on the ERM, Figure 4.12 illustrates the design of the database.

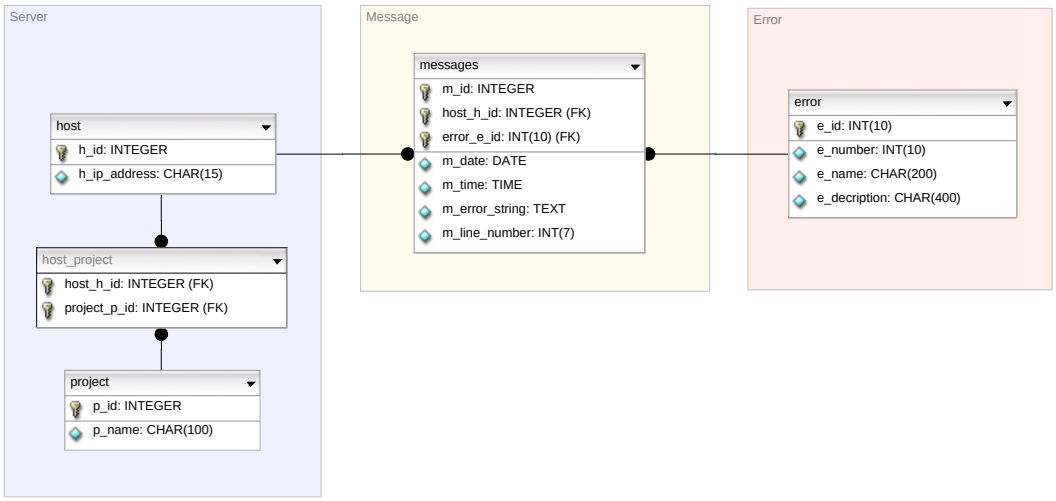

FIGURE 4.12: Database Design

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`	TEXT	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 multiple 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. The connection table (Table 4.21) is needed to associate 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 itself. Table 4.22 contains a summary of all required 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)

Summarised, the user is able, through a combination of parameters, to gain access to the required information in 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 error occurrences (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 date, the error occurrence 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 more 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 is provided. To prevent the user accidentally closing a window, a message box is implemented to get a confirmation about the forthcoming action from the user.

The class `Display` evaluates the given parameters and queries the database. If a graph is needed the queried 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 4.13 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.

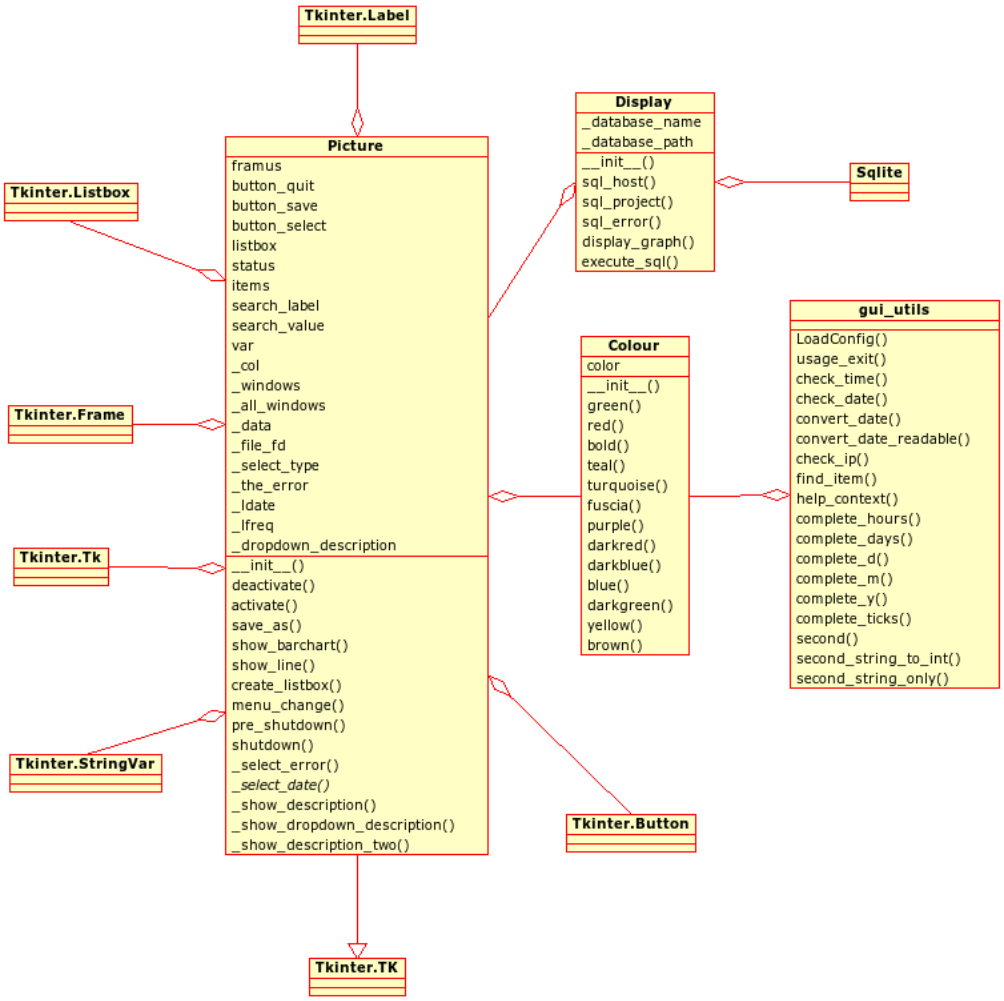

FIGURE 4.13: Virtualiser Class Diagram

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

The class `Picture` handles 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_dropdown_description, and _show_description_two are used to modify the status bar. The actual graphs 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 source code of Graph.py is taken from an example presented in the book “Python and Tkinter Programming” by John E. Grayson [22] 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
–interval_status	show status of parsing interval
–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 4.14 shows the class diagram for the Remote Controller. The class `Colour`, already utilised for the Virtualiser, is used to colour the console output. This is a special feature to increase usability and is described more closely in Chapter 5.8.

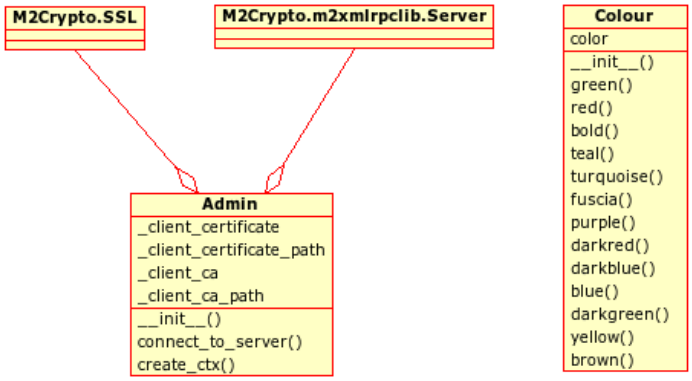

FIGURE 4.14: Remote Controller Class Diagram

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 compressed and saved 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 which structure is already described in Section 4.2. Following issues can be configured in the configuration file:

- location and name of keyword file
- location of *.gz files
- location of the directory where to store the XML files
- errors which are to be ignored

The user ensures by adjusting the configuration file, that the XML file is placed in the correct 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 which increases with each new XML file.

5 Implementation

In this chapter some aspects concerning the implementation of the design described previously are elaborated. Problems which had to be faced and corresponding solutions are highlighted.

During the software development the object-oriented programming paradigms were mostly followed. The goal was to write independent modules which can be reused for similar purposes. The development can be referred to as “agile” software development. In this case “agile” means being flexible in all directions. This software development technique follows the principles (manifesto) listed below.

- Individuals and interactions over processes and tools [23]
- Working software over comprehensive documentation [23]
- Customer collaboration over contract negotiation [23]
- Responding to change over following a plan [23]

Considering these principles, the customers satisfaction and the aim to secure the robustness of the software product have a high priority. Changes are accepted at any time. Many methods are hidden behind the idea of agile software development . This project used more or less the software management method scrum. Scrum concentrates more on the execution process. Having regular meetings and setting certain scopes each time, the development process was constantly supervised and changes could be applied immediately. At each meeting a running software product could be presented.

For each application , verified 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 appendix D. The class documentation was created with Doxygen [24] and can be found in appendix E.

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 additional software packages a C compiler is needed. The GNU Compiler Collection is recommended. Note that Tkinter has to be enabled before compilation. Furthermore, the following additional software packages were used to develop the applications and they are required to run the software successfully

- M2Crypto Version 0.13 requires OpenSSL 0.9.7 and SWIG 1.3.2(1,2,3)
- sqlite Version 2.8.16
- pysqlite Version 1.0.1
- module `Graph.py` and `tooltips.py` [25]
- 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 or are installed on UNIX operated systems by default. The script structure of all applications are very similar. Files matching `*classes.py` contain most of the needed classes, files like `utils_*.py` contain additional small functions which are used by multiple classes. The start scripts usually contain the manager class and for the server and client the daemonise 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 to change the `SocketServer.TCPServer` into a secure TCP server was reached by creating a new class which is derived from the `SimpleXMLRPCServer` and `SSL.Server`. The M2Crypto package provides the secure `SSL.Server`. The new class `SimpleSSLXMLRPCServer` is shown in Listing 5.1.

LISTING 5.1: `SimpleSSLXMLRPCServer`

```
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()
```

In the next step 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 its 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. Some pre-written software was used, however, mostly original 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 the ability to define keywords by using a separate file. These keywords are case sensitive. In a first **positive approach** all those keywords were defined which the user might be interested in. As the user does not know what kind of messages or errors he might have to face, chances are high that he might miss important information. On the other hand the user gains the possibility to exclude unimportant information. Due to the chance to overlook information the keyword approach was replaced by a **negative approach** method.

During the initialising process the keywords are saved into an array. The elements of the array can be seen as `OR` combination. Each element of this array can contain two items. These 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 is **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 are passed on to the recursive function `_test_keywords()`. The function processes the array element by element and if a keyword or a keyword combination is detected the function returns 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 such a case the character "–" is taken instead. In next step date and time are extracted. Unfortunately, the log file entry does not contain any information about the 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 process these 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 lose 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 bourne again shell (bash) 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`

```
1  #!/bin/sh
2  #
3  # Script to shutdown client daemon
4  #
5  # by Andrea Weise – December 2005
6  # University of Reading
7  # MSc in Network Centred Computing
8  #
9  echo "stopping client ...."
10
11 name=start_client.py
12
13 # Find all clients
14 client_pid=`ps -elf | egrep $name | egrep -v grep | awk '{ print $4 }'`
```

```
15
16 if [ "$client_pid" = "" ]
17 then
18   echo No client is running !
19 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 defined pattern. In this case all lines from the process table which contain the previous saved name are extracted. 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 the end of Line 14, the 4th value, the process ID, of the extracted line 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 terminates 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 generates an XML structured file. Files which are not well formed will decrease the performance of the server since it is an XML based server. Therefore, the creation of the XML has to ensure the final file is well formed.

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 easily, 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 a potential 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 continuously reorganised and restructured the XML file, which itself increased in size in memory. That slowed down the whole system. This result was unacceptable.

Therefore the XML file creation code was rewritten, using SAX. The SAX implementation of creating an XML file is not as straightforward as DOM's. After finishing this implementation the same test was run again. The application terminated within approximately 10 minutes. SAX streams the data and triggers an event if certain keywords occur. Compared to DOM the amount of memory needed handle the data is relatively small when using SAX.

Since all the allowed characters 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 rerun several times using SAX as well as the system functions. Table 5.1 presents 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

In consequence of these results, the XML creation is finally realised by using the standard system functions. A DTD is not needed because the parser on the client side has a verifying content handler.

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 which is handled by an XML based server. Therefore those “illegal” characters have to be found. Deleting the unwanted characters is not a good option, since it might 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

```
1  def write_entry(self, tagname, content):
2      '''
3      This function inserts an entry into the xml file.
4
5      tagname = tag name
6      content = message between start and end tag
7      '''
8      #find all not allowed character
9      bad_character = re.sub('[\x09\x0a\x0d\x20-\xd7]*', "", content)
10     # replace each not allowed character with "?"
11     for i in range(len(bad_character)):
12         if bad_character[i] == '\x00':
13             # delete NUL character
14             content = content.replace(bad_character[i], '')
15         else:
16             content = content.replace(bad_character[i], "?")
```

```
17
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 in more detail, an introduction about threads is given.

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. To 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 [26].

The SQLite developer assert the software to be “threadsafe”. SQLite uses posix threads on Unix [27]. To 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

```
1  class Mutex:
2      '''
3      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.
4      '''
5      # database lock
6      _db_locked = threading.Lock()
7      # critical section lock
8      _locked = threading.Lock()
9
10     def __init__(self):
11         '''
12         Constructor
13         '''
14         self.writing = 0
15         self._the_thread = 0
16
17     def set_variable(self, threadus):
18         '''
19         set variable writing and the_thread
20         '''
21         Mutex._db_locked.acquire()
22         if self.writing == 0:
23             #set variable
24             self.writing = 1
25             self._the_thread = threadus
```

```
26              Mutex._db_locked.release()
27              return 0
28          else:
29              if (1 != self._the_thread.isAlive()):
30                  # if the thread, which set the variable is dead, reset variable
31                  self.writing = 0
32              Mutex._db_locked.release()
33              return -1
34
35      def reset_variable(self):
36          '''
37          reset variable writing and the_thread
38          '''
39          Mutex._db_locked.acquire()
40          self.writing = 0
41          self._the_thread = 0
42          Mutex._db_locked.release()
43
44      def lock(self):
45          '''
46          This functions acquires the look.
47          '''
48          Mutex._locked.acquire()
49
50      def release(self):
51          '''
52          This function releases the lock.
53          '''
54          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`. With `Tkinter.Tk` the main window is generated. Within the main window a frame is placed. Within the frame widgets such as buttons or listboxes are positioned. 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 the `grid()` manager was mainly used, because this layout manager allows placing of widgets in a very precise way. The handling is straightforward and the layout is simple to conceive. 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.

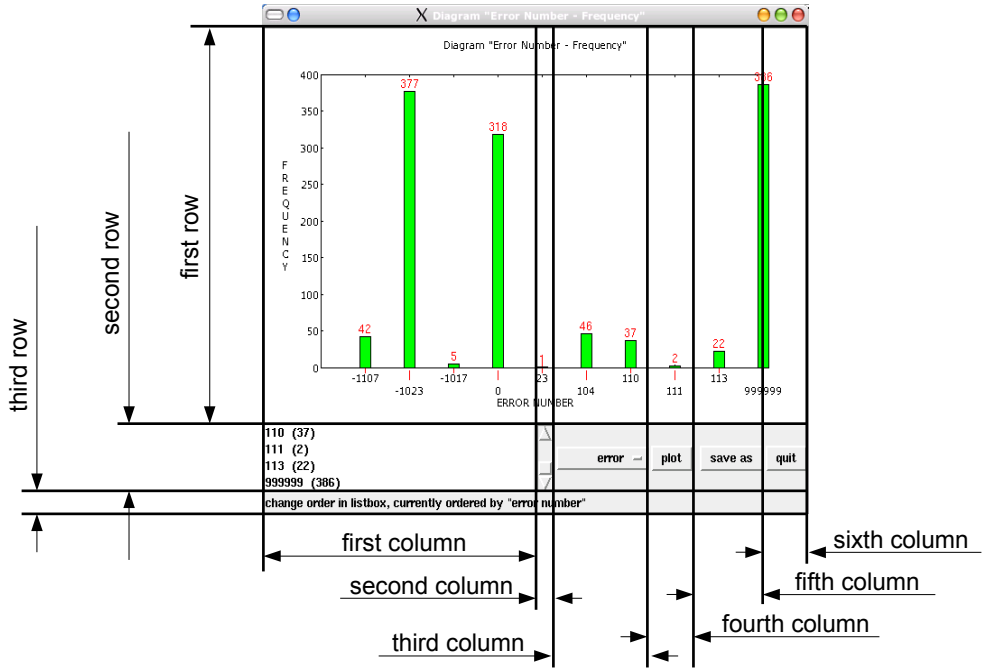

FIGURE 5.1: Grid Layout

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 occurrences. 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.

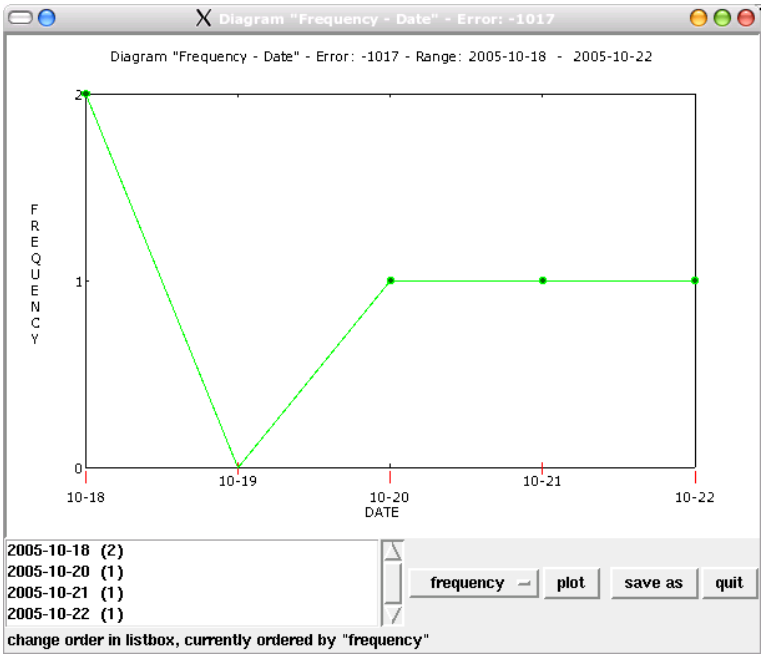

FIGURE 5.2: Particular Error as Line Diagram

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 item chosen . 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 wishes to close the window. Again, the message box has to be closed first in order 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 explain the usage of the widgets and are meant to support new users initially.

The `Picture` class is written in this way, so that it can be used for all graphs. If the

user wants to get more detailed information, 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 graphs, apart from the main window, can be created quickly. 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 [28] 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 results gained 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 of the software evaluation is given.

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

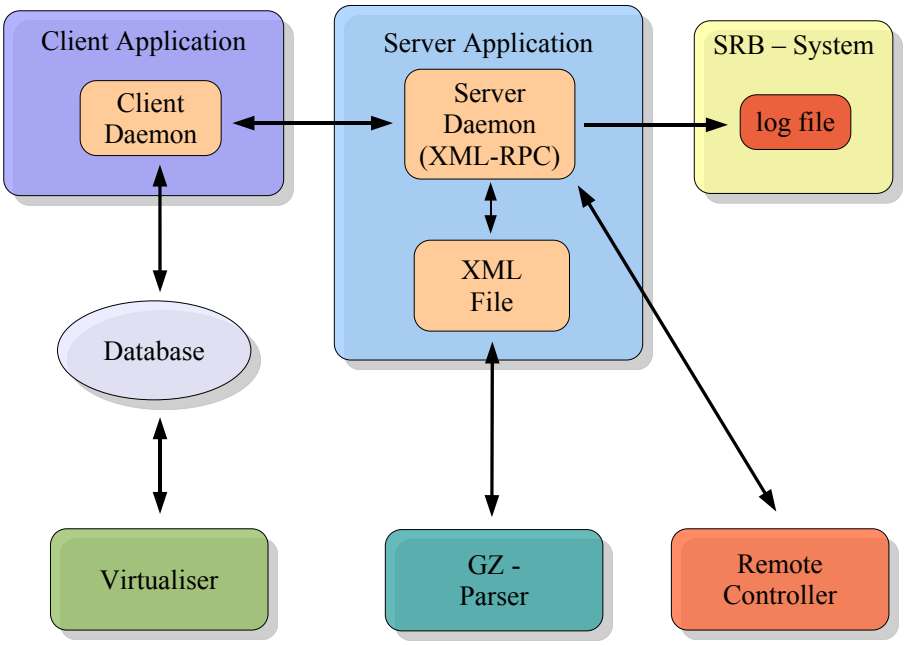

FIGURE 6.1: Application Overview

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, small 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. Furthermore, it was ensured that the program ran at least once 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 efficient the program handles system resources. The resources which can reasonably 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. Here the interest lies in 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 [29] 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 (coppermine)	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. no 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 freely giving up his CPU usage. The other threads did not utilise the CPU according to `ps` which only displays the avarage CPU utilisation of each process. However, it is proved that the sleeping process is not using CPU capacities. This was expected and therefore, implemented this way. Depending on the system properties it took different amounts of time, but none of the systems failed to process 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 zero

keywords 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 [29]. 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 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 this 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 now free disk space and the parser will continue exactly where it 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 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

From Figure 6.4 it can be seen that the 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 efficiently. Misconfiguration of the keyword file such as inserting zero keywords can produce huge XML files.

6.2 Client

For the client the database is the most important issue. The database is only a file and a file is easy to tamper with. Therefore, before the client takes up its real work it checks if the database file exits 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 were 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 tests, as made with the server, were executed. The client application uses more virtual memory as the server but is still using the memory efficiently. The database updating uses approximately 50% of the CPU and database initialising process uses as much as CPU capacity as possible. Large files were handled without problems. This also makes the client a reliable application.

6.3 Other Applications

The Virtualiser queries the database. The query can be specified with different parameters. All parameters, and in combination, were tested successfully. Furthermore, 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 or 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 was inserted with the value 0

All test were successfully executed.

For the Remote Controller as a console application only the parameter, and in combination, were tested extensively. The collaboration was also tested with server which fulfilled the given task successfully.

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

Summarised, it can be said that all applications work reliable. Problems are reported to the user with appropriate advice.

7 Summary

The complex SRB system is used to manage data within a grid environment. To be able to reconstruct the multiple activities within the system, one log file is maintained. Since every process is appending data into this file, the log file becomes confusing and the evaluation process intricate. The main goal of this project was to develop tools which bring designated clarity into the log file to ease the SRB system administration and debugging process. With the SRB system comes a distributed network which demands a whole monitoring system.

Monitoring systems are well established as performance measurement tools. Therefore a wide range of network tools are available already *e.g.* the Netmon [30] or Ganglia system [31]. Those applications provide very good statements about the network activities. However, this project requires mechanisms to monitor a log file. This implies a log file parsing where for this project emphasis had to be put on the error messages. A literature review and Internet research found that many parsers exist. Some were described in Chapter 3. These parsers are often specialised on a certain pattern. What was needed, was a parser which can be configured for any pattern. Due to this flexibility, a compact parser was developed which was integrated into a server based application. The project required the possibility to monitor several SRB servers simultaneously. Since the SRB system works across a network, the tools have to operate through a network as well, what led to the development of a client-server-application. This architecture is adequate for this task, because the time-consuming parsing and evaluation of the log file is executed by the server. The server provides the analysed data for further processing. The corresponding client collects the gathered information from the server, whereas several servers can be addressed at the same time. In the end the client unites the log file data from a whole SRB system.

The SRB system holds huge amounts of data and contains certain security already. The monitoring system should not provide a security risk. Consequently state-of-the-

art security mechanisms were implemented. The communication with the server is encrypted by using SSL technology.

The SRB system administrator also needed the ability to search within the collected data and generate statistics, respectively. A databases is a modern, common technology to hold data and provides an efficient way to return information according to certain conditions. A few database systems were analysed for this project (Chapter 3). The data which has to be stored is not complex, therefore a light database was chosen which brings performance and administration benefits. The client maintains a database for saving the collected information. Server and client work independently and if daemonised, invisibly in the background. The possibility to run the client as a daemon, eliminated the option to use the client as a database query tool. The visualisation of the database content was sourced out to an individual application.

The Visualiser does not have to be network-compatible, since the client monitors several servers and stores the data in one database only. The application has to be able to provide any required data in a perspicuous way. The user can realise this by forming combinations of parameters which the Visualiser provides. Statistics can be viewed as graphs which are embedded in a graphical user interface. The GUI enables the user to save the graph for further processing. In addition, the error occurrence can be zoomed in up to a 24 hours overview. The zoom existence in conjunction with parameter combinations allows the user to gain a distinct view of specific error data. This is not provided by the SRB system.

Working with technology which operates across networks implies that one can not be always locally present. Therefore the project indicated indirectly the need of a possibility to control the parsing process remotely. This was accomplished by developing the Remote Controller application. This application enables the user to influence the parsing behaviour of the server in many ways.

An administrator wants to know what happens with the system which was handed over to him for safekeeping. Therefore, the notification of an important event might be a favoured feature. The developed system is able to notify the administrator by occurrences of defined errors utilising the latest commonly used technology email.

The usage of Python proved a good choice. The substantial standard library provided almost all appliances needed, to implement the monitoring system. Combined with the object-oriented programming approach, modules were produced which are easy to reuse.

8 Conclusions and Future Work

In this dissertation, the development of a highly configurable and scalable monitoring system for Storage Resource Brokers is presented. The targeted flexibility of the software is achieved. The adaptiveness makes the system independent from the SRB system. Consequently, any system which maintains an ASCII log file can be supervised with the framework developed within this project.

The formed system, composed of five individual applications, allows to monitor SRB systems across a network. Valuable time and therefore costs are saved as well as additional flexibility for the SRB system administrator is given through the possibility to control the log file evaluation remotely. Further support is provided through email notification.

Performance and software robustness tests have proved that the presented framework is a stable and reliable software package which handles different log file sizes as well as various types of log file patterns. Statistics and error occurrence overviews can be easily created and stored for further processing. This is supported by a graphical user interface developed within this project.

The provided tools will support the SRB administration and ease the SRB system evaluation by identifying problems within the SRB system quickly and therefore, will maintain a robust data grid management system.

For future work it is planed to integrated this monitoring system into the next SRB development.

Extensive and advanced diagrams and graphs might be preferred. To ease the implementation process the `matplotlib` library can be used as already analysed in Chapter 3.7.

Since firewalls influence the work of the monitoring system a trade-off between network security policies and usability have to be found. To avoid opening the firewall and still being able to run the software, HTTP tunnelling is a potential and promising solution, since HTTP connections are usually allowed. The developed software could be extended, so that the connections are made, using a proxy server. The incorporation of the module `pytunnel.py`[1], that tunnels TCP/IP through another server, is a possible a solution.

In the long term, another approach to monitor the SRB servers could be conceivable. The unification of the system in once place, but accessing the data from any remote workstation will bring even more flexibly. This might be achieved by using further web technologies such as web services, or the incorporation of a web server.

[1] http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/213238

References

[1] Grid-Computing. http://www.nationmaster.com/encyclopedia/Grid_computing, access date: 12.0.12006.

[2] Council for the Central Laboratory of the Research Councils (CCLRC). http://www.cclrc.ac.uk/Activity/WhoWeAre, access date: 27.01.2006.

[3] CCLRC e-Science Centre. http://www.e-science.clrc.ac.uk/web, access date: 27.01.2006.

[4] SDSC Storage Resource Broker. http://www.sdsc.edu/srb, last access date: 17.01.2006.

[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, access date: 10.01.2006.

[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, access date: 23.01.2006.

[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, access date: 20.09.2005.

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

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

[13] Python. http://www.python.org, access date: 01.03.2006.

[14] P. McGuire. pyparsing – an object-oriented approach to text processing in Python. http://pyparsing.sourceforge.net/, access date: 02.10.2005.

[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, access date: 05.10.2005.

[17] M. Sjögren. Python OpenSSL Manual. Technical report, Release 0.6., access date: 29.09.2006. 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, access date: 29.09.2005.

[20] About SQLite. http://www.sqlite.org, access date: 02.11.2005.

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

[22] J. E. Grayson. Python and Tkinter Programming. Manning, access date: 14.12.2005.

[23] Manifesto for Agile Software Development. http://www.agilemanifesto.org, access date: 18.10.2005.

[24] Doxygen. http://www.stack.nl/ dimitri/doxygen, access date: 19.02.2006.

[25] M. Lange. Tooltip. http://tkinter.unpythonic.net/wiki/ToolTip, Juli 2004.

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

[27] SQLite Threading. http://sandbox.rulemaker.net/ngps/149, access date: 15.12.2005.

[28] ANSI CODES. http://rrbrandt.dyndns.org:60000/docs/tut/redes/ansi.php, access date: 10.12.2006.

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

[30] Introducing Netmon, an all-in-one network monitoring solution. http://www.netmon.ca, access date: 03.03.2006.

[31] Ganglia Monitoring System. http://ganglia.info, March 2006.

[32] 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.

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

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

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

[36] J. Plate and J. Holzmann. Sichere Protokolle. http://www.inf-wiss.uni-konstanz.de/CURR/summer00/ec/sicher.html, access date: 29.09.2005.

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

[38] DOM - Document Object Model. http://www.w3.org/TR/REC-DOM-Level-1/introduction.html, access date: 25.10.2005.

[39] Official Website for SAX. http://www.saxproject.org/, access date: 25.10.2005.

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

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

[42] About pysqlite. http://initd.org/tracker/pysqlite/wiki/About, access date: 02.11.2005.

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

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

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

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

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

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

[49] 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.

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

[51] fabFORCE.net. DB Designer 4. http://fabforce.net/dbdesigner4, access date: 27.10.2005.

[52] ActiveState. Komodo. http://www.activestate.com, access date: 04.03.2006.

[53] B. Dufour. A Comprehensive Introduction to Python Programming. Technical report, McGill Univeristy, access date: 10.12.2005.

[54] SRB Workshop. http://www.sdsc.edu/srb/Workshop, access date: 23.02.2006.

[55] K. Schwaber. SCRUM - It's About Common Sense. http://www.controlchaos.com, access date: 07.12.2005.

[56] Informal Language Comparison Chart(s). http://www.smallscript.org, access date: 17.01.2006.

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

[58] MySQL. http://www.mysql.de, access date: 27.11.2005.

[59] PostgreSQL. http://www.postgresql.org, access date: 27.11.2005.

Appendix A

Development Environment

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

- DB Designer 4.0.4.9 Beta [51]
- SuSE Linux 9.2
- Komodo 3.2 Trail [52]
- gcc 3.3.4
- egrep 2.5.1
- GNU Awk 3.1.4
- GNU bash 3.00.0(1)
- Python 2.2.3 [13]
- LaTeX2ε
- M2Crypto 0.13 [19]
- SQLite 2.8.16 [20]
- pysqlite 1.0.1 [42]
- TeX*nicCenter* 1 *Beta* 6.31[1]
- MiKTeX 2.4[2]
- Umbrello
- doxygen 1.4.6[3]

[1] http://www.toolscenter.org
[2] http://www.miktex.org
[3] http://www.stack.nl/~dimitri/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 (coppermine)	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

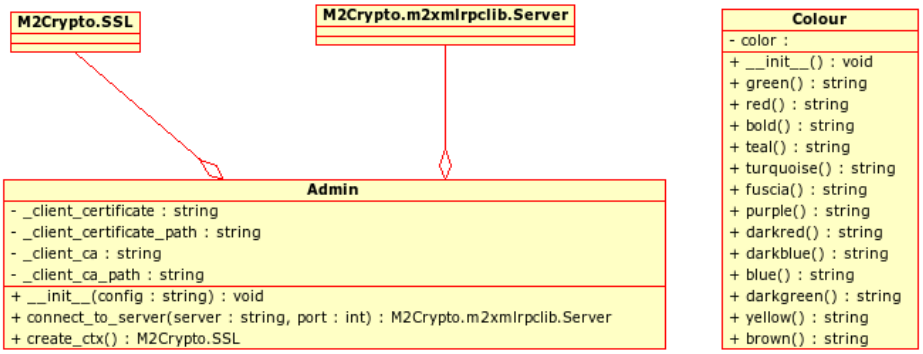

FIGURE B.1: Remote Controller Class Diagram

B.2 Server

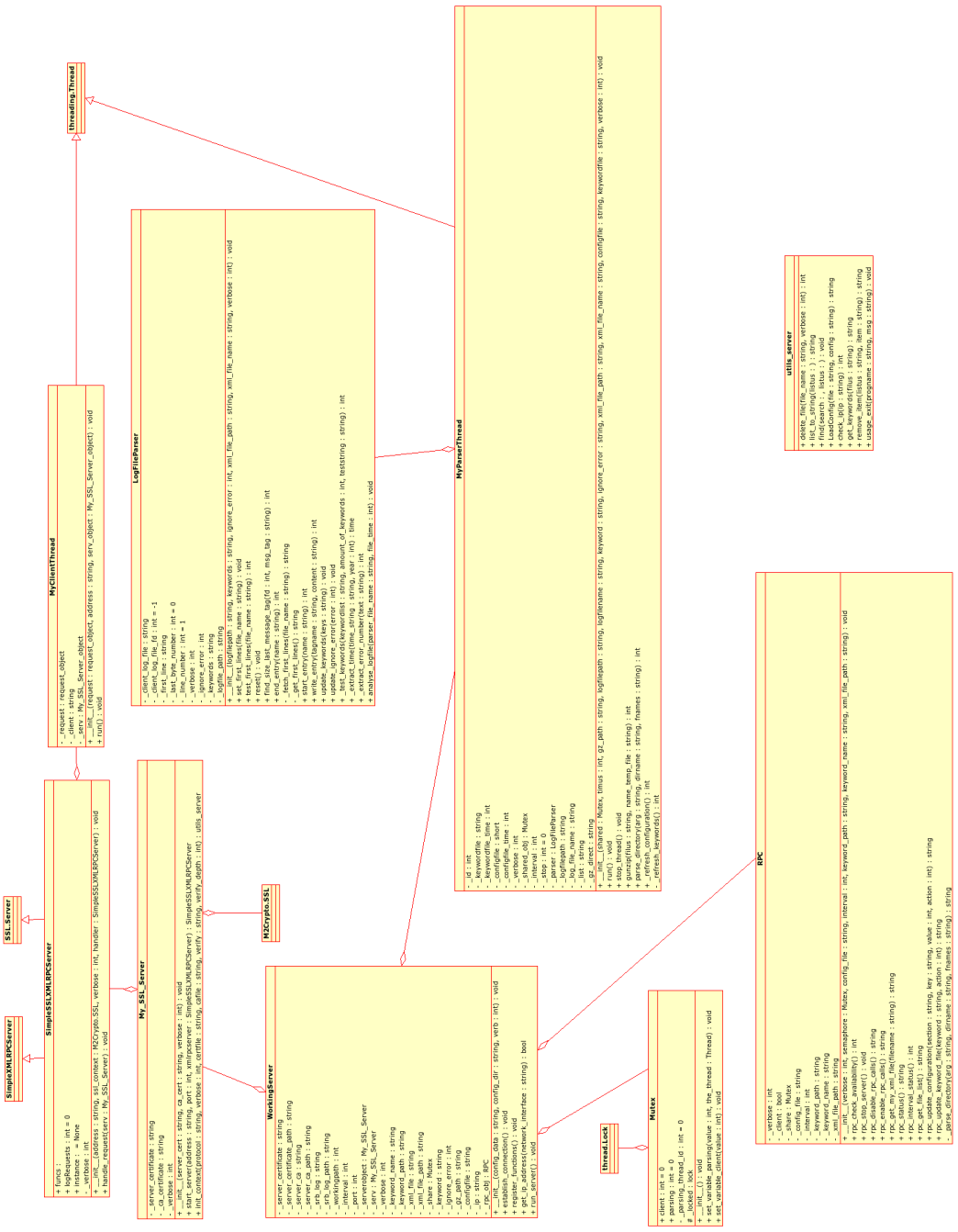

FIGURE B.2: Server Class Diagram

B.3 Client

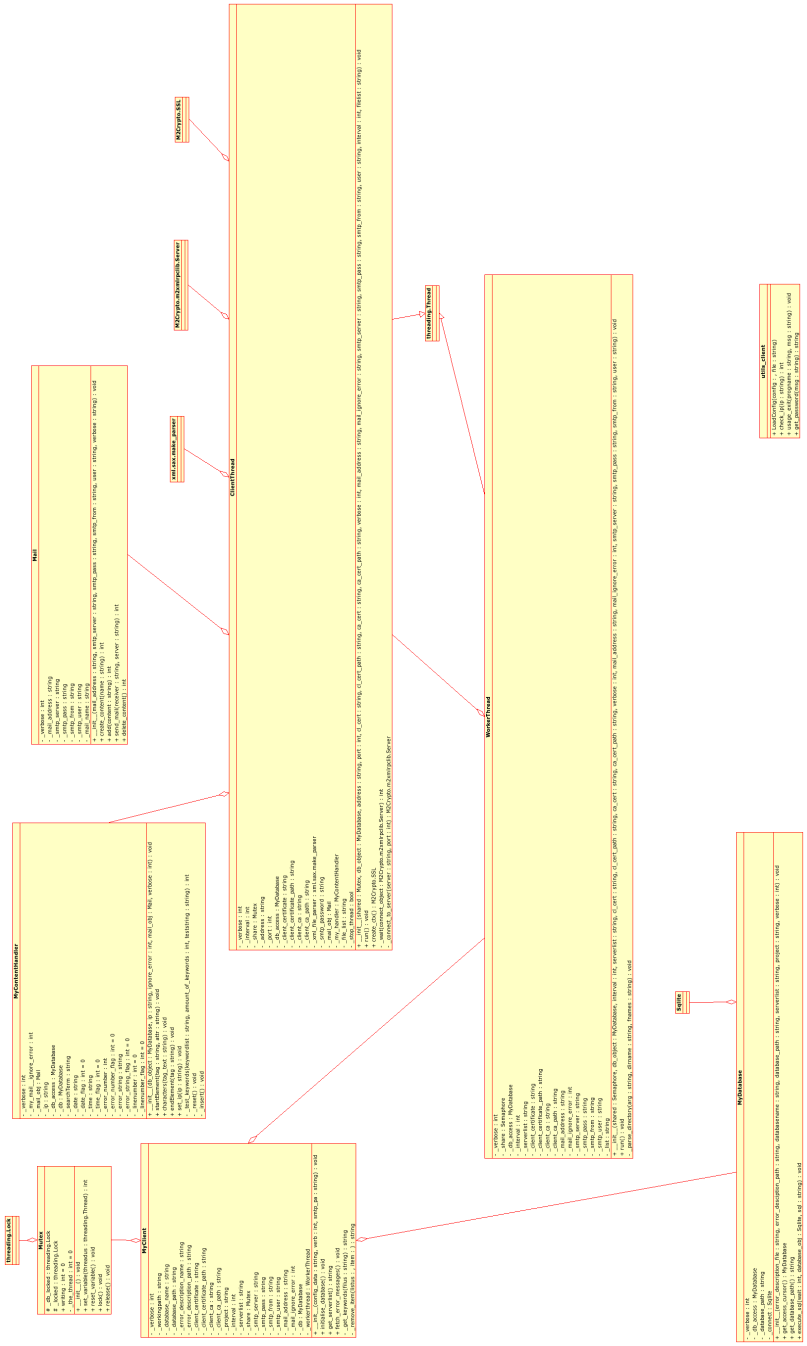

FIGURE B.3: Client Class Diagram

B.4 Virtualiser

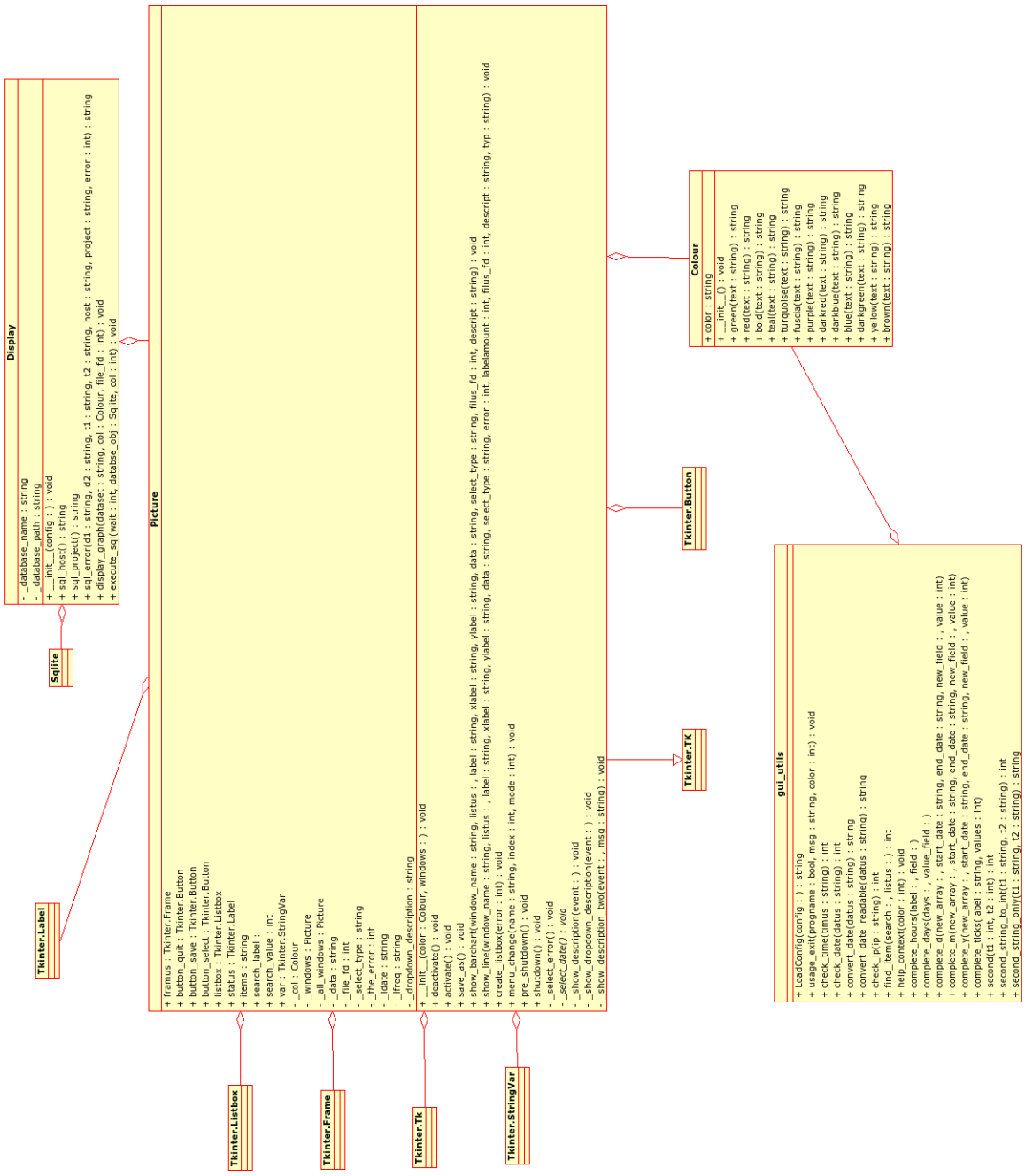

FIGURE B.4: Virtualiser Class Diagram

Appendix C

Software User Manuals

C.1 Introduction

The user manuals give assistance in software usage. All applications are equipped with a substantial help. Before each application is described more detailed, a short installation guide for the required additional software is given.

C.2 Installation

All software package have they own installation guide. The listed steps are meant to be as a quick setup only.

C.2.1 M2Crypto 0.13

M2Crypto is a Python interface to OpenSSL.

Installation steps:

```
$ unzip m2crypto-0.13.zip
$ cd m2crypto-0.13
$ python setup.py build
# python setup.py install
```

or if you want to install it locally

```
$ python setup.py install
  --home=/home/yourhome/yourpath_to_the_desired_folder
$ cd tests # optional
$ python alltests.py # optional
```

C.2.2 sqlite 2.8.16

SQLite is a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine.

Installation step:

```
$ tar -xzf sqlite-2.8.16.tar.gz
$ cd sqlite-2.8.16
$ ./configure
$ make
# make install
```

or if you want to install it locally

```
$ ./configure --prefix=
  /home/yourhome/yourpath_to_the_desired_folder
$ make
$ make install
```

C.2.3 pysqlite 1.0.1

Pysqlite is a Python DB-API 2.0 interface for the SQLite embedded relational database engine.

Installation steps:

```
$ tar -xzf pysqlite-1.0.1.tar.gz
$ cd pysqlite
$ python setup.py build
# python setup.py install
```

or if you want to install it locally

```
$ python setup.py install
  --home=/home/yourhome/yourpath_to_the_desired_folder
```

You might have to export the python path, if you use the local installation.

```
$ export PYTHONPATH=
  /home/yourhome/yourpath_to_the_desired_folder/lib/Python
```

C.3 Server

The Server parses the SRB log file and is a console application, 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 if `-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 also contains basic examples.

Before the server can be started by using

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

the configuration file has to be adjusted.

C.3.1 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 $\Rightarrow$ 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.3.2 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.4 Client

The client fetches 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 parameters displayed in Table C.3.

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 if `-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 also contains basic examples. 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 which may take a few seconds. If the test fails, the client will terminate.

To run the client the command

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

is used, but the configuration file has to be adjusted first.

C.4.1 Configure the Client

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

TABLE C.4: Configuration File Client

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
minute	how often should the client fetch the XML file from server in minutes

Continued on next page

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

Parameter	Explanation
	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 1st person
file_keyword_1	file where keywords are defined
path_keyword_1	path of keyword file for 1st person
address_2	email address of the 2nd person
file_keyword_2	file where keywords are defineend
path_keyword_2	path of keyword file for 2nd 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.4.2 Examples

For a better understanding 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.5 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 the 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)
additional parameters	

Continued on next page

Table C.5 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)

The database commands can only be used individually. Additional commands can be combined to define a certain interesting range or errors, 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 fade in. The created graph can be saved as a postscript file. A dialog leads 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 required information.

Figure C.1 depicts the main window.

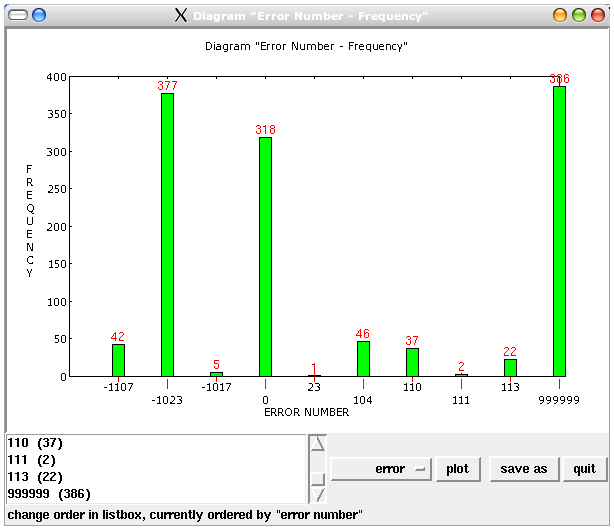

FIGURE C.1: Main Window

The bar chart graph shows all errors. The error number is used as the x-axis description. The error frequency is displayed 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 is required, a error can be selected within the listbox. The button “plot” will then generate a new window like that displayed in figure C.2. The order within the listbox can be changed with the dropdown menu. The listbox can be ordered by error number or error frequency.

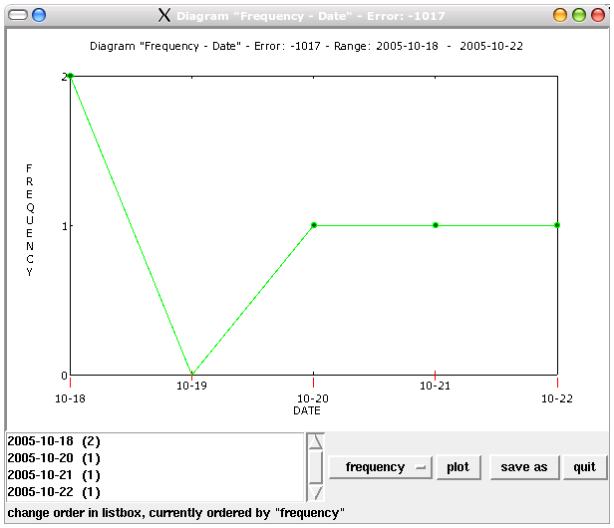

FIGURE C.2: Error Window I

The design of the window is similar to the main window. The line diagram displayed 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.

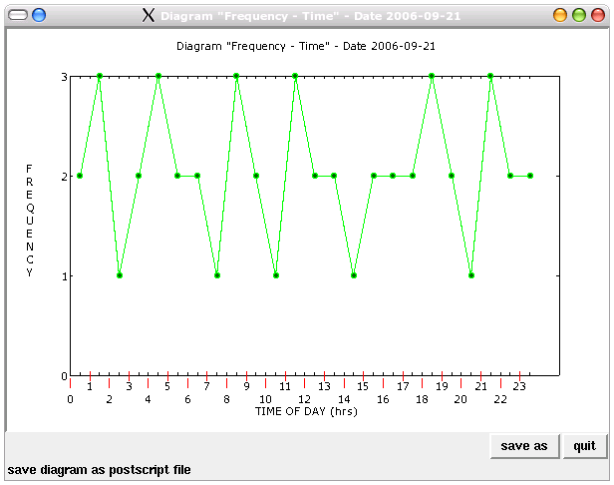

FIGURE C.3: Error Window II

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

C.5.1 Examples

For a better understanding 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.6 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
–interval_status	show status of parsing interval
–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 one at a time. The additional commands can be combined. The necessary configuration file contains only the 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 is started the required action is executed and the server's answer is displayed. The application terminates after the task is processed.

C.6.1 Examples

For a better understanding 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.7 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 noted that the XML file path is very important. Only if the application places the XML file within the server XML directory, the client can fetch the files.

Appendix D

Source Code

D.1 Server

D.1.1 Module `start_server.py`

LISTING D.1: Module `start_server.py`

```
1  #!/usr/bin/env python
2
3
4  '''
5  This file is the log file parser server start file.
6
7  Reading University
8  MSc in Network Centered Computing
9  a.weise - a.weise@reading.ac.uk - December 2005
10 '''
11
12 import server_classes
13 import sys, os, time, getopt
14 import socket
15 import fcntl
16 import struct
17
18 from utils_server import LoadConfig, check_ip, usage_exit
19
20
21
22
23 class WorkingServer:
24     '''
25     This is the main class for the server.
26     '''
```

```
27
28      def __init__(self, config_data, config_dir, verb):
29          '''
30          Constructor
31          '''
32          self._verbose = verb
33          config = config_data
34          self._workingpath = os.getcwd()
35
36          try:
37              self._server_certificate = config.get("files.server_certificate")
38              self._server_certificate_path = config.get("path.path_server_certificate"
                    )
39              self._server_certificate_path = self._server_certificate_path.rstrip("/")
40              if(self._server_certificate_path == '' or self._server_certificate_path
                    == None):
41                  self._server_certificate_path = self._workingpath
42              else:
43                  self._server_certificate = self._server_certificate.strip()
44                  if (-1 != self._server_certificate_path.find("/", 0, 1)):
45                      # first character "/"
46                      pass
47                  else:
48                      self._server_certificate_path = self._workingpath+"/"+self.
                            _server_certificate_path
49
50              self._server_ca = config.get("files.server_ca")
51              self._server_ca_path = config.get("path.path_server_ca")
52              self._server_ca_path = self._server_ca_path.rstrip("/")
53              if(self._server_ca_path == '' or self._server_ca_path == None):
54                  self._server_ca_path = self._workingpath
55              else:
56                  self._server_ca = self._server_ca.strip()
57                  if (-1 != self._server_ca_path.find("/", 0, 1)):
58                      # first character "/"
59                      pass
60                  else:
61                      self._server_ca_path = self._workingpath+"/"+self._server_ca_path
62
63              self._srb_log = config.get("files.srb_log")
64              self._srb_log_path = config.get("path.path_srb_log")
65              self._srb_log_path = self._srb_log_path.rstrip("/")
66              if(self._srb_log_path == '' or self._srb_log_path == None):
67                  self._srb_log_path = self._workingpath
68              else:
69                  self._srb_log = self._srb_log.strip()
70                  if (-1 != self._srb_log_path.find("/", 0, 1)):
71                      # first character "/"
72                      pass
73                  else:
74                      self._srb_log_path = self._workingpath+"/"+self._srb_log_path
75
```

```
76              self._gz_path = config.get("path.path_gz")
77              self._gz_path = self._gz_path.rstrip("/")
78              if(self._gz_path == '' or self._gz_path == None):
79                  self._gz_path = self._workingpath
80              else:
81                  if (-1 != self._gz_path.find("/", 0, 1)):
82                      # first character "/"
83                      pass
84                  else:
85                      self._gz_path = self._workingpath+"/"+self._gz_path
86
87              self._keyword_name = config.get("files.keyword")
88              self._keyword_path = config.get("path.path_keyword")
89              self._keyword_path = self._keyword_path.rstrip("/")
90              if(self._keyword_path == '' or self._keyword_path == None):
91                  self._keyword_path = self._workingpath
92              else:
93                  self._keyword_name = self._keyword_name.strip()
94                  if (-1 != self._keyword_path.find("/", 0, 1)):
95                      # first character "/"
96                      pass
97                  else:
98                      self._keyword_path = self._workingpath+"/"+self._keyword_path
99
100             self._xml_file = "client_log.xml"
101             self._xml_file_path = os.getcwd()+"/xml_client"
102
103             try:
104                 self._interval = int(config.get("misc.minute"))
105             except ValueError:
106                 print "Please check the configuration in the config file (section:
                        misc, item: minute). It should have the following pattern:\
                        nminute = <int>"
107                 os._exit(-1)
108             try:
109                 self._port = int(config.get("misc.port"))
110             except ValueError:
111                 print "Please check the configuration in the config file (section:
                        misc, item: port). It should have the following pattern:\nport =
                        <int>"
112                 os._exit(-1)
113             if (self._port < 1024 or self._port > 50001):
114                 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"
115                 os._exit(-1)
116
117              #check if the configuration if correct
118             if(0 == os.path.exists(self._server_certificate_path+"/"+self.
                    _server_certificate)):
119                 print "Could not locate server certifiate under %s !\nMaybe change
                        configuration file and try again!\n\n" % self.
```

```
                _server_certificate_path
120             os._exit(-1)
121
122         if(0 == os.path.exists(self._server_ca_path+"/"+self._server_ca)):
123             print "Could not locate server ca certificate under %s !\nMaybe
                    change configuration file and try again!\n\n" % self.
                    _server_ca_path
124             os._exit(-1)
125
126         if(0 == os.access((self._srb_log_path+"/"+self._srb_log), 4)):    # 4
                R_OK
127             print "Could not access SRB server log file under %s !\nMaybe change
                    configuration file and try again!\n\n" % self._srb_log_path
128             os._exit(-1)
129
130         if(0 == os.path.exists(self._xml_file_path)):
131             print "Creating path \"%s\"\n\n" % self._xml_file_path
132             os.mkdir(self._xml_file_path)
133
134         self._share = server_classes.Mutex()
135
136         self._keyword = server_classes.get_keywords(self._keyword_path+"/"+self.
                _keyword_name)
137
138         error = config.get("misc.ignore_error")
139
140         self._ip = config.get("misc.interface")
141
142         error = error.strip()
143         error = error.strip(",")
144         self._ignore_error = error.split(",")
145         for i in range(len(self._ignore_error)):
146             if self._ignore_error[i] != '':
147                 try:
148                     self._ignore_error[i] = int(self._ignore_error[i].strip())
149                 except ValueError:
150                     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> "
151                     os._exit(-1)
152             else:
153                 del self._ignore_error[i]
154
155         self._configfile = config_dir
156
157         self._rpc = server_classes.RPC(self._verbose, self._share, self.
                _configfile, self._interval, self._keyword_path, self._keyword_name,
                self._xml_file_path)
158
159     except:
```

```
160             print "\nPlease check to configuraton file, some required information
                    seems to be missing or invalid !\n"
161             os._exit(0)
162
163     def establish_connection(self):
164         '''
165         establish a working connection using MySSLServer
166         '''
167         cert = self._server_certificate_path+"/"+self._server_certificate
168         ca = self._server_ca_path+"/"+self._server_ca
169         self._serverobject = server_classes.My_SSL_Server(cert, ca, self._verbose)
170         ip = self.get_ip_address(self._ip)
171         if (0 == check_ip(ip)):
172             self._serv = self._serverobject.start_server(ip, self._port)
173         else:
174             print "Could not start the server. (IP: \"%s\")" % ip
175             os._exit(-1)
176
177         # start thread for parsing log file
178         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)
179         workerthread.setName("parser")
180         workerthread.start()
181         print "Started!\n\n"
182
183     def get_ip_address(self, network_interface):
184         '''
185         Uses the Linux SIOCGIFADDR ioctl to find the IP address associated with a
                network interface, given the name of that interface, e.g. "eth0".
186
187         source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/439094
188
189         modified by a.weise (December 2005)
190         '''
191         s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
192         try:
193             ip = socket.inet_ntoa(fcntl.ioctl(
194                 s.fileno(),
195                 0x8915,  # SIOCGIFADDR
196                 struct.pack('256s', network_interface[:15])
197             )[20:24])
198             return ip
199         except IOError, e:
200             return e
201
202     def register_functions(self):
203         '''
204         register all the rpc - functions
205         '''
206         self._serv.register_function(self._rpc.rpc_stop_server, 'stop_server')
```

```
207         self._serv.register_function(self._rpc.rpc_status, 'rpc_status')
208         self._serv.register_function(self._rpc.rpc_disable_rpc_calls, '
                disable_rpc_calls')
209         self._serv.register_function(self._rpc.rpc_enable_rpc_calls, '
                enable_rpc_calls')
210         self._serv.register_function(self._rpc.rpc_get_my_xml_file, 'get_my_xml_file'
                )
211         self._serv.register_function(self._rpc.rpc_get_file_list, 'get_file_list')
212         self._serv.register_function(self._rpc.rpc_update_configuration, '
                rpc_update_configuration')
213         self._serv.register_function(self._rpc.rpc_update_keyword_file, '
                rpc_update_keyword_file')
214         self._serv.register_function(self._rpc.rpc_check_availabitity, '
                rpc_check_availabitity')
215         self._serv.register_function(self._rpc.rpc_interval_status, '
                rpc_interval_status')
216
217     def run_server(self):
218         '''
219         handle all client requests
220         '''
221         try:
222             #serv.serve_forever()
223             if self._verbose == 1:
224                 print "\nSimple SSL XML RPC Server is running ....\n"
225             while (1):
226                 if self._verbose == 1:
227                     print "%s -> waiting for request ...." % time.ctime()
228                 self._serv.handle_request(self._serv)
229         except KeyboardInterrupt:
230             # if the server is not running as a daemon shutdown with Ctrl+c is
                    possible
231             if self._verbose == 1:
232                 sys.stdout.write("\n\nShutdown !!!\n\n")
233
234             command = "./stop_server"
235             os.system(command)
236
237
238 ##########################################################################
239
240
241 def daemonize(verbose, stdout = '/dev/null', stderr = None, stdin = '/dev/null',
        pidfile = None, startmsg = 'Server daemon started with pid %s'):
242
243     '''
244     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.
245
```

```
246     source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/66012
247     modified by a.weise November 2005
248     '''
249
250     # first fork => fork creates first child-process
251     try:
252         pid = os.fork()
253         if (pid > 0):
254             sys.exit(0) # close first parent-process
255     except OSError, e:
256         sys.stderr.write("fork #1 failed: (%d) %s\n" % (e.errno, e.strerror))
257         sys.exit(1)
258
259     os.umask(0)
260     os.setsid()
261
262     # second fork
263     try:
264         pid = os.fork()
265         if (pid > 0):
266             sys.exit(0) # close second parent-process
267     except OSError, e:
268
269         sys.stderr.write("fork #2 failed: (%d) %s\n" % (e.errno, e.strerror))
270         sys.exit(1)
271
272     # open standard in and out and print standard message
273     if (not stderr):# if not stderr given => take stdout-path
274         stderr = stdout
275
276     if verbose == 1:
277         si = file(stdin, 'r')
278         so = file(stdout, 'w+') # w -> overwrite old log content
279         se = file(stderr, 'w+', 0)
280         pid = str(os.getpid())
281         sys.stderr.write("\n%s\n" % startmsg % pid)
282         sys.stderr.flush()
283         if pidfile:
284             file(pidfile,'w+').write("%s\n" % pid)
285
286         # redirect standard in and out to files
287         os.dup2(si.fileno(), sys.stdin.fileno())
288         os.dup2(so.fileno(), sys.stdout.fileno())
289         os.dup2(se.fileno(), sys.stderr.fileno())
290
291 ######################################################################
292
293 def start():
294
295     '''
296     START THE APPLICATION
297     '''
```

```
298     configfile = ""
299     verbose = 0
300     daemon = 0
301 
302     try:
303         opts, args = getopt.getopt(sys.argv[1:], 'c:vhd', ['config=', 'verbose', '
            help', 'daemon'])
304         for opt, value in opts:
305             if opt in ('-h','--help'):
306                 msg = "\n\t\t-----------   Help   ----------\n\n\n"\
307                       "-c or --config\t->  defines config file, if no config file
                          given, default values are used\n"\
308                       "-v or --verbose\t->  activates printing of messages [debug
                          option]\n"\
309                       "-d or --daemon\t->  daemonize the server\n"\
310                       "-h or --help\t->  print this help\n\n"
311                 usage_exit(sys.argv[0], msg)
312             if opt in ('-c','--config'):
313                 value = value.replace("=", "")
314                 configfile = os.getcwd()+"/"+value
315             if opt in ('-v','--verbose'):
316                 verbose = 1
317             if opt in ('-d', '--daemon'):
318                 daemon = 1
319     except getopt.error, e:
320         usage_exit(sys.argv[0], e)
321 
322     # load config file or default values
323     if (configfile != ""):
324         # check if file exists
325         if(1 == os.path.exists(configfile)):
326             config = LoadConfig(configfile)
327         else:
328             # if file NOT exists terminate program
329             print "Sorry, the given file does NOT exist !\nPlease try again!\n\n"
330             os._exit(-1)
331     else:
332         msg = "\nNo configuration file spezified !\n"
333         usage_exit(sys.argv[0], msg)
334 
335     print "\n\n---------------------- SRB LOG FILE PARSER [ SERVER ]
          ------------------ \n\n"
336     print "Starting ..."
337 
338     worker = WorkingServer(config, configfile, verbose)
339 
340     if daemon == 1:
341         if verbose == 1:
342             #if verbose then write messages in log file
343             daemonize(verbose, stdout = 'daemonise.log')
344         else:
345             # quit mode
```

```
346             daemonize(verbose)
347     else:
348         pass
349
350     worker.establish_connection()
351     worker.register_functions()
352     worker.run_server()
353
354
355 if __name__ == '__main__':
356
357     start()
```

D.1.2 Module `server_classes.py`

LISTING D.2: Module `server_classes.py`

```
1  #!/usr/bin/env python
2
3  '''
4  This module contains all necessary classes and functions for the gz_parser.py and srb
       log file parser -> start_server.py .
5
6  Reading University
7  MSc in Network Centered Computing
8  a.weise - a.weise@reading.ac.uk - December 2005
9  '''
10
11 # ssl connection
12 from SimpleXMLRPCServer import SimpleXMLRPCServer, SimpleXMLRPCRequestHandler
13 from M2Crypto import SSL
14
15 # misc
16 import os, time, stat
17
18 # regular expressions
19 import re
20
21 # utilities
22 from utils_server import delete_file, list_to_string, get_keywords, LoadConfig, find
23
24 # threads
25 import thread
26 import threading
27
28 import socket
29
30 ################## CLASS  SimpleSSLXMLRPCServer  ##################
31
```

```
32 class SimpleSSLXMLRPCServer(SSL.SSLServer, SimpleXMLRPCServer):
33     '''
34     This class is derived from SSL.SSLServer and SimpleXMLRPCServer.
35     '''
36     def __init__(self, ssl_context, address, verbose, handler=
           SimpleXMLRPCRequestHandler):
37         '''
38         Constructor overwrites the init function of the SimpleXMLRPCServer and
               replace it with the secure SSLServer.
39         '''
40         SSL.SSLServer.__init__(self, address, handler, ssl_context)
41         self.funcs = {}
42         self.logRequests = 0
43         self.instance = None
44         self._verbose = verbose
45 
46     def handle_request(self, serv):
47         '''
48         Handle one request by passing it on the a thread.
49         '''
50         try:
51             request, client_address = self.get_request()
52             if self._verbose == 1:
53                 print "%s -> request accepted from %s....." % (time.ctime(),
                       client_address[0])
54 
55         except socket.error:
56             return
57         if self.verify_request(request, client_address):
58             thd = MyClientThread(request, client_address, serv)
59             thd.start()
60 
61 ################## CLASS  My_SSL_Server  ##################
62 
63 class My_SSL_Server:
64     '''
65     provide functions for the server class
66     '''
67 
68     def __init__(self, server_cert, ca_cert, verbose):
69         '''
70         Constructor
71         '''
72         self._server_certificate = server_cert
73         self._ca_certificate = ca_cert
74         self._verbose = verbose
75 
76     def start_server(self, address, port, xmlrpcserver=SimpleSSLXMLRPCServer):
77         '''
78         Start the actual server using SSL:
79 
```

```
80          sslv23 -> compatibility mode, can handle any of the three SSL/TLS protocol
                versions
81          server.pem -> server certificate including server RSA private key
82          ca.pem -> root certificate
83          SSL.verify_none -> no request that the client has to send his certificate as
                well
84          '''
85          # create SSL context
86          ctx = self.init_context('sslv3', self._verbose, self._server_certificate,
                self._ca_certificate, SSL.verify_none)
87          # create server object
88          server = xmlrpcserver(ctx, (address, port), self._verbose)
89          # return server object
90          return server
91
92      def init_context(self, protocol, verbose, certfile, cafile, verify, verify_depth
            =10):
93          '''
94          This function is used to generate the SSL context:
95          - verify_depth -> chain depth
96          '''
97          ctx = SSL.Context(protocol)   # create context object
98          ctx.load_cert_chain(certfile)   # load server certificate chain
99          ctx.load_verify_locations(cafile)
100         ctx.set_client_CA_list_from_file(cafile)
101         ctx.set_verify(verify, verify_depth)  # verfiy options
102         ctx.set_session_id_ctx('server')        # set session id
103         #if verbose == 1:
104         #    ctx.set_info_callback()    # show handshake information --- debug
105         return ctx
106
107 ################## CLASS  MyClientThread  ##################
108
109 class MyClientThread(threading.Thread):
110     '''
111     This class presents a client, which connects to the server.
112     '''
113
114     def __init__(self, request, address, serv_object):
115         '''
116         Constructor
117         '''
118         self._request = request
119         self._client = address
120         self._serv = serv_object
121         threading.Thread.__init__(self)
122
123     def run(self):
124         '''
125         This function overrides the standard run method.
126         '''
127         try:
```

```
128             self._serv.process_request(self._request, self._client)
129             self._serv.close_request(self._request)
130         except:
131             self._serv.handle_error(self._request, self._client)
132             self._serv.close_request(self._request)
133 
134 ################## CLASS  LogFileParser  ##################
135 
136 class LogFileParser:
137     '''
138     This class provides all the neccessarey tools to parse and work up to logfile of
            the SRB-System.
139     '''
140 
141     def __init__(self, logfilepath, keywords, ignore_error, xml_file_path,
            xml_file_name, verbose):
142         '''
143         Constructor
144         '''
145         self._verbose = verbose
146         self._ignore_error = ignore_error
147         self._keywords = keywords  # keywords
148         self._logfile_path = logfilepath
149         self._client_log_file =  "%s/%s" % (xml_file_path, xml_file_name)  # name and
                path of client log file
150         self._client_log_file_fd = -1       # client log file - file descriptor
151         self._first_line = range(15)   # save first 15 lines of log file
152         self._last_byte_number = 0     # save last byte number which was parsed
153         self._line_number      = 1         #save last line number which was parsed
154 
155         if(0 == os.path.exists(self._logfile_path)):
156             print "Could not locate log file path under %s !\nMaybe change
                    configuration file and try again!\n\n" % self._logfile_path
157             os._exit(-1)
158 
159     def _fetch_first_lines(self, file_name):
160         '''
161         This function returns the first 15 lines from current logfile without saving
                them anywhere.
162         '''
163         try:
164             log_file_fd = open(file_name, "r")
165 
166             listline = range(15)
167             log_file_fd.seek(0) #set cursor on first position
168             for i in range(15):
169                 listline[i] = log_file_fd.readline()
170             log_file_fd.close()
171             return listline
172         except IOError:
173             print "Could not open file -> ", file_name
174 
```

```
175     def set_first_lines(self, file_name):
176         '''
177         This function saves the first 15 lines of the log file into the member
                variable.
178         '''
179         try:
180             log_file_fd = open(file_name, 'r')
181             log_file_fd.seek(0) #set cursor on first position
182             for i in range(15):
183                 self._first_line[i] = log_file_fd.readline()
184             log_file_fd.close()
185         except IOError:
186             if self._verbose == 1:
187                 print "%s -> Could not open file -> \"%s\"" % (time.ctime(),
                        file_name)
188 
189     def get_first_lines(self):
190         '''
191         This function returns the member variable _first_lines.
192         '''
193         return self._first_line
194 
195     def test_first_lines(self, file_name):
196         '''
197         This function compares the first 15 lines of a log file and return 0 if they
                are the same, otherwise -1.
198         '''
199         listline = self._fetch_first_lines(file_name)
200         z = 0
201         while(z<15):
202             if(listline == self._first_line):
203                 z += 1
204             else:
205                 return -1
206 
207         return 0
208 
209     def find_size_last_message_tag(self, fd, msg_tag):
210         '''
211         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.
212 
213         fd = file descriptor of the file
214         msg_tag = message tag to search for
215         '''
216         #set cursor back
217         z = -1
218         fd.seek(0, 2)
219 
```

```
220         while(1):
221             fd.seek(z, 2)
222             tag = fd.read()
223             if (-1 != tag.rfind(msg_tag)):
224                 return z
225             z -= 1
226             if -15 == z:
227                 # message tag was not part of the file
228                 return 0
229 
230     def analyse_log_file(self, parser_file_name, file_time=None):
231         '''
232         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.
233 
234         parser_file_name = file name of the file, which needs to be parsed
235         '''
236         # determine year of paser file, needed for extract time, since log file
              content does not provide a year
237         if file_time != None:
238             tupel = time.gmtime(file_time)
239         else:
240             status = os.stat(parser_file_name)
241             file_time = status[8]
242             tupel = time.gmtime(file_time)
243 
244         pf_year = time.strftime("%Y ", tupel)
245 
246         byte_count = 0
247         interrupt = 0
248         z = 0
249 
250         try:
251             self._client_log_file_fd = file(self._client_log_file, 'r+')
252 
253         except IOError:
254             # create new client log file
255             self._client_log_file_fd = open(self._client_log_file, 'w')
256 
257             xml_header = "<?xml version=\"1.0\" encoding=\"utf-8\" standalone=\"yes
                  \"?>\n"
258             self._client_log_file_fd.writelines(xml_header)
259             self._client_log_file_fd.writelines("<message>\n")
260             self._client_log_file_fd.writelines("</message>\n")
261             self._client_log_file_fd.close()
262             self._client_log_file_fd = file(self._client_log_file, 'r+')
263 
264 
265         # set cursor in file
```

```
266         x = self.find_size_last_message_tag(self._client_log_file_fd, "</message>")
267 
268         self._client_log_file_fd.seek(x, 2)
269         shorten = self._client_log_file_fd.tell()
270         # delete last </message>
271         self._client_log_file_fd.truncate(shorten)
272         self._client_log_file_fd.seek(0, 2)
273 
274         #open log file
275         log_file = parser_file_name
276 
277         try:
278             log_file_fd = open(log_file, "r")
279             log_file_fd.seek(self._last_byte_number)
280 
281         except IOError:
282             if self._verbose == 1:
283                 print "%s -> could not open srb log file -> %s" % (time.ctime(),
                        log_file)
284             return -1
285 
286         if self._verbose == 1:
287             print "%s -> start parsing " % (time.ctime())
288         #starttime = time.time()
289         while(interrupt == 0):
290             # read line
291             content = log_file_fd.readline()
292             if(content == ''):
293                 interrupt = 1
294                 break
295 
296             if 0 == len(self._keywords):
297                 check = 0
298             else:
299                 check = self._test_keywords(self._keywords, len(self._keywords)-1,
                        content)
300 
301             if(0 == check):
302                 # extract error number
303                 error_number = self._extract_error_number(content)
304 
305                 if (None == error_number):
306                     error_number = "-"
307                     temp = ""
308                 else:
309                     temp = int(error_number)
310 
311                 if ( None == find(temp, self._ignore_error)):
312 
313                     line_number_string = "%d" % self._line_number
314                     # delete whitespace
315                     content_ = content.rstrip()
```

```
316
317                     date_time = self._extract_time(content, pf_year)
318
319                     if (date_time == -1):
320                         date_time = ["", ""]
321                         # save current byte count
322                         byte_count = log_file_fd.tell()
323                         back = -1
324                         read = 0
325                         while(1):
326                             try:
327                                 # find time pattern by going back character by
                                        character
328                                 log_file_fd.seek(back, 1)
329                                 read = back*(-1)
330                                 tag = log_file_fd.read(read)
331                                 if (None !=  re.search('^NOTICE: *[A-Z][a-z]{2}
                                        +[0-9]{1,2} +[0-9]{2}:[0-9]{2}:[0-9]{2}:', tag)):
332                                     date_time = self._extract_time(tag, pf_year)
333                                     break
334                                 back = back-1
335                             except IOError:
336                                 # no time available
337                                 date_time[0] = time.strftime("%Y-%m-%d", tupel)
338                                 date_time[1] = time.strftime("%H:%M:%S", tupel)
339
340                                 break
341
342                         #restore byte count
343                         log_file_fd.seek(byte_count)
344
345                     z += 1 # entry counter
346
347                     if -1 == self.start_entry('entry'):
348                         interrupt = 1
349                         break
350                     if -1 == self.write_entry('date', date_time[0]):
351                         interrupt = 1
352                         break
353                     if -1 == self.write_entry('time', date_time[1]):
354                         interrupt = 1
355                         break
356                     if -1 == self.write_entry('error_number', error_number):
357                         interrupt = 1
358                         break
359                     if -1 == self.write_entry('error_string', content_):
360                         interrupt = 1
361                         break
362                     if -1 == self.write_entry('linenumber', line_number_string):
363                         interrupt = 1
364                         break
365                     if -1 == self.end_entry('entry'):
```

```
366                         interrupt = 1
367                         break
368
369             self._line_number += 1
370         if self._verbose == 1:
371             print "%s -> end parsing " % (time.ctime())
372         #endtime = time.time()
373         #if self._verbose == 1:
374         #    print "%s -> parsing time: %s" % (time.ctime(), (endtime-starttime))
375         self.end_entry('message')
376         if self._verbose == 1:
377             print "%s -> %d errors found\n" % (time.ctime(), z) #--- debug ---
378         # save last byte number
379         try:
380             self._client_log_file_fd.close()
381         except IOError, e:
382             if self._verbose == 1:
383                 print "%s -> Problem closing XML file: \"%s\" !" % (time.ctime(), e)
384         self._last_byte_number = log_file_fd.tell()
385
386     def start_entry(self, name):
387         '''
388         This function inserts a start tag into the XML file. (name = tag name)
389         '''
390         start_tag = "<%s>\n" % name
391         try:
392             self._client_log_file_fd.write(start_tag)
393             return 0
394         except IOError, e:
395             if self._verbose == 1:
396                 print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(), e)
397             return -1
398
399     def write_entry(self, tagname, content):
400         '''
401         This function inserts an entry into the xml file.
402
403         tagname = tag name
404         content = message between start and end tag
405         '''
406
407         if len(content) < 50000000:
408             #find all not allowed character  old: [\x09\x0a\x0d\x20-\xd7]*
409             bad_character = re.sub('[\x09\x0a\x0d\x20-\x25\x27-\xd7]*', "", content)
410             # replace each not allowed character with "?"
411             for i in range(len(bad_character)):
412                 if bad_character[i] == '\x00':
413                     # delete NUL character
414                     content = content.replace(bad_character[i], '')
415                 else:
416                     content = content.replace(bad_character[i], "?")
417
```

```
418             entry = "<%s>%s</%s>\n" % (tagname, content, tagname)
419             try:
420                 self._client_log_file_fd.write(entry)
421                 return 0
422             except IOError, e:
423                 if self._verbose == 1:
424                     print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(),
                            e)
425                 return -1
426 
427         else:
428             entry = "<%s>LOGFILE ENTRY TO LONG !!!</%s>\n" % (tagname, tagname)
429             try:
430                 self._client_log_file_fd.write(entry)
431                 return 0
432             except IOError, e:
433                 if self._verbose == 1:
434                     print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(),
                            e)
435                 return -1
436 
437     def end_entry(self, name):
438         '''
439         This function inserts an end tag into the XML file. (name = tag name)
440         '''
441         endtag = "</%s>\n" % name
442         try:
443             self._client_log_file_fd.write(endtag)
444             return 0
445         except IOError, e:
446             if self._verbose == 1:
447                 print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(), e)
448             return -1
449 
450     def reset(self):
451         '''
452         This function resets member variable, in case of a new log file.
453         '''
454         self._line_number = 1
455         self._last_byte_number = 0
456 
457     def _test_keywords(self, keywordlist, amount_of_keywords, teststring):
458         '''
459         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
460 
461         keywordlist = list of all keywords
462         amount_of_keywords = number of keywords in list
463         teststring = string, which needs to be investigated
464 
465         return -1 if line is not interesting
466         return 0 if line is taken
```

```
467         '''
468         if (amount_of_keywords == 0):
469             #last keyword check   -1 != content.rfind("NOTICE")
470             if( 2 == len(keywordlist[amount_of_keywords])):
471                 if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
472                     # not in string go to next keyword
473                     return 0
474                 else:
475                     if( -1 == keywordlist[amount_of_keywords][1].rfind("!")):
476                         # check for NO keyword
477                         temp = keywordlist[amount_of_keywords][1].strip("!")
478                         if ( -1 == teststring.rfind(temp)):
479                             # go on to next keyword
480                             return 0
481                         else:
482                             return -1
483                     else:
484                         # there is no "!"
485                         if ( -1 != teststring.rfind(keywordlist[amount_of_keywords
                                ][1])):
486                             # string is there, go on to next keyword
487                             return 0
488                         else:
489                             return -1
490             else:
491                 if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
492                     # not in string go to next keyword
493                     return 0
494                 else:
495                     return -1
496         else:
497             if( 2 == len(keywordlist[amount_of_keywords])):
498                 if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
499                     # not in string go to next keyword
500                     return self._test_keywords(keywordlist, amount_of_keywords-1,
                            teststring)
501                 else:
502                     if( -1 == keywordlist[amount_of_keywords][1].rfind("!")):
503                         # check for NO keyword
504                         temp = keywordlist[amount_of_keywords][1].strip("!")
505                         if ( -1 == teststring.rfind(temp)):
506                             # go on to next keyword
507                             return self._test_keywords(keywordlist,
                                    amount_of_keywords-1, teststring)
508                         else:
509                             return -1
510                     else:
511                         # there is no "!"
512                         if ( -1 != teststring.rfind(keywordlist[amount_of_keywords
                                ][1])):
513                             # string is there, go on to next keyword
```

```
514                             return self._test_keywords(keywordlist,
                                    amount_of_keywords-1, teststring)
515                         else:
516                             return -1
517             else:
518                 if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
519                     # not in string go to next keyword
520                     return self._test_keywords(keywordlist, amount_of_keywords-1,
                            teststring)
521                 else:
522                     return -1
523
524     def update_keywords(self, keys):
525         '''
526         This function updates the member variable keywords.
527
528         keys = new keyword list
529         '''
530         self._keywords = keys
531
532     def update_ignore_error(self, error):
533         '''
534         This function updates the ignore_error list.
535
536         error = new ignore list
537         '''
538         self._ignore_error = error
539
540     def _extract_time(self, time_string, year):
541         '''
542         This function takes a line from the logfile and extracts the time from there.
543         '''
544         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)):
545             return -1
546         else:
547             listus = time_string.split(":")
548             zeit = year+listus[1]+":"+listus[2]+":"+listus[3]
549             time_tupel = time.strptime(zeit, "%Y %b %d %X")
550             date_time = range(2)
551             date_time[0] = time.strftime("%Y-%m-%d", time_tupel)
552             date_time[1] = time.strftime("%H:%M:%S", time_tupel)
553             return date_time
554
555     def _extract_error_number(self, text):
556         '''
557         Thhis function takes a line from the logfile and extract the error number.
558         '''
559         match = re.search('(status|errno) *= *-*\d{1,10}', text)
560         if(None != match):
561             #if match then give me number
562             listus = match.string[match.start():match.end()]
```

```
563             listus = re.findall('-*\d{1,10}', listus)
564             if(1 == len(listus)):
565                 return listus[0]
566             else:
567                 return None
568         else:
569             return None
570 
571 ################## CLASS  MyParserThread  ##################
572 
573 class MyParserThread(threading.Thread):
574     '''
575     This class is used to create a thread, which is doing all the necessery work in
            the background.
576     '''
577     def __init__(self, shared, timus, gz_path, logfilepath, logfilename, keyword,
            ignore_error, xml_file_path, xml_file_name, configfile, keywordfile, verbose)
            :
578         '''
579         Constructor
580         '''
581         self._keywordfile = keywordfile
582         temp = os.stat(self._keywordfile)
583         self._keywordfile_time = temp[8]
584         self._configfile = configfile
585         temp = os.stat(self._configfile)
586         self._configfile_time = temp[8]
587         self._verbose = verbose
588         self._id = thread.get_ident()
589         self._shared_obj = shared
590         self._interval = timus
591         self._stop = 0
592         threading.Thread.__init__(self)
593         self._parser = LogFileParser (logfilepath, keyword, ignore_error,
                xml_file_path, xml_file_name, self._verbose)
594         self._logfilepath = logfilepath
595         self._log_file_name = logfilepath+"/"+logfilename
596         self._gz_diect = gz_path
597         self._list = [] #save *.gz files
598 
599     def run(self):
600         '''
601         This function overwrites the standard run method.
602         '''
603         block_counter = 0
604         while(1):
605             # acquire lock
606             if self._stop == 1:
607                 print "%s -> working thread stopped !!!" % time.ctime()
608                 os._exit(0)
609             # check if config file has changed
610             if (self._configfile != ""):
```

```
611                 temp = os.stat(self._configfile)
612                 if self._configfile_time != temp[8]:
613                     #if time has changed save new time
614                     if self._verbose == 1:
615                         print "%s -> config file has changed, reading new values ..."
                                % time.ctime()
616                     self._configfile_time = temp[8]
617                     self._refresh_configuration()
618 
619             # check if keyword file has changed
620             if (self._keywordfile != ''):
621                 temp = os.stat(self._keywordfile)
622                 if self._keywordfile_time != temp[8]:
623                     #if time has changed
624                     if self._verbose == 1:
625                         print "%s -> keyword file has changed, reading new values ...
                                " % time.ctime()
626                     self._keywordfile_time = temp[8]
627                     self._refresh_keywords()
628 
629             if(-1 == self._shared_obj.set_variable_parsing(1, self)):
630                 # client is busy
631                 block_counter += 1
632                 time.sleep(5*block_counter)
633                 if self._verbose == 1:
634                     print "%s -> client busy" % time.ctime()
635                 if block_counter > 5:
636                     if self._verbose == 1:
637                         print "%s -> client needs a long time, miss this parsing
                                period" % time.ctime()
638                     block_counter = 0
639                     time.sleep(self._interval*60)
640             else:
641                 # no client busy
642                 try:
643 
644                     # check if the first lines the same
645                     if(0 == self._parser.test_first_lines(self._log_file_name)):
646                         #if the first 15 lines still the same
647                         if self._verbose == 1:
648                             print "%s -> no log file rotation" % time.ctime()
649                         self._parser.analyse_log_file(self._log_file_name)
650                     else:
651                         if self._verbose == 1:
652                             print "%s -> new log file" % time.ctime()
653                         # create gz file list
654                         # empty list for the gz_files
655                         self._list = []
656 
657                         os.path.walk(self._gz_diect, self.parse_directory, self._list
                                )
658                         if (0 < len(self._list)):
```

```
659                             self.gunzip(self._list[0][1])
660                             d = os.getcwd()
661                             try:
662                                 os.chdir(self._gz_diect)
663                                 self.gunzip(self._list[0][1])
664                                 self._parser.analyse_log_file(self._gz_diect+"/
                                        temp_srbLog")
665                                 delete_file("temp_srbLog", self._verbose)
666                                 os.chdir(d)
667                             except:
668                                 if self._verbose == 1:
669                                     print "%s -> could not find directory \"%s\"" % (
                                            time.ctime(), self._gz_diect)
670                         else:
671                             pass
672 
673                         self._parser.reset()
674                         self._parser.set_first_lines(self._log_file_name)
675                         self._parser.analyse_log_file(self._log_file_name)
676 
677                 finally:
678 
679                     # release lock
680                     self._shared_obj.set_variable_parsing(0, self)
681 
682             time.sleep(self._interval*60)#
683 
684             if self._verbose == 1:
685                 print "\n%s -> -------- parse ------" % time.ctime() #--- debug ---
686 
687     def _refresh_keywords(self):
688         '''
689         This function gets keywords from the keyword file !
690         '''
691         if(1 == os.path.exists(self._keywordfile)):
692             keyword = get_keywords(self._keywordfile)
693             self._parser.update_keywords(keyword)
694             return 0
695         else:
696             # if file NOT use old configuration
697             if self._verbose == 1:
698                 print "%s -> Sorry, the keyword file does NOT exist !\nUsing old
                        configuration!\n\n" % time.ctime()
699             return -1
700 
701     def _refresh_configuration(self):
702         '''
703         This function gets the needed information from the configfile!
704         '''
705         if(1 == os.path.exists(self._configfile)):
706             config = LoadConfig(self._configfile)
707             error = config.get("misc.ignore_error")
```

```
708             error = error.strip()
709             error = error.strip(",")
710             ignore_error = error.split(",")
711             for i in range(len(ignore_error)):
712                 # check if there is an entry at all
713                 if ignore_error[i] != '':
714                     try:
715                         ignore_error[i] = int(ignore_error[i].strip())
716                     except ValueError:
717                         if self._verbose == 1:
718                             print "%s -> \"ignore_error\" in the config file has NO
                                    valid values, use old configuration" % time.ctime()
719                             return -1
720                 else:
721                     del ignore_error[i]
722 
723             self._parser.update_ignore_error(ignore_error)
724 
725             return 0
726         else:
727             # if file NOT exists terminate program
728             if self._verbose == 1:
729                 print "%s -> Sorry, the given config file does NOT exist !\nUsing old
                        configuration!\n\n" % time.ctime()
730             return -1
731 
732     def stop_thread(self):
733         """
734         Stop the thread
735         """
736         self._stop = 1
737 
738     def parse_directory(self, arg, dirname, fnames):
739         '''
740         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)!
741         '''
742         d = os.getcwd()
743         # change into log file directory
744         try:
745             os.chdir(dirname)
746         except:
747             if self._verbose == 1:
748                 print "%s -> could not find directory \"%s\"" % (time.ctime(),
                        dirname)
749             return -1
750         # for each file
751         for f in fnames:
752             # check if file and if file is a log file e.g. srbLog.20051003.gz
```

```
753             if (not os.path.isfile(f)) or (None == re.search('^srbLog[_0-9.-]*.gz', f
                    )):
754                 continue
755             # get last modified time
756             date = os.stat(f)[stat.ST_MTIME]
757             # create tupel
758             tupel = (date, f)
759             # save last modified time and filename into am arrray (list)
760             self._list.append(tupel)
761         # change back into the working directory
762         os.chdir(d)
763         # sort list ascending (aufsteigend)
764         self._list.sort()
765         # reverse list order, sorted descending (absteigend), the greater the time
                number the younger the file
766         self._list.reverse()
767         return 0
768
769     def gunzip(self, filus, name_temp_file="temp_srbLog"):
770         '''
771         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!
772         '''
773         if (not os.path.isfile(filus)):
774             return -1
775         else:
776             command = "gunzip -c %s > %s" % (filus, name_temp_file)
777             try:
778                 os.system(command)
779                 return 0
780             except:
781                 return -1
782
783 ############################  C L A S S   M U T E X    ########################
784
785 class Mutex:
786     '''
787     This class makes sure that server and client are not accessing the same file at
            the same time.
788     '''
789     # lock
790     _locked = threading.Lock()
791
792
793     def __init__(self):
794         '''
795         Constructor
796         '''
797         self.parsing = 0
798         self._parsing_thread_id = 0
799         self.client = 0
```

```
800
801
802     def set_variable_parsing(self, value, the_thread):
803         '''
804         set variable parsing
805         '''
806         Mutex._locked.acquire()  # lock
807         self._parsing_thread_id = the_thread
808
809         if self.client == 0:
810             #set variable
811             self.parsing = value
812             Mutex._locked.release()
813             time.sleep(1)
814             if value == 0:
815                 # reset parsing thread identity
816                 self._parsing_thread_id = 0
817             return 0
818         else:
819             Mutex._locked.release()   # release lock
820             time.sleep(1)
821             return -1
822
823     def set_variable_client(self, value):
824         '''
825         set variable client
826         '''
827         Mutex._locked.acquire()  # lock
828         # if client is not fetching the file
829         if self.parsing == 0:
830             #set variable
831             self.client = value
832             #print "client variable gesetzt"
833             Mutex._locked.release()   # release lock
834             time.sleep(1)
835             return 0
836         else:
837             if (0 != self._parsing_thread_id):
838                 if (1 != self._parsing_thread_id.isAlive()):
839                     # if parsing thread dead, reset semphore
840                     self.parsing = 0
841                     self._parsing_thread_id = 0
842             Mutex._locked.release()   # release lock
843             time.sleep(1)
844             return -1
845
846 ########################   C L A S S     R P C     ########################
847
848 class RPC:
849     '''
850     This class contains the RPC functions.
851     '''
```

```
852     def __init__(self, verbose, semaphore, config_file, interval, keyword_path,
            keyword_name, xml_file_path):
853         '''
854         constructor
855         '''
856         self._verbose = verbose
857         self._client = True
858         self._share = semaphore
859         self._config_file = config_file
860         self._interval = interval
861         self._keyword_path = keyword_path
862         self._keyword_name = keyword_name
863         self._xml_file_path = xml_file_path
864 
865         self._list = [] #for walking through the directory
866 
867     def rpc_stop_server(self):
868         '''
869         This function stops the server!
870         '''
871         command = "./stop_server"
872 
873         answer = os.system(command)
874         print answer
875         return answer
876 
877     def rpc_disable_rpc_calls(self):
878         '''
879         This function disables rpc.
880         '''
881         self._client = False
882         if self._verbose == 1:
883             print "%s -> RPC through \"admin tool\" disabled!" % time.ctime()
884         return "RPC disabled"
885 
886     def rpc_enable_rpc_calls(self):
887         '''
888         This function enables rpc.
889         '''
890         self._client = True
891         if self._verbose == 1:
892             print "%s -> RPC through \"admin tool\" enabled!" % time.ctime()
893         return "RPC enabled"
894 
895     def rpc_status(self):
896         '''
897         This function return the current status of the self._client variable.
898         '''
899         if self._client == True:
900             return "RPC enabled"
901         else:
902             return "RPC disabled"
```

```
903
904     def rpc_interval_status(self):
905         '''
906         This function returns the current parsing interval time.
907         '''
908         if self._client == True:
909             return self._interval
910         else:
911             return -2
912
913     def rpc_get_my_xml_file(self, filename):
914         '''
915         This functions gets the xml file from the server !
916         '''
917         if (self._client == True):
918             if (0 == self._share.set_variable_client(1)):
919                 # check if file is available
920                 try:
921                     filus = self._xml_file_path+"/"+filename
922                     client_xml_fd = open(filus, 'r')
923                     file_content = client_xml_fd.read()
924                     client_xml_fd.close()
925                 except IOError:
926                     self._share.set_variable_client(0)
927                     return "no file"
928                 #delete xml file
929                 if (0 == delete_file((self._xml_file_path+"/"+filename), self.
                    _verbose)):
930                     self._share.set_variable_client(0) # reset variable
931                     return file_content
932                 else:
933                     if self._verbose == 1:
934                         print "problems deleting file"
935                     self._share.set_variable_client(0)
936                     return -1
937             else:
938                 return -3   # server is busy parsing
939         else:
940             return -2 # rpc disalbed
941
942     def rpc_check_availabitity(self):
943         '''
944         This function check if the server is still in the parsing process.
945         '''
946         if (self._client == True):
947             if (0 == self._share.set_variable_client(1)):
948                 return 0
949             else:
950                 return -3
951         else:
952             return -2
953
```

```
954     def rpc_get_file_list(self):
955         '''
956         This function walks through the *.xml directory and finds all files, which
                need to be fetched form the client.
957         '''
958         if (self._client == True):
959             self._list = [] # empty list
960             try:
961                 os.path.walk(self._xml_file_path, self._parse_directory, self._list)
962                 if (0 < len(self._list)):
963                     return self._list
964                 else:
965                     return 0
966             except:
967                 return -1
968         else:
969             return -2 # rpc disalbed
970 
971     def rpc_update_configuration(self, section, key, value, action):
972         '''
973         This functions adds or deletes values in the config.ini.
974         action:
975         0 = delete
976         1 = add
977         2 = exchange
978         3 = info
979         '''
980         if(self._client == True):
981             try:
982                 config_fd = file(self._config_file, 'r+')
983             except IOError, e:
984                 return "Remote Control Attempt => Problem -> %s" % e
985 
986             byte_count = 0
987 
988             while(1):
989                 byte_count = config_fd.tell()
990                 line = config_fd.readline()
991                 if line == '':
992                     if self._verbose == 1:
993                         print "%s -> Remote Control Attempt => Section: \"%s\" and
                                key: \"%s\" do not exist in config file!" % (time.ctime()
                                , section, key)
994                     config_fd.close()
995                     return "Section: \"%s\" and key: \"%s\" do not exist in config
                            file!" % (section, key)
996                 if (-1 != line.find(section)):
997                     while(1):
998                         byte_count = config_fd.tell()
999                         line = config_fd.readline()
1000                        if line == '':
1001                            if self._verbose == 1:
```

```
1002                            print "%s -> Remote Control Attempt => Key: \"%s\" do
                                     not exist under section \"%s\" in config file!"
                                    % (time.ctime(), key, section)
1003                        config_fd.close()
1004                        return "Key \"%s\" do not exist under section \"%s\" in
                                config file!" % (key, section)
1005                    if (-1 != line.find(key) and -1 != line.find("=") and -1 ==
                            line.find("#", 0, 1)):
1006                        # if key word AND = AND no #
1007                        if action == 0:
1008                            # delete
1009                            if self._verbose == 1:
1010                                print "%s ->  Remote Control => Delete \"%s:%s\"
                                        value \"%s\"" % (time.ctime(), section, key,
                                        value)
1011                            listus = line.split("=")
1012                            listus[1] = listus[1].strip()
1013                            listus[1] = listus[1].strip(",")
1014                            listus = listus[1].split(",")
1015                            for i in range(len(listus)):
1016                                listus[i] = listus[i].strip()
1017                            new_content = ''
1018                            for i in range(len(listus)):
1019                                if int(listus[i]) != value:
1020                                    new_content = new_content+"%s, " % listus[i]
1021
1022                            new_content = new_content.strip()
1023                            new_content = new_content.strip(",")
1024                            new_content = "%s = %s\n" % (key, new_content)
1025                            rest = config_fd.read()
1026                            # truncate file content
1027                            config_fd.truncate(byte_count)
1028                            config_fd.seek(byte_count)
1029                            # write new line content
1030                            config_fd.writelines(new_content)
1031                            # write rest of file
1032                            config_fd.write(rest)
1033                            config_fd.close()
1034                            return "Changes applied: %s" % new_content
1035                        elif action == 1:
1036                            # add
1037                            if self._verbose == 1:
1038                                print "%s -> Remote Control => Add \"%s:%s\"
                                        value \"%s\"" % (time.ctime(), section, key,
                                        value)
1039                            listus = line.split("=")
1040                            listus[1] = listus[1].strip()
1041                            listus[1] = listus[1].strip(",")
1042                            listus = listus[1].split(",")
1043                            finish = 0
1044                            while (finish == 0):
1045                                if len(listus) == 0:
```

```
1046                                finish = 1
1047                            for i in range(len(listus)):
1048                                finish = 1 # break the while loop
1049                                listus[i] = listus[i].strip()
1050
1051                                try:
1052                                    #test if int
1053                                    temp_value = int(listus[i])
1054                                    if value == temp_value:
1055                                        config_fd.close()
1056                                        return "Value %d already exists!" %
                                                value
1057                                    except ValueError, e:
1058                                        finish = 0 # activate while loop
1059                                        #remove false/ invalid item
1060                                        del listus[i]
1061                                        break
1062
1063                        new_content = ''
1064                        for i in range(len(listus)):
1065                            new_content = new_content+"%s, " % listus[i]
1066                        new_content = "%s = %s%s\n" % (key, new_content,
                                value)
1067                        rest = config_fd.read()
1068                        config_fd.truncate(byte_count)
1069                        config_fd.seek(byte_count)
1070                        config_fd.writelines(new_content)
1071                        config_fd.write(rest)
1072                        config_fd.close()
1073                        return "Changes applied: %s" % new_content
1074                    elif action == 2:
1075                        # exchange
1076                        if self._verbose == 1:
1077                            print "%s -> Remote Control => Change \"%s:%s\"
                                    to value \"%s\"" % (time.ctime(), section,
                                    key, value)
1078                        rest = config_fd.read()
1079                        config_fd.truncate(byte_count)
1080                        config_fd.seek(byte_count)
1081                        new_content = "%s = %s\n" % (key, value)
1082                        config_fd.writelines(new_content)
1083                        config_fd.write(rest)
1084                        config_fd.close()
1085                        if section == 'misc' and key == 'minute':
1086                            self._interval = int(value)
1087                        return 0
1088                    elif action == 3:
1089                        # info
1090                        config_fd.close()
1091                        return line
1092                    else:
1093                        config_fd.close()
```

```
1094                                return -1
1095            else:
1096                return "RPC disabled" # rpc disalbed
1097
1098        def rpc_update_keyword_file(self, keyword, action):
1099            '''
1100            This function updates the keyword file.
1101
1102            action:
1103            0 = delete
1104            1 = add
1105            2 = info
1106            '''
1107            if(self._client == True):
1108                byte_count = 0
1109                file_size = 0
1110                comments = ''
1111
1112                filus = self._keyword_path+"/"+self._keyword_name
1113
1114                try:
1115                    key_fd = file(filus, 'r+')
1116                except IOError, e:
1117                    if self._verbose == 1:
1118                        print "%s -> Remote Control Attempt => Problem open keyword file
                                -> %s !" % (time.ctime(), e)
1119                    return "Problem -> %s" % e
1120
1121                key_fd.seek(0, 2) # set cursor to end of file
1122                file_size = key_fd.tell()
1123                key_fd.seek(0) # set cursor to begining of file
1124
1125                while(1):
1126                    # get comments
1127                    byte_count = key_fd.tell()
1128                    line = key_fd.readline()
1129                    if byte_count >= file_size:
1130                        break
1131                    if (-1 != line.find("#", 0, 1)):
1132                        comments += line
1133
1134                key_fd.close()
1135                keyword_list = get_keywords(filus)
1136                keyword = keyword.split(":")
1137
1138                if action == 0:
1139                    # test if keyword is already there
1140                    for i in range(len(keyword_list)):
1141                        if 1 == len(keyword) and 1 == len(keyword_list[i]):
1142                            if keyword[0] == keyword_list[i][0]:
1143                                del keyword_list[i]
1144                                try:
```

```
1145                                key_fd = file(filus, 'w+')
1146                                key_string = list_to_string(keyword_list)
1147                                key_fd.write(comments)
1148                                key_fd.write(key_string)
1149                                key_fd.close()
1150                            except IOError, e:
1151                                if self._verbose == 1:
1152                                    print "%s -> Remote Control Attempt => Problem
                                        open keyword file -> %s !" % (time.ctime(), e
                                        )
1153                                return "Problem -> %s" % e
1154                            return "keyword %s from keyword list deleted" % keyword
1155                    elif 2 == len(keyword) and 2 == len(keyword_list[i]):
1156                        if keyword[0] == keyword_list[i][0] and keyword[1] ==
                            keyword_list[i][1]:
1157                            del keyword_list[i]
1158                            try:
1159                                key_fd = file(filus, 'w+')
1160                                key_string = list_to_string(keyword_list)
1161                                key_fd.write(comments)
1162                                key_fd.write(key_string)
1163                                key_fd.close()
1164                            except IOError, e:
1165                                if self._verbose == 1:
1166                                    print "%s -> Remote Control Attempt => Problem
                                        open keyword file -> %s !" % (time.ctime(), e
                                        )
1167                                return "Problem -> %s" % e
1168                            return "keyword %s from keyword list deleted" % keyword
1169
1170                return "keyword %s was not part of keyword list" % keyword
1171
1172            if action == 1:
1173                # test if keyword is already there
1174                for i in range(len(keyword_list)):
1175                    if 1 == len(keyword) and 1 == len(keyword_list[i]):
1176                        if keyword[0] == keyword_list[i][0]:
1177                            return "keyword %s already in keyword list" % keyword
1178                    elif 2 == len(keyword) and 2 == len(keyword_list[i]):
1179                        if keyword[0] == keyword_list[i][0] and keyword[1] ==
                            keyword_list[i][1]:
1180                            return "keywords %s already in keyword list" % keyword
1181
1182                keyword_list.append(keyword)
1183                try:
1184                    # open file for writing
1185                    key_fd = file(filus, 'w+')
1186                except IOError, e:
1187                    if self._verbose == 1:
1188                        print "%s -> Remote Control Attempt => Problem open keyword
                            file -> %s !" % (time.ctime(), e)
1189                    return "Problem -> %s" % e
```

```
1190
1191                key_string = list_to_string(keyword_list)
1192                key_fd.write(comments)
1193                #write keyword list in new line
1194                key_string = "\n"+key_string
1195                # write keywords in file
1196                key_fd.write(key_string)
1197                key_fd.close()
1198
1199                # do not return first "\n"
1200                return key_string[1:]
1201
1202            if action == 2:
1203                return list_to_string(keyword_list)
1204
1205        else:
1206            return "RPC disabled"
1207
1208    def _parse_directory(self, arg, dirname, fnames):
1209        '''
1210        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)!
1211
1212        dirname = directory which need to be pared
1213        fnames = files within dirname
1214        '''
1215        d = os.getcwd()
1216        # change into log file directory
1217        try:
1218            os.chdir(dirname)
1219        except:
1220            if self._verbose == 1:
1221                print "could not find directory \"%s\"" % dirname
1222            return -1
1223        # for each file
1224        for f in fnames:
1225            # check if file and if file is a log file e.g. client_log.xml
1226            if (not os.path.isfile(f)) or (None == re.search('client_log.xml', f)):
1227                continue
1228            else:
1229            # save filename into an arrray (list)
1230                self._list.append(f)
1231        # change back into the working directory
1232        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:
```

```
50              return -1
51
52      return 0
53
54  def get_keywords(filus):
55      '''
56      This function extracts keywords from a give file!
57      '''
58      keys = []
59
60      try:
61          file_fd = file(filus, 'r')
62      except IOError, e:
63          print "Problem with keyword file -> ", e
64          return -1
65
66      content = file_fd.readlines()# save file content as list (1 line == 1 entry)
67
68      file_fd.close()
69
70      content = remove_item(content, "#")
71      content = remove_item(content, "\n")
72
73      for i in range(len(content)):
74          content[i] = content[i].strip()
75          content[i] = content[i].rstrip(",")
76          content[i] = content[i].split(",")
77          for a in range(len(content[i])):
78              keys.append(content[i][a])
79
80      for i in range(len(keys)):
81          keys[i] = keys[i].strip() # remove whitespace
82          keys[i] = keys[i].split(":")
83
84      return keys
85
86  def remove_item(listus, item):
87      '''
88      This function removes an item for a list (2 dimentional) as a rekursive function.
89      '''
90
91      while(1):
92
93          for i in range(len(listus)):
94              if -1 != listus[i].find(item, 0, 1):
95                  del listus[i]
96                  remove_item(listus, item)
97                  break
98          else:
99              break
100
101     return listus
```

```
102
103 def list_to_string(listus):
104     '''
105     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.
106     '''
107     str_listus = ''
108
109     for i in range(len(listus)):
110         if 1 == len(listus[i]):
111             str_listus += listus[i][0]+", "
112         elif 2 == len(listus[i]):
113             str_listus += listus[i][0]+":"+listus[i][1]+", "
114
115     str_listus = str_listus.strip()
116     str_listus = str_listus.strip(",")
117
118     return str_listus
119
120 def delete_file(file_name, verbose):
121     '''
122     This function deletes a file.
123     '''
124     try:
125         os.remove(file_name)
126         return 0
127     except:
128         if verbose == 1:
129             print "%s -> could not delete -> \"%s\"" % (time.ctime(), file_name)
130         return -1
131
132 def usage_exit(progname, msg=None):
133     '''
134     This function diplays the usage of this program and terminates the program!
135     '''
136     if msg:
137         print msg
138         print
139     print "usage: python %s [ -h|--help -c|--config -v|--verbose -d|--daemon] \n\n" %
            progname
140     os._exit(-1)
141
142 def find(search, listus):
143     '''
144     This function finds an item within a list (1 dimensional).
145     '''
146     for i in range(len(listus)):
147         if listus[i] == search:
148             return listus[i]
149     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
```

```
8  '''
9
10 import client_classes, os, sys, time
11 import  getopt, smtplib, socket
12 from utils_client import LoadConfig, check_ip, usage_exit, get_password
13
14 class MyClient:
15     '''
16     main class for the client application
17     '''
18     def __init__(self, config_data, verb, smtp_pa):
19         '''
20         Constructor
21         '''
22         self._verbose = verb
23         config = config_data
24         self._workingpath = os.getcwd()
25
26         #------------------ put together path and file
               ------------------------------------
27         self._database_name = config.get("database.name")
28         self._database_path = config.get("database.path")
29         self._database_path = self._database_path.rstrip("/")
30         if(config.get("database.path") == '' or config.get("database.path") == None):
31             # field is empty
32             self._database_path = self._workingpath
33
34         else:
35             self._database_name = self._database_name.strip()
36             if (-1 != self._database_path.find("/", 0, 1)):
37                 # first character "/"
38                 pass
39             else:
40                 self._database_path = self._workingpath+"/"+self._database_path
41
42         self._error_description_name = config.get("files.error_description")
43         self._error_description_path = config.get("path.path_error_description")
44         self._error_description_path = self._error_description_path.rstrip("/")
45         if(config.get("path.path_error_description") == '' or config.get("path.
               path_error_description") == None):
46             self._error_description_path = self._workingpath
47         else:
48             self._error_description_name = self._error_description_name.strip()
49             if (-1 != self._error_description_path.find("/", 0, 1)):
50                 # first character "/"
51                 pass
52             else:
53                 self._error_description_path = self._workingpath+"/"+self.
                       _error_description_path
54
55         self._client_certificate = config.get("files.client_certificate")
56         self._client_certificate_path = config.get("path.path_client_certificate")
```

```
57          self._client_certificate_path = self._client_certificate_path.rstrip("/")
58          if(config.get("path.path_client_certificate") == '' or config.get("path.
                path_client_certificate") == None):
59              self._client_certificate_path = self._workingpath
60          else:
61              self._client_certificate = self._client_certificate.strip()
62              if (-1 != self._client_certificate_path.find("/", 0, 1)):
63                  # first character "/"
64                  pass
65              else:
66                  self._client_certificate_path = self._workingpath+"/"+self.
                        _client_certificate_path
67
68          self._client_ca = config.get("files.client_ca")
69          self._client_ca_path = config.get("path.path_client_ca")
70          self._client_ca_path = self._client_ca_path.rstrip("/")
71          if(config.get("path.path_client_ca") == '' or config.get("path.path_client_ca
                ") == None):
72              self._client_ca_path = self._workingpath
73          else:
74              self._client_ca = self._client_ca.strip()
75              if (-1 != self._client_ca_path.find("/", 0, 1)):
76                  # first character "/"
77                  pass
78              else:
79                  self._client_ca_path = self._workingpath+"/"+self._client_ca_path
80
81          # check if the configuration if correct
82          if(0 == os.path.exists(self._client_certificate_path+"/"+self.
                _client_certificate)):
83              print "Could not locate client certifiate under %s !\nMaybe change
                    configuration file and try again!\n\n" % self.
                    _client_certificate_path
84              os._exit(-1)
85
86          if(0 == os.path.exists(self._client_ca_path+"/"+self._client_ca)):
87              print "Could not locate client ca certificate under %s !\nMaybe change
                    configuration file and try again!\n\n" % self._client_ca_path
88              os._exit(-1)
89
90          if(0 == os.path.exists(self._error_description_path+"/"+self.
                _error_description_name)):
91              print "Could not locate error description file under %s !\nMaybe change
                    configuration file and try again!\n\n" % self._error_description_path
92              os._exit(-1)
93
94          self._project = config.get("project.name")
95          self._interval = int(config.get("misc.minute"))
96
97          #------------------ create server list -----------------------
98          servers = config.get("server.serverlist")
99          # split where commas
```

```
100         servers_split = servers.split(",")
101         # create dictionary
102         self._serverlist = {}    # dictionary for serverlist:port
103         for i in range(len(servers_split)):
104             # remove whitespace
105             servers_split[i] = servers_split[i].strip()
106             temp_list = servers_split[i].split(":")
107             # remove whitespace
108             if len(temp_list) != 2:
109                 print "The IP configuration \"%s\" seems not correct. \nPlease check
                        the configuration file!\n\n" % temp_list[0]
110                 os._exit(-1)
111             temp_list[0] = temp_list[0].strip()
112             temp_list[1] = temp_list[1].strip()
113             # check if IP is valid
114             if( -1 == check_ip(temp_list[0])):
115                 print "The IP \"%s\" seems not correct. \nPlease check the
                        configuration file!\n\n" % temp_list[0]
116                 os._exit(-1)
117             try:
118                 temp_list[1] = int(temp_list[1])
119             except ValueError:
120                 print "The port \"%s\" is not valid.\nPlease check the configuration
                        file!\n\n" % temp_list[1]
121                 os._exit(-1)
122             if (temp_list[1] < 1024 or temp_list[1] > 50001):
123                 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"
124                 os._exit(-1)
125             self._serverlist[temp_list[0]] = temp_list[1]
126
127         self._share = client_classes.Mutex()
128
129         # mail issues
130         self._smtp_server = config.get("mail.smtp_server")
131         self._smtp_pass = smtp_pa
132         self._smtp_from = config.get("mail.from")
133         self._smtp_user = config.get("mail.user")
134
135         if (None != self._smtp_pass):
136             # test if smtp server and login is possible
137             try:
138                 if self._verbose == 1:
139                     print "Test SMTP connection to server \"%s\"...." % self.
                            _smtp_server
140                 server = smtplib.SMTP(self._smtp_server)
141                 if self._verbose == 1:  # --- debug ---
142                     server.set_debuglevel(1) # ----- debug -----
143                 server.login(self._smtp_user, self._smtp_pass)
144                 server.quit()
145                 if self._verbose == 1:
```

```
146                 print "SMTP connection successfully tested"
147             except smtplib.SMTPAuthenticationError, e:
148                 print "Problem with SMTP server authentication -> \"%s\" !" % e
149                 print "\n"
150                 os._exit(-1)
151             except socket.error, e:
152                 print "Problem with SMTP server -> \"%s\" !" % e
153                 print "\n"
154                 os._exit(-1)
155
156         self._mail_address = []     # mail address list
157         z = 1
158         while(1):
159             temp = "mail_to.address_%d" % z
160             testus = config.get(temp)
161             if testus == None:
162                 break
163             self._mail_address.append(testus)
164             z += 1
165
166         keywordfiles = []
167
168         z = 1
169         while(1):
170             temp = "mail_to.path_ignore_error_%d" % z
171             path = config.get(temp)
172             if(path == '' or path == None):
173                 path = self._workingpath
174             else:
175                 path = path.rstrip("/")
176                 if (-1 != path.find("/", 0, 1)):
177                     # first character "/"
178                     pass
179                 else:
180                     path = self._workingpath+"/"+path
181
182             temp = "mail_to.file_ignore_error_%d" % z
183             filus = config.get(temp)
184             if filus == None:
185                 break
186             filus = filus.strip()
187
188             keywordfilus = path+"/"+filus
189
190             if(0 == os.access(keywordfilus, 4)):     # 4 -> R_OK -> read only
191                 print "Could not access keyword file under %s !\nMaybe change
                        configuration file and try again!\n\n" % keywordfilus
192                 os._exit(-1)
193
194             keywordfiles.append(keywordfilus)
195             z += 1
196
```

```
197         self._mail_ignore_error = range(len(keywordfiles))
198         for i in range(len(keywordfiles)):
199             self._mail_ignore_error[i] = self._get_keywords(keywordfiles[i])
200 
201     def _get_keywords(self, filus):
202         '''
203         This function extracts keyword from a give file!
204         '''
205         keys = []
206 
207         try:
208             file_fd = file(filus, 'r')
209         except IOError, e:
210             print "Problem with keyword file -> ",  e
211             return -1
212 
213         content = file_fd.readlines()# save file contetn as list (1 line == 1 entry)
214 
215         file_fd.close()
216 
217         content = self._remove_item(content, "#")
218         content = self._remove_item(content, "\n")
219 
220         for i in range(len(content)):
221             content[i] = content[i].strip()
222             content[i] = content[i].rstrip(",")
223             content[i] = content[i].split(",")
224             for a in range(len(content[i])):
225                 keys.append(content[i][a])
226 
227         for i in range(len(keys)):
228             keys[i] = keys[i].strip() # remove whitespace
229             keys[i] = keys[i].split(":")
230 
231         return keys
232 
233     def _remove_item(self, listus, item):
234         '''
235         This function removes an item for a list as a rekursive function.
236         '''
237         while(1):
238 
239             for i in range(len(listus)):
240                 if -1 != listus[i].find(item, 0, 1):
241                     del listus[i]
242                     self._remove_item(listus, item)
243                     break
244             else:
245                 break
246 
247         return listus
248 
```

```
249     def initialise_database(self):
250         '''
251         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.
252         '''
253         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)
254
255     def get_serverlist(self):
256         '''
257         This function returns the server list.
258         '''
259         return self._serverlist
260
261     def fetch_error_messages(self):
262         '''
263         This function starts the worker thread, who initalises the regular fetching
                of the error messages.
264         '''
265         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)
266         self._workerthread.setName("workerthreadDaemon")
267         self._workerthread.start()
268
269         if self._verbose == 1:
270             print "%s -> Manager thread started !" % ( time.ctime() ) #--- debug ----
271
272
273 ##############################################################
274
275 def daemonize(verbose, stdout = '/dev/null', stderr = None, stdin = '/dev/null',
      pidfile = None, startmsg = 'Client daemon started with pid %s'):
276
277     '''
278     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.
279
280     source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/66012
281     modified by a.weise November 2005
282     '''
283
284     # first fork => fork creates first child-process
285     try:
286         pid = os.fork()
```

```
287         if (pid > 0):
288             sys.exit(0) # close first parent-process
289 
290     except OSError, e:
291         sys.stderr.write("fork #1 failed: (%d) %s\n" % (e.errno, e.strerror))
292         sys.exit(1)
293 
294     os.umask(0)
295     os.setsid()
296 
297     # second fork
298     try:
299         pid = os.fork()
300         if (pid > 0):
301             sys.exit(0) # close second parent-process
302     except OSError, e:
303 
304         sys.stderr.write("fork #2 failed: (%d) %s\n" % (e.errno, e.strerror))
305         sys.exit(1)
306 
307     # open standard in and out and print standard message
308     if (not stderr):# if not stderr given => take stdout-path
309         stderr = stdout
310 
311     if verbose == 1:
312         si = file(stdin, 'r')
313         so = file(stdout, 'w+') # w -> overwrite old log content
314         sys.stderr.write("%s" % so)
315         se = file(stderr, 'w+', 0)
316         pid = str(os.getpid())
317         sys.stderr.write("\n%s\n" % startmsg % pid)
318         sys.stderr.write("\nwarum\n")
319         sys.stderr.flush()
320         if pidfile:
321             file(pidfile,'w+').write("%s\n" % pid)
322 
323         # redirect standard in and out to files
324         os.dup2(si.fileno(), sys.stdin.fileno())
325         os.dup2(so.fileno(), sys.stdout.fileno())
326         os.dup2(se.fileno(), sys.stderr.fileno())
327 
328 ######################################################################
329 
330 def start():
331 
332     '''
333     Start the application.
334     '''
335     configfile = ""
336     verbose = 0
337     smtp_pass = None
338     daemon = 0
```

```
339
340     try:
341         opts, args = getopt.getopt(sys.argv[1:], 'c:vhpd', ['config=', 'verbose', '
                help', 'smtp_password', '--daemon'])
342         for opt, value in opts:
343             if opt in ('-h','--help'):
344                 msg = "\n-----------   Help   ----------\n\n\n"\
345                       "-c or --config\t\t->  defines config file, if no config file
                            given, default values are used\n"\
346                       "-p or --smtp_password\t->  activates mail notification sending
                            \n"\
347                       "-v or --verbose\t\t->  activates printing of messages [debug
                            option]\n"\
348                       "-d or --daemon\t\t->  daemonize the client\n"\
349                       "-h or --help\t\t->  print this help\n\n"
350                 usage_exit(sys.argv[0], msg)
351             if opt in ('-c','--config'):
352                 value = value.replace("=", "")
353                 configfile = os.getcwd()+"/"+value
354             if opt in ('-v','--verbose'):
355                 verbose = 1
356             if opt in ('-p', '--smtp_password'):
357                 smtp_pass = get_password("Please enter SMTP password: ")
358             if opt in ('-d', '--daemon'):
359                 daemon = 1
360     except getopt.error, e:
361         usage_exit(sys.argv[0], e)
362
363     # load config file or default values
364     if (configfile != ""):
365         # check if file exists
366         if(1 == os.path.exists(configfile)):
367             config = LoadConfig(configfile)
368         else:
369             # if file NOT exists terminate program
370             print "Sorry, a given file does NOT exist !\nPlease try again!\n\n"
371             os._exit(-1)
372     else:
373         msg = "\nNo config file spezified !\n"
374         usage_exit(sys.argv[0], msg)
375
376     print "\n\n---------------------- SRB LOG FILE PARSER [ CLIENT ]
            ------------------ \n\n"
377     print "Starting ..."
378
379
380     worker = MyClient(config, verbose, smtp_pass)
381     worker.initialise_database()
382
383     if daemon == 1:
384         if verbose == 1:
385             daemonize(verbose, stdout = 'daemonise.log')
```

```
386         else:
387             daemonize(verbose)
388     else:
389         pass
390 
391     print "%s -> Start manager thread ..." % (time.ctime())
392     worker.fetch_error_messages()
393 
394 if __name__ == '__main__':
395 
396     start()
```

D.2.2 Module `client_classes.py`

LISTING D.6: Module `client_classes.py`

```
1  #!/usr/bin/env python
2  '''
3  This module contains all imports, defines and basic classes for start_client.py.
4  
5  Reading University
6  MSc in Network Centered Computing
7  a.weise - a.weise@reading.ac.uk - December 2005
8  '''
9  # misc
10 import os, sys, signal, re, copy
11 import string, time
12 
13 # database
14 import sqlite
15 
16 #mail
17 import smtplib, socket
18 
19 # xml parsing
20 from xml.sax import make_parser
21 from xml.sax.handler import ContentHandler, feature_namespaces
22 import xml.sax
23 
24 # connection issues
25 from M2Crypto.m2xmlrpclib import Server, SSL_Transport
26 from M2Crypto import SSL
27 
28 # threads
29 import threading, thread
30 
31 ################## CLASS  MyContentHandler  ##################
32 
33 class MyContentHandler(ContentHandler):
```

```
34      '''
35      This class is derived from _xmlplus.sax.handler and provides individual functions
             for parsing the xml file.
36      '''
37
38      def __init__ (self, db_object, ip, ignore_error, mail_obj, verbose):
39          '''
40          Constructor
41          '''
42          self._verbose = verbose
43          self._my_mail_ignore_error = ignore_error
44          self._mail_obj = mail_obj
45          self._ip = ip
46          self._db = db_object
47          self._db_access = self._db.get_access_cursor()
48          self._searchTerm = ""
49          self._date = ""
50          self._date_flag = 0
51          self._time = ""
52          self._time_flag = 0
53          self._error_number = 0
54          self._error_number_flag = 0
55          self._error_string = ""
56          self._error_string_flag = 0
57          self._linenumber = 0
58          self._linenumber_flag = 0
59
60      def set_ip(self, ip):
61          '''
62          The function sets the member variable _ip.
63          '''
64          self._ip = ip
65
66      def startElement(self, tag, attr):
67          '''
68          The function overwrites the startElement function.
69          '''
70          self._searchTerm = tag
71
72      def characters (self, tag_text):
73          '''
74          This function overwrites the character function to extract the tag content.
75          '''
76          if (self._searchTerm == "date"):
77              self._date = tag_text
78              self._date_flag = 1
79          elif (self._searchTerm == "time"):
80              self._time = tag_text
81              self._time_flag = 1
82          elif (self._searchTerm == "error_number"):
83              self._error_number = tag_text
84              self._error_number_flag = 1
```

```
85          elif (self._searchTerm == "error_string"):
86              self._error_string = tag_text
87              self._error_string_flag = 1
88          elif (self._searchTerm == "linenumber"):
89              self._linenumber = tag_text
90              self._linenumber_flag = 1
91
92      def endElement(self, tag):
93          '''
94          This function overwrites endElement function.
95          '''
96          if (self._searchTerm == "date"):
97              pass
98          elif (self._searchTerm == "time"):
99              pass
100         elif (self._searchTerm == "error_number"):
101             pass
102         elif (self._searchTerm == "error_string"):
103             self._error_string = self._error_string.replace("\"", "")
104         elif (self._searchTerm == "linenumber"):
105             pass
106         self._searchTerm = "" #reset variable
107
108         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):
109             # save in database
110             success = self._insert()
111             if success == -1:
112                 # raise exception to exit
113                 print " raise exception"
114                 assert success == 0
115
116             # add mail content
117             if (0 != len(self._mail_obj)):
118                 for i in range(len(self._mail_obj)):
119                     check = self._test_keywords(self._my_mail_ignore_error[i], len(
                            self._my_mail_ignore_error[i])-1, self._error_string)
120                     if (0 == check):
121                         # print " add mail content"
122                         cont = "\n-------------------------------------"\
123                                "\ndate:\t\t\t"+self._date+ \
124                                "\ntime:\t\t\t"+self._time+ \
125                                "\nerror message:\t\t"+self._error_string+ \
126                                "\nline number:\t\t"+self._linenumber
127                         # add mail content
128                         self._mail_obj[i][0].add(cont)
129                         # modify error counter
130                         self._mail_obj[i][0].count()
131                         # set first date
132                         temp = "%s (%s)" % (self._date, self._time)
133                         if self._mail_obj[i][0].get_first_date() == '':
134                             self._mail_obj[i][0].set_first_date(temp)
```

```
135                         # set last date
136                         self._mail_obj[i][0].set_last_date(temp)
137
138             # reset variables
139             self._reset()
140
141     def _test_keywords(self, keywordlist, amount_of_keywords, teststring):
142         '''
143         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
144
145         keywordlist = list of all keywords
146         amount_of_keywords = number of keywords in list
147         teststring = string, which needs to be investigated
148
149         return -1 if line is not interesting
150         return 0 if line is taken
151         '''
152         if (amount_of_keywords == 0):
153             #last keyword check   -1 != content.rfind("NOTICE")
154             if( 2 == len(keywordlist[amount_of_keywords])):
155                 if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
156                     # not in string go to next keyword
157                     return 0
158                 else:
159                     if( -1 == keywordlist[amount_of_keywords][1].rfind("!")):
160                         # check for NO keyword
161                         temp = keywordlist[amount_of_keywords][1].strip("!")
162                         if ( -1 == teststring.rfind(temp)):
163                             # go on to next keyword
164                             return 0
165                         else:
166                             return -1
167                     else:
168                         # there is no "!"
169                         if ( -1 != teststring.rfind(keywordlist[amount_of_keywords
                                ][1])):
170                             # string is there, go on to next keyword
171                             return 0
172                         else:
173                             return -1
174             else:
175                 if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
176                     # not in string go to next keyword
177                     return 0
178                 else:
179                     return -1
180         else:
181             if( 2 == len(keywordlist[amount_of_keywords])):
182                 if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
183                     # not in string go to next keyword
```

```
184                     return self._test_keywords(keywordlist, amount_of_keywords-1,
                            teststring)
185                 else:
186                     if( -1 == keywordlist[amount_of_keywords][1].rfind("!")):
187                         # check for NO keyword
188                         temp = keywordlist[amount_of_keywords][1].strip("!")
189                         if ( -1 == teststring.rfind(temp)):
190                             # go on to next keyword
191                             return self._test_keywords(keywordlist,
                                    amount_of_keywords-1, teststring)
192                         else:
193                             return -1
194                     else:
195                         # there is no "!"
196                         if ( -1 != teststring.rfind(keywordlist[amount_of_keywords
                                ][1])):
197                             # string is there, go on to next keyword
198                             return self._test_keywords(keywordlist,
                                    amount_of_keywords-1, teststring)
199                         else:
200                             return -1
201             else:
202                 if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
203                     # not in string go to next keyword
204                     return self._test_keywords(keywordlist, amount_of_keywords-1,
                            teststring)
205                 else:
206                     return -1
207
208     def _reset(self):
209         '''
210         This function resets member variables.
211         '''
212         self._date = ""
213         self._date_flag = 0
214         self._time = ""
215         self._time_flag = 0
216         self._error_number = 0
217         self._error_number_flag = 0
218         self._error_string = ""
219         self._error_string_flag = 0
220         self._linenumber = 0
221         self._linenumber_flag = 0
222
223     def _insert(self):
224         '''
225         This function inserts the data from the xml file into the database.
226         '''
227         # get first error number id !!!!!!!!!!!!!!!!!!
228         if ('-' == self._error_number):
229             #if no error number
230             self._error_number = 999999
```

```
231
232         sql = ' SELECT * FROM error WHERE e_number = "%s" ' % self._error_number
233         success, self._db_access = self._db.execute_sql(1200, self._db_access, sql)
234         if success == -1:
235             return -1
236         data = self._db_access.fetchall()
237         if ( 0 == len(data)):
238             # error number not in database -> insert new error number into database
239             sql = ' INSERT INTO error (e_number, e_name, e_description) VALUES ("%s",
                    "not specified", "") ' % self._error_number
240             data = self._db_access.execute(sql)
241             sql = ' SELECT * FROM error WHERE e_number = "%s" ' % self._error_number
242             success, self._db.execute_sql(1200, self._db_access, sql)
243             if success == -1:
244                 return -1
245             data = self._db_access.fetchall()
246
247         error_id = data[0]["e_id"]
248
249         # check if dataset already there
250         sql = 'SELECT * FROM messages WHERE error_e_id = "%s"'% error_id+ \
251             ' AND m_date = "%s" '% self._date + \
252             ' AND m_time = "%s" '% self._time + \
253       ' AND m_error_string = "%s"' %( self._error_string)
254
255         success, self._db.execute_sql(1200, self._db_access, sql)
256         if success == -1:
257             return -1
258         data = self._db_access.fetchall()
259         if (0 == len(data)):
260             # if dataset is not in database insert it
261             # 2. get host id
262             sql = 'SELECT * FROM host WHERE h_ip_address = "%s";' % self._ip
263             success, self._db.execute_sql(1200, self._db_access, sql)
264             if success == -1:
265                 return -1
266             data = self._db_access.fetchall()
267             if (1 == len(data)):
268                 ip = data[0]["h_id"]
269             else:
270                 ip = data[0]["h_id"]
271             # insert data in database
272             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)
273             success, self._db.execute_sql(1200, self._db_access, sql)
274             if success == -1:
275                 return -1
276         else:
277             pass
278
```

```
279         return 0
280
281 ################## CLASS  Mail  ##################
282
283 class Mail:
284     '''
285     This class deals with the mail issues.
286     '''
287
288     def __init__(self, mail_address, smtp_server, smtp_pass, smtp_from, user, verbose
            ):
289         '''
290         Constructor
291         '''
292         self._verbose = verbose
293         self._mail_address = mail_address
294         self._smtp_server = smtp_server
295         self._smtp_pass = smtp_pass
296         self._smtp_from = smtp_from
297         self._smtp_user = user
298         self._mail_name = "temp_email_unknown.txt"
299
300         self._error_count = 0
301         self._first_date = ''
302         self._last_date = ''
303         self._first_date_flag = 0
304
305
306
307     def create_content(self, name):
308         '''
309         This function creates a temorary file, where the mail content gets saved
                temporarly.
310         '''
311         try:
312             file_fd = open(name, 'w')
313             self._mail_name = name
314             file_fd.close()
315             return 0
316         except IOError, e:
317             if self._verbose == 1:
318                 print "%s -> Problem creating email content -> " % (time.ctime(), e)
319             return -1
320
321     def add(self, content):
322         '''
323         This function adds to the mail content.
324         '''
325         try:
326             file_fd = file(self._mail_name, 'r+')
327             file_fd.seek(0, 2) # cursor to end of file
328             file_fd.writelines(content)
```

```
329             file_fd.close()
330             return 0
331         except IOError, e:
332             if self._verbose == 1:
333                 print "%s -> Problem adding email content -> " % (time.ctime(), e)
334             return -1
335 
336     def count(self):
337         '''
338         This function counts all inserted error within the mail by incrementing the
                member variable self._error_count.
339         '''
340         self._error_count += 1
341 
342     def set_first_date(self, value):
343         '''
344         This function modifies the memeber variable self._first_date.
345         '''
346         self._first_date = value
347 
348     def get_first_date(self):
349         '''
350         This function returns the content of the memeber variable self._first_date.
351         '''
352         return self._first_date
353 
354     def set_last_date(self, value):
355         '''
356         This function modifies the memeber variable self._last_date
357         '''
358         self._last_date = value
359 
360     def send_mail(self, receiver, server):
361         '''
362         This function sends the mail away.
363         '''
364         if self._verbose == 1:
365             print "%s -> Try to send Mail, to -> \"%s\" ..." % (time.ctime(),
                    receiver)
366 
367         # put together mail content
368         subject = 'SRB LOG FILE PARSER NOTIFICATION - %s' % time.ctime(time.time())
369         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)
370 
371         try:
372             if self._error_count <= 5000:
373                 file_fd = open(self._mail_name, 'r')
374                 mail_error = file_fd.read()
```

```
375                 file_fd.close()
376             else:
377                 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"
378
379             if mail_error != "":
380                 content += mail_error
381                 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]'
382
383                 timus = time.strftime("%d %B %Y %H:%M:%S")
384
385                 text = 'From: '+self._smtp_from+'\n'
386                 text += 'To: '+receiver+'\n'
387                 text += 'Date: '+timus+'\n'
388                 text += 'Subject: '+subject+'\n'
389
390                 text = text + content
391
392                 # establish connection to smtp server
393                 server = smtplib.SMTP(self._smtp_server)
394                 server.login(self._smtp_user, self._smtp_pass)
395
396                 #transmit
397                 server.sendmail(self._smtp_from, receiver, text)
398                 #done
399                 if self._verbose == 1:
400                     print "%s -> Mail sent to \"%s\" !" % (time.ctime(), receiver)
401                 server.quit()
402                 self._error_count = 0
403                 return 0
404             else:
405                 if self._verbose == 1:
406                     print "%s -> Nothing to send to \"%s\" !" % (time.ctime(),
                            receiver)
407                 self._error_count = 0
408                 return -1
409         except smtplib.SMTPAuthenticationError, e:
410             if self._verbose == 1:
411                 print "%s -> Proplem with SMTP server authentication -> \"%s\" !" % (
                        time.ctime(), e)
412                 print "\n"
413             self._error_count = 0
414             return -1
415         except socket.error, e:
416             if self._verbose == 1:
417                 print "%s -> Problem with SMTP server -> \"%s\" !" % (time.ctime(), e
                        )
418                 print "\n"
419             self._error_count = 0
```

```
420             return -1
421         except:
422             if self._verbose == 1:
423                 print "%s -> Problem with sending mail to \"%s\" !" % (time.ctime(),
                        receiver)
424             self._error_count = 0
425             return -1
426 
427     def delete_content(self):
428         '''
429         This function deletes the temorary file with the mail content.
430         '''
431         try:
432             os.remove(self._mail_name)
433             if self._verbose == 1:
434                 print "%s -> Deleted -> \"%s\"" % (time.ctime(),  self._mail_name)
435             return 0
436         except OSError, e:
437             if self._verbose == 1:
438                 print "%s -> Could not delete mail content file! -> \"%s\" -> %s" % (
                        time.ctime(), self._mail_name, e)
439             return -1
440 
441 ################## CLASS  MyDatabase  ##################
442 
443 class MyDatabase:
444     '''
445     This class deals with all the database issues.
446     '''
447 
448     def __init__(self, error_description_file, error_desciption_path, databasename,
            database_path, serverlist, project, verbose):
449         '''
450         constructor
451         '''
452         self._verbose = verbose
453         error = "%s/%s" % (error_desciption_path, error_description_file)
454         self._db_access = None
455         self._database_path = database_path
456         #check if path exists
457         if(1 == os.path.exists(database_path)):
458             if self._verbose == 1:
459                 print "%s -> Database exists" % (time.ctime())# ---- debug ----
460             os.chdir(database_path)
461         else:
462             #create wanted path
463             if self._verbose == 1:
464                 print "%s -> Create database " % time.ctime() # ---- debug ----
465             os.mkdir(database_path)
466             os.chdir(database_path)
467 
468         if(0 == os.path.exists(databasename)):
```

```
469             try:
470                 # 1. create database
471                 self._connect = sqlite.connect(databasename, autocommit = 1)
472
473                 # 2. create access cursor
474                 self._db_access = self._connect.cursor()
475
476                 # 3. create tables
477
478                 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);"
479                 self._db_access.execute(sql)
480
481                 sql = "CREATE TABLE host (h_id INTEGER NOT NULL PRIMARY KEY,
                        h_ip_address CHAR(15) NOT NULL, h_hostname CHAR(30) NULL);"
482                 self._db_access.execute(sql)
483
484                 sql = "CREATE TABLE host_project (hp_h_id INTEGER UNSIGNED NOT NULL,
                        hp_p_id INTEGER UNSIGNED NOT NULL);"
485                 self._db_access.execute(sql)
486
487                 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);"
488                 self._db_access.execute(sql)
489
490                 sql = "CREATE TABLE project (p_id INTEGER NOT NULL PRIMARY KEY,
                        p_name CHAR(100) NOT NULL);"
491                 self._db_access.execute(sql)
492
493                 # insert data if necessary
494                 # insert error codes
495                 error_file_fd = open(error, 'r')
496                 content = error_file_fd.readline()              # get first line
497                 x = 0
498                 if self._verbose == 1:
499                     print "%s -> Initialising database ...\n" % time.ctime()
500                 z = 0
501                 while(1):
502                     if(content == "\n" or content == "\t"):
503                         content = error_file_fd.readline
504                     else:
505
506                         content = content.lstrip("{")       # remove first "{"
507                         content_list = content.split(",")   # divide into pieces
508                         left = content_list[0].strip()      # remove whitespace
509                         if (left == '0' or left == '1'):    # remove non error codes
510                             content = error_file_fd.readline()
511                         else:
512                             if self._verbose == 1:
```

```
513                                 x += 1
514                                 # spinning line
515                                 if (0 == x%2):
516                                     if z == 0:
517                                         sys.stdout.write("-\r")
518                                         sys.stdout.flush()
519                                         z = 1
520                                     elif z == 1:
521                                         sys.stdout.write("\\\r")
522                                         sys.stdout.flush()
523                                         z = 2
524                                     elif z == 2:
525                                         sys.stdout.write("|\r")
526                                         sys.stdout.flush()
527                                         z = 3
528                                     elif z == 3:
529                                         sys.stdout.write("/\r")
530                                         sys.stdout.flush()
531                                         z = 4
532                                     elif z == 4:
533                                         sys.stdout.write("-\r")
534                                         sys.stdout.flush()
535                                         z = 5
536                                     elif z == 5:
537                                         sys.stdout.write("\\\r")
538                                         sys.stdout.flush()
539                                         z = 6
540                                     elif z == 6:
541                                         sys.stdout.write("|\r")
542                                         sys.stdout.flush()
543                                         z = 7
544                                     elif z == 7:
545                                         sys.stdout.write("/\r")
546                                         sys.stdout.flush()
547                                         z = 0
548                             sys.stdout.flush
549                             right = content_list[1].strip() # remove whitespace
550                             sql = "INSERT INTO error (e_number, e_name) VALUES (%s,
                                    \"%s\");" % (left, right)
551                             self._db_access.execute(sql)
552                             content = error_file_fd.readline()
553                             if(content == ''):
554                                 break
555
556                 error_file_fd.close()
557                 sql = "INSERT INTO error (e_number, e_name, e_description) VALUES (%s
                        , \"%s\", \"%s\");" % (999999, "unknown", "unknown error number")
558                 self._db_access.execute(sql)
559
560                 #insert project
561                 sql = 'INSERT INTO project (p_name) VALUES ("%s")' % project
562                 self._db_access.execute(sql)
```

```
563
564                 for i in range(len(serverlist)):
565                     # insert in host
566                     sql = 'INSERT INTO host (h_ip_address) VALUES ("%s")' %
                            serverlist[i][0]
567                     self._db_access.execute(sql)
568                     # get host id
569                     sql = 'SELECT * FROM host WHERE h_ip_address = "%s"' % serverlist
                            [i][0]
570                     data = self._db_access.execute(sql)
571                     data = self._db_access.fetchall()
572                     host_id = data[0][0]
573                     # get project id
574                     sql = 'SELECT * FROM project WHERE p_name = "%s"' % project
575                     data = self._db_access.execute(sql)
576                     data = self._db_access.fetchall()
577                     project_id = data[0][0]
578                     # connect host and project
579                     sql = 'INSERT INTO host_project (hp_h_id, hp_p_id) VALUES (%s, %s
                            )' % (host_id, project_id)
580                     data = self._db_access.execute(sql)
581
582                 if self._verbose == 1:
583                     print "\n%s -> Database new created !" % time.ctime()
584             except:
585                 print "%s -> Problem creating database!" % time.ctime()
586                 os.rmdir(self._database_path)
587                 os._exit(-1)
588         else:
589             try:
590                 #check if tables there
591                 # 1. connect to  database
592                 self._connect = sqlite.connect(databasename, autocommit=1)
593
594                 # 2. create access cursor
595                 self._db_access = self._connect.cursor()
596
597                 # 3. check if table messages is still there
598                 sql = "SELECT * FROM messages"
599                 self._db_access.execute(sql)
600                 data = self._db_access.fetchall()
601                 if (0 == len(data)):
602                     print "%s -> No data in table \"messages\" !" % (time.ctime())
603                 else:
604                     print "%s -> Database holds %s error messages !" % (time.ctime(),
                            len(data))
605
606                 # 4. check if table error is still there
607                 sql = "SELECT * FROM error"
608                 self._db_access.execute(sql)
609                 data = self._db_access.fetchall()
610                 if (0 == len(data)):
```

```
611                     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())
612                 else:
613                     print "%s -> Database holds %s defined error numbers !" % (time.
                            ctime(), len(data))
614
615                 # 5. check if table project is still there
616                 sql = "SELECT * FROM host_project"
617                 self._db_access.execute(sql)
618                 data = self._db_access.fetchall()
619                 if (0 == len(data)):
620                     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())
621                 else:
622                     print "%s -> Database holds %s defined connections between table
                            \"project\" and \"host\" !" % (time.ctime(), len(data))
623
624                 # 6. check if table host_project is still there
625                 sql = "SELECT * FROM project"
626                 self._db_access.execute(sql)
627                 data = self._db_access.fetchall()
628                 if (0 == len(data)):
629                     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)
630                     #insert project
631                     sql = 'INSERT INTO project (p_name) VALUES ("%s")' % project
632                     self._db_access.execute(sql)
633                 else:
634                     print "%s -> Database holds %s defined projects !" % (time.ctime
                            (), len(data))
635
636                 # 7. check if table host is still there
637                 sql = "SELECT * FROM host"
638                 self._db_access.execute(sql)
639                 data = self._db_access.fetchall()
640                 if (0 == len(data)):
641                     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())
642                 else:
643                     print "%s -> Database holds %s defined hosts !" % (time.ctime(),
                            len(data))
644
```

```
645                 # check if the is a new host in the log file
646                 for i in range(len(serverlist)):
647                     #check if host is there
648                     sql = 'SELECT * FROM host WHERE h_ip_address = "%s"' % serverlist
                            [i][0]
649                     self._db_access.execute(sql)
650                     data = self._db_access.fetchall()
651                     if (0 == len(data)):
652                         if self._verbose == 1:
653                             print "%s -> Insert new host \"%s\" into database !" % (
                                    time.ctime(), serverlist[i][0])
654                         # insert in host
655                         sql = 'INSERT INTO host (h_ip_address) VALUES ("%s")' % (
                                serverlist[i][0])
656                         self._db_access.execute(sql)
657                         # get host id
658                         sql = 'SELECT * FROM host WHERE h_ip_address = "%s"' %
                                serverlist[i][0]
659                         data = self._db_access.execute(sql)
660                         data = self._db_access.fetchall()
661                         host_id = data[0]["h_id"]
662                         # get project id
663                         sql = 'SELECT * FROM project WHERE p_name = "%s"' % project
664                         data = self._db_access.execute(sql)
665                         data = self._db_access.fetchall()
666                         project_id = data[0][0]
667                         # connect host and project
668                         sql = 'INSERT INTO host_project (hp_h_id, hp_p_id) VALUES (%s
                                , %s)' % (host_id, project_id)
669                         data = self._db_access.execute(sql)
670             except sqlite.DatabaseError, e:
671                 print "%s -> %s" % (time.ctime(), e)
672                 os._exit(-1)
673
674     def execute_sql(self, wait, database_obj, sql):
675         '''
676         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.
677         '''
678         for i in range(0, wait):
679             try:
680                 database_obj.execute(sql)
681                 return 0, database_obj
682             except sqlite.OperationalError:
683                 if self._verbose == 1:
684                     if i%20 == 0:
685                         text = "%s -> database temporary locked - keep trying for
                                another %d seconds ...." % (time.ctime(), wait-i)
686                         print text
687                 time.sleep(1)
688             except:
```

```
689                 if self._verbose == 1:
690                     print "%s -> database query execution error" % (time.ctime())
691
692         return -1, database_obj
693
694     def get_access_cursor(self):
695         '''
696         This function returns the database access cursor.
697         '''
698         return self._db_access
699
700     def get_database_path(self):
701         '''
702         This function returns the database path
703         '''
704         return self._database_path
705
706 ################## CLASS  ClientThread  ##################
707
708 class ClientThread(threading.Thread):
709     '''
710     This class gets the information from the server and puts it into the database !
711     '''
712     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):
713         '''
714         Constructor
715         '''
716
717         self._file_list = filelist
718         self._interval = interval
719         self._verbose = verbose
720         self._share = shared
721         self._address = address
722         self._port = port
723         self._db_access = db_object
724         self._client_certificate = cl_cert
725         self._client_certificate_path = cl_cert_path
726         self._client_ca = ca_cert
727         self._client_ca_path = ca_cert_path
728         threading.Thread.__init__(self)
729         # create XML-reader
730         self._xml_file_parser = make_parser()
731         # turn off namespace
732         self._xml_file_parser.setFeature(feature_namespaces, 0)
733         self._smtp_password = smtp_pass
734         self._mail_obj = []
735
736         if self._smtp_password != None:
737             for i in range(len(mail_address)):
```

```
738                 obj = Mail(mail_address[i], smtp_server, smtp_pass, smtp_from, user,
                        verbose)
739                 self._mail_obj.append((obj, mail_address[i]))
740 
741             for i in range(len(self._mail_obj)):
742                 name = self._address+"-"+self._mail_obj[i][1]
743                 self._mail_obj[i][0].create_content(name)
744 
745         # overwrite the default ContextHandler with my own
746         self._my_handler = MyContentHandler(self._db_access, self._address,
                mail_ignore_error, self._mail_obj, self._verbose )
747         self._xml_file_parser.setContentHandler(self._my_handler)
748 
749         self._stop_thread = False # variable to indecate thread termination
750 
751     def run(self):
752         '''
753         This functions overwrites the standard run method.
754         '''
755         filenames = []
756 
757         if ( (0 == len(self._file_list)) & (self._stop_thread == False)):
758             # if no old xml files fetch your own xml file
759             try:
760                 if self._verbose == 1:
761                     print "%s -> Client %d connecting to server %s" % (time.ctime(),
                            thread.get_ident(), self._address)
762                 try:
763                     #if self._verbose == 1:
764                     #    print "try to connect: ",self._address
765                     connect = self._connect_to_server(self._address, self._port)
766                     #if self._verbose == 1:
767                     #    print "connected -> ", connect
768                 except:
769                     if self._verbose == 1:
770                         print "%s -> Could not connect to host \"%s\"" % (time.ctime
                                (), self._address)
771                     if (self._smtp_password != None):
772                         for g in range(len(self._mail_obj)):
773                             self._mail_obj[g][0].delete_content()
774                     self._stop_thread = True
775 
776                 if (self._stop_thread == False):
777                     # get file names
778                     try:
779                         if self._verbose == 1:
780                             print "%s -> Get file names !!!" % time.ctime()
781                         filenames = connect.get_file_list()
782                         if ((-3 == filenames) & (self._stop_thread == False)):
783                             # server is busy parsing
784                             check = self._wait(connect)
785                             if check == 0:
```

```
786                             filenames = connect.get_file_list()
787                             if (-3 == xml_content):
788                                 # terminate thread
789                                 self._stop_thread = True
790                     if ((-2 == filenames) & (self._stop_thread == False)):
791                         if self._verbose == 1:
792                             print "%s -> RPC calls disabled !" % time.ctime()
793                             self._stop_thread = True
794                     if ((filenames == 0) & (self._stop_thread == False)):
795                         filenames = []
796                     if self._verbose == 1:
797                         print "%s -> %s files to fetch " % (time.ctime(), len(
                                filenames))
798                 except:
799                     if self._verbose == 1:
800                         print "%s -> Could not connect (check) to IP  \"%s\"" % (
                                time.ctime(), self._address)
801                     if (self._smtp_password != None):
802                         for g in range(len(self._mail_obj)):
803                             self._mail_obj[g][0].delete_content()
804                     self._stop_thread = True
805 
806             if ( (0 < len(filenames)) & (self._stop_thread == False)):
807                 # fetch files
808                 for g in range(len(filenames)):
809                     xml_content = connect.get_my_xml_file(filenames[g])
810                     if ( -3 == xml_content ):
811                         if self._verbose == 1:
812                             print "%s -> Parsing in progress ..." % time.ctime()
813                         check = self._wait(connect)
814                         if check == 0:
815                             xml_content = connect.get_my_xml_file(filenames[g])
816                             if (-3 == xml_content):
817                                 # terminate thread
818                                 self._stop_thread = True
819                                 break
820                     if ( -2 == xml_content ):
821                         if self._verbose == 1:
822                             print "%s -> RPC calls disabled !" % time.ctime()
823                         self._stop_thread = True
824                         break
825                     if (xml_content == "no file"):
826                         # there is no new file available
827                         if self._verbose == 1:
828                             print "%s -> No file available !!!" % time.ctime()
829                         self._stop_thread = True
830                         break
831 
832                     # name of temporary XML file
833                     name = "%s_client_xml_file_%d.xml" % (self._address, g)
834                     # lock critical section
835                     self._share.lock()
```

```
836                         try:
837                             c = g
838                             while(1):
839                                 if(0 == os.path.exists(name)):
840                                     # save xml file locally
841                                     name = "%s_client_xml_file_%d.xml" % (self.
                                            _address, c)
842                                     file_fd = open(name, 'w')
843                                     file_fd.write(xml_content)
844                                     file_fd.close()
845                                     self._file_list.append(name)
846                                     break
847                                 name = "%s_client_xml_file_%d.xml" % (self._address,
                                        c)
848                                 c += 1
849                         finally:
850                             # unlock critical section
851                             self._share.release()
852             except SSL.SSLError, e:
853                 if self._verbose == 1:
854                     print "%s -> Connection error (server \"%s\"): %s !" % (time.
                            ctime(), self._address, e)
855                 self._stop_thread = True
856             except:
857                 if self._verbose == 1:
858                     print "%s -> Error connecting to server -> \"%s\" !" % (time.
                            ctime(), self._address)
859                 self._stop_thread = True
860
861         if ( (0 < len(self._file_list)) & (self._stop_thread == False)):
862             # deal with own generated file list
863             for g in range(len(self._file_list)):
864                 name = self._file_list[g]
865                 if self._address == None:
866                     dbpath = "%s/"% self._db_access.get_database_path()
867                     ad = re.sub(dbpath, "", self._file_list[g])
868                     ad = re.sub('_client_xml_file_[0-9]+.xml', "", ad)
869                     self._my_handler.set_ip(ad)
870                 try:
871                     file_fd = open(name, 'r')
872                 except IOError, e:
873                     print e
874                     self._stop_thread = True
875                     break # aborts for or while loop
876
877                 # write in database
878                 z = 0
879                 while(1):
880                     try:
881                         if ((0 == self._share.set_variable(self)) & (self.
                                _stop_thread == False)):
882                             try:
```

```
883                             self._xml_file_parser.parse(file_fd)
884                         except xml.sax.SAXParseException,e :
885                             if self._verbose == 1:
886                                 print "%s -> sax parser error: %s" % ( time.ctime
                                        (), e)
887
888                         self._share.reset_variable()
889                         if (None != self._smtp_password):
890                             # if mail is sendable
891                             for a in range(len(self._mail_obj)):
892                                 self._mail_obj[a][0].send_mail(self._mail_obj[a
                                        ][1], self._address)
893
894                         file_fd.close()
895                         os.remove(name)
896                         if (self._smtp_password != None):
897                             for g in range(len(self._mail_obj)):
898                                 self._mail_obj[g][0].delete_content()
899                         break
900                     else:
901                         z += 1
902                         if z == 10:
903                             if self._verbose == 1:
904                                 print "%s -> can not access database -->
                                        terminating" % time.ctime()
905                             break
906
907                 except AssertionError:
908                     file_fd.close()
909                     if self._verbose == 1:
910                         print "%s -> can not access database -> terminating" %
                                time.ctime()
911                     if (self._smtp_password != None):
912                         for g in range(len(self._mail_obj)):
913                             self._mail_obj[g][0].delete_content()
914                     break
915
916                 except:
917                     file_fd.close()
918                     if self._verbose == 1:
919                         print "%s -> problem processing XML file -> terminating"
                                % time.ctime()
920                     if (self._smtp_password != None):
921                         for g in range(len(self._mail_obj)):
922                             self._mail_obj[g][0].delete_content()
923                     break
924
925         else:
926             if (self._smtp_password != None):
927                 for g in range(len(self._mail_obj)):
928                     self._mail_obj[g][0].delete_content()
929
```

```
930     def _wait(self, connect_object):
931         '''
932         This function waits if the server is busy parsing the log file (busy waiting)
                .
933         '''
934         counter = 0
935         max_sleeping_time = 60 * self._interval
936         sleeped = 0
937         while(1):
938             # check again
939             check = connect_object.rpc_check_availabitity()
940             if check == 0:
941                 return 0
942             counter += 1
943             if (counter == 30):
944                 if self._verbose == 1:
945                     print "%s -> Server takes a long time to parse file -> thread
                            terminating" % time.ctime()# --- debug ---
946                 return -1
947             # sleep for ten seconds and try again
948             sleeped += 10
949             if sleeped > max_sleeping_time:
950                 return -1
951             time.sleep(10)
952
953     def _connect_to_server(self, server, port):
954         '''
955         This function establishes the connection to the server.
956         '''
957         serverus = server
958         ctx = self.create_ctx()
959         # connect to server via SSL using the created context
960         urladdress = "https://%s:%d" % (serverus, port)
961         server = Server(urladdress, SSL_Transport(ctx))
962         # return server object
963         return server
964
965     def create_ctx(self):
966         '''
967         The function creates the necessary SSL context using certificates.
968         '''
969         ctx = SSL.Context(protocol='sslv3') # use SSLv3 only
970         ctx.load_cert(self._client_certificate_path+"/"+self._client_certificate)
                        # load client certificate
971         ctx.load_client_CA(self._client_ca_path+"/"+self._client_ca)        # load
                certificate authority private key
972       # if self._verbose == 1:
973           # ctx.set_info_callback()             # tell me what you're doing ---
                  debug ---
974         ctx.set_session_id_ctx('server')     # session name
975         return ctx
976
```

```
977 ################## CLASS  WorkerThread  ##################
978
979 class WorkerThread(threading.Thread):
980     '''
981     This class is responsible for starting the ClientThreads within a certain
            interval.
982     '''
983
984     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):
985         '''
986         Constructor
987         '''
988         self._verbose = verbose
989         self._share = shared
990         self._db_access = db_object
991         self._interval = interval
992         self._serverlist = serverlist
993         self._client_certificate = cl_cert
994         self._client_certificate_path = cl_cert_path
995         self._client_ca = ca_cert
996         self._client_ca_path = ca_cert_path
997         self._mail_address = mail_address
998         self._mail_ignore_error = mail_ignore_error
999         self._smtp_server = smtp_server
1000        self._smtp_pass = smtp_pass
1001        self._smtp_from = smtp_from
1002        self._smtp_user = user
1003        self._list = []
1004        threading.Thread.__init__(self)
1005
1006    def run(self):
1007        '''
1008        This function overwrites the standard run method.
1009        '''
1010
1011        temp_list = []
1012
1013        while(1):
1014            # deal with not processed, but fetched XML files first
1015            #find files
1016            os.path.walk(self._db_access.get_database_path(), self._parse_directory,
                    self._list)
1017            if (0 < len(self._list)):
1018                temp_list = copy.deepcopy(self._list)
1019                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)
```

```
1020                self._thread.start()
1021                del self._list[:] # delete list content
1022
1023            # then initiase new XML file fetching
1024            for i in range(len(self._serverlist)):
1025                dummy_list = []
1026                # start thread for fetching log file
1027                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)
1028                self._thread.start()
1029
1030            if self._verbose == 1:
1031                print "\n%s -> sleeping for %d minutes\n" % (time.ctime(), (self.
                        _interval))
1032            time.sleep(self._interval*60)
1033
1034
1035    def _parse_directory(self, arg, dirname, fnames):
1036        '''
1037        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)!
1038        '''
1039        d = os.getcwd()
1040        # change into log file directory
1041        try:
1042            os.chdir(dirname)
1043        except:
1044            print "could not find directory \"%s\"" % dirname
1045            return -1
1046        # for each file
1047        for f in fnames:
1048            # check if file and if file is a log file e.g. srbLog.20051003.gz
1049            if (not os.path.isfile(f)) or (None == re.search('client_xml_file_[0-9]+.
                    xml', f)):
1050                continue
1051            else:
1052            # save filename into an arrray (list)
1053                filus = dirname+"/"+f
1054                self._list.append(filus)
1055        # change back into the working directory
1056        os.chdir(d)
1057
1058
1059 ################## CLASS Mutex ##################
1060
```

```
1061 class Mutex:
1062     '''
1063     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.
1064     '''
1065     # database lock
1066     _db_locked = threading.Lock()
1067     # critical section lock
1068     _locked = threading.Lock()
1069 
1070     def __init__(self):
1071         '''
1072         Constructor
1073         '''
1074         self.writing = 0
1075         self._the_thread = 0
1076 
1077     def set_variable(self, threadus):
1078         '''
1079         set variable writing
1080         '''
1081         Mutex._db_locked.acquire()  # lock
1082         # if nobody is accessing the database
1083         if self.writing == 0:
1084             #set variable
1085             self.writing = 1
1086             self._the_thread = threadus
1087             Mutex._db_locked.release()
1088             return 0
1089         else:
1090             if (1 != self._the_thread.isAlive()):
1091                 # if the thread, which set the variable is dead, reset variable
1092                 self.writing = 0
1093             Mutex._db_locked.release()   # release lock
1094             return -1
1095 
1096     def reset_variable(self):
1097         '''
1098         reset variable writing
1099         '''
1100         Mutex._db_locked.acquire()  # lock
1101         self.writing = 0
1102         self._the_thread = 0
1103         Mutex._db_locked.release()
1104 
1105     def lock(self):
1106         '''
1107         This functions acquires the look.
1108         '''
1109         Mutex._locked.acquire()
1110 
```

```
1111     def release(self):
1112         '''
1113         This function releases the lock.
1114         '''
1115         Mutex._locked.release()
```

D.2.3 Module `utils_client.py`

LISTING D.7: Module `utils_client.py`

```
1  #!/usr/bin/env python
2  '''
3  This module provides basic funcitons for the client_classes.py and start_client.py.
4
5  Reading University
6  MSc in Network Centered Computing
7  a.weise - a.weise@reading.ac.uk - December 2005
8  '''
9
10 import ConfigParser, string, os, sys, termios
11
12 def LoadConfig(file_name, config={}):
13     """
14     returns a dictionary with key's of the form
15     <section>.<option> and the values
16
17     source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/65334
18     """
19     config = config.copy()
20     cp = ConfigParser.ConfigParser()
21     cp.read(file_name)
22     for sec in cp.sections():
23         name = string.lower(sec)
24         for opt in cp.options(sec):
25             config[name + "." + string.lower(opt)] = string.strip(cp.get(sec, opt))
26     return config
27
28 def check_ip(ip):
29     '''
30     This function check if a given IP is valid.
31     '''
32     try:
33         ip = ip.split(".")
34     except AttributeError:
35         return -1
36
37     for i in range(len(ip)):
38         check =  ip[i].find("0", 0, 1)
39         if -1 != check and 1 < len(ip[i]):
```

```
40              return -1
41          try:
42              ip[i] = int(ip[i])
43          except ValueError:
44              return -1
45          if ip[i] >= 0 and ip[i] <= 255:
46              pass
47          else:
48              return -1
49
50      return 0
51
52  def usage_exit(progname, msg=None):
53      '''
54      This function gives usage help and exits script.
55      '''
56      if msg:
57          print msg
58          print # lf cr
59      print "usage: python %s [ -h|--help -c|--config -p|--smtp_passord -v|--verbose -d
            |--daemon] \n\n" % progname
60      os._exit(-1)
61
62  def get_password(msg):
63      '''
64      This function reads from stdin without echoing the input.
65
66      source: http://gnu.kookel.org/ftp/www.python.org/doc/faq/library.html
67      modified by a. weise December 2005
68      '''
69      fd = sys.stdin.fileno()
70      # turn off stdin's echoing
71      old = termios.tcgetattr(fd)
72      new = termios.tcgetattr(fd)
73      new[3] = new[3] & ~termios.ICANON & ~termios.ECHO
74      new[6][termios.VMIN] = 1
75      new[6][termios.VTIME] = 0
76      termios.tcsetattr(fd, termios.TCSANOW, new)
77      s = ''    # save the characters typed and add them together
78      try:
79          print
80          print msg
81          while 1:
82              c = os.read(fd, 1)
83              if c == "\n":
84                  break
85              s = s+c
86      finally:
87          # turn on stdin's echoing again
88          termios.tcsetattr(fd, termios.TCSAFLUSH, old)
89          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.
```

```
4
5  Reading University
6  MSc in Network Centred Computing
7  a.weise - a.weise@reading.ac.uk - December 2005
8  '''
9  import gui_classes
10 import os, getopt, sys, re, time
11
12 # database
13 import sqlite
14
15 # functions
16 from gui_utils import usage_exit, check_date, convert_date, check_ip, find_item,
       help_context, check_time, LoadConfig
17
18 class Display:
19     '''
20     This is main class for the gui application.
21     '''
22
23     def __init__(self, config):
24         '''
25         Constructor
26         '''
27         workingpath = os.getcwd()
28
29         self._database_name = config.get("database.name")
30         self._database_path = config.get("database.path")
31         self._database_path = self._database_path.rstrip("/")
32         if(config.get("database.path") == '' or config.get("database.path") == None):
33             # field is empty
34             self._database_path = workingpath
35
36         else:
37             self._database_name = self._database_name.strip()
38             if (-1 != self._database_path.find("/", 0, 1)):
39                 # first character "/"
40                 pass
41             else:
42                 self._database_path = workingpath+"/"+self._database_path
43
44         if(0 == os.access((self._database_path+"/"+self._database_name), 4)):     # 4
               -> R_OK
45             print "\nCould not access database under \"%s\" !\nMaybe change
                   configuration file and try again!\n\n" % (self._database_path+"/"+
                   self._database_name)
46             os._exit(-1)
47
48     def execute_sql(self, wait, database_obj, sql, col):
49         '''
50         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.
51          '''
52
53          for i in range(0, wait):
54              try:
55                  database_obj.execute(sql)
56                  data = database_obj.fetchall()
57                  return data
58              except sqlite.OperationalError:
59                  if i%10 == 0:
60                      text = "database temporary locked - keep trying for another %d
                            seconds ...." % (wait-i)
61                      if col == 1:
62                          col_obj = gui_classes.Colour()
63                          text = col_obj.yellow(text)
64                      print text
65                  time.sleep(1)
66
67          text = "Database busy, could not apply request. Please try again."
68          if col == 1:
69              text = col_obj.yellow(text)
70
71          print text,"\n\n"
72          os._exit(0)
73
74      def sql_host(self, col):
75          '''
76          This function gets all hosts from the database.
77          '''
78          sql = ' SELECT * FROM host, host_project, project'\
79                ' WHERE host.h_id = host_project.hp_h_id '\
80                ' AND host_project.hp_p_id = project.p_id'
81
82          database = self._database_path+"/"+self._database_name
83
84          connect = sqlite.connect(database, autocommit = 1)
85          db_access = connect.cursor()
86          data = self.execute_sql(120, db_access, sql, col)
87          return data
88
89
90      def sql_project(self, col):
91          '''
92          This function gets all projects from the database
93          '''
94          sql = ' SELECT * FROM project '
95          database = self._database_path+"/"+self._database_name
96          connect = sqlite.connect(database, autocommit = 1)
97          db_access = connect.cursor()
98          data = self.execute_sql(120, db_access, sql, col)
99          return data
100
```

```
101
102     def sql_error(self, col, d1 = None, d2 = None, t1 = None, t2 = None, host = None,
            project = None, error = None):
103         '''
104         This function gets all error messages from the database.
105         '''
106         where = 0
107         sql = ' SELECT * FROM error, messages, host, host_project, project '
108
109         if d1 != None and d2 != None:
110             # between date1 and date2
111             sql += ' WHERE '
112             where = 1
113             sql += ' messages.m_date BETWEEN "%s" AND "%s" ' % (d1, d2)
114
115         elif d1 != None and d2 == None:
116             # from date1 until now
117             sql += ' WHERE '
118             where = 1
119             now = time.strftime("%Y-%m-%d", time.localtime())
120             sql += ' messages.m_date BETWEEN "%s" AND "%s" ' % (d1, now)
121
122         elif d1 == None and d2 != None:
123             # until date2
124             sql += ' WHERE '
125             where = 1
126             start_ = "1970-01-01"
127             sql += ' messages.m_date BETWEEN "%s" AND "%s" ' % (start_, d2)
128
129         else:
130             pass
131
132         if where == 0:
133             sql += ' WHERE '
134             where = 1
135         else:
136             sql += ' AND '
137
138         if t1 != None and t2 != None:
139             # between date1 and date2
140             sql += ' messages.m_time BETWEEN "%s" AND "%s" ' % (t1, t2)
141             sql += ' AND '
142
143         elif t1 != None and t2 == None:
144             # from date1 until now
145             now = time.strftime("%H:%M:%S", time.localtime())
146             sql += ' messages.m_time BETWEEN "%s" AND "%s" ' % (t1, now)
147             sql += ' AND '
148
149         elif t1 == None and t2 != None:
150             # until date2
151             start_ = "00:00:00"
```

```
152             sql += ' messages.m_time BETWEEN "%s" AND "%s" ' % (start_, t2)
153             sql += ' AND '
154 
155         sql += ' messages.error_e_id = error.e_id '
156 
157         if error != None:
158             sql += ' AND ( '
159             for i in range(len(error)):
160                 sql += ' error.e_number = \"%s\" ' % error[i]
161                 if len(error) > (i+1):
162                     sql += ' OR '
163             sql += ' ) '
164 
165         if host != None:
166             sql += ' AND messages.host_h_id = host.h_id AND host.h_ip_address = "%s"
                    ' % host
167         elif host == None:
168             sql += ' AND messages.host_h_id = host.h_id '
169 
170         sql += ' AND host.h_id = host_project.hp_h_id  '\
171                ' AND host_project.hp_p_id = project.p_id '
172 
173         if project != None:
174             sql += ' AND project.p_name = "%s" ' % project
175 
176 
177         sql += 'ORDER BY messages.m_date, messages.m_time'
178 
179         database = self._database_path+"/"+self._database_name
180         connect = sqlite.connect(database, autocommit = 1)
181         db_access = connect.cursor()
182         data = self.execute_sql(120, db_access, sql, col)
183         return data
184 
185     def display_graph(self, dataset, col, file_fd = None):
186         '''
187         This function displays a barchart diagram containing Error Numbers and the
                corresponding Frequency
188         '''
189         field = []
190         field_label = []
191         table_error = []
192         for i in range(len(dataset)):
193             # prepare data
194             index = find_item(int(dataset[i]['error.e_number']), field)
195             if (None == index):
196                 field.append([int(dataset[i]['error.e_number']), 1])
197                 field_label.append([int(dataset[i]['error.e_number']), 1])
198                 table_error.append( [int(dataset[i]['error.e_number']), 1, dataset[i
                        ]['error.e_name']])
199             else:
200                 count = field[index][1]
```

```
201                 count += 1
202                 field[index][1] = count
203                 field_label[index][1] = count
204                 table_error[index][1] = count
205 
206         field.sort()
207         field_label.sort()
208         table_error.sort()
209 
210         h_line = "
                ---------------------------------------------------------------------------
                "
211         v_line = "|"
212         header = "\nFrequency of Errors: \n"
213 
214         if file_fd != None:
215             content = header
216             content += "\n\n Nr.   | Error Number\t| Frequency\t| Error Name\t\t\t\n\
                    n"
217             file_fd.write(content)
218 
219         if col == 1:
220             col_obj = gui_classes.Colour()
221             header = col_obj.yellow(header)
222             h_line = col_obj.yellow(h_line)
223             v_line = col_obj.yellow(v_line)
224 
225         print header
226         print h_line
227         print " Nr.   "+v_line+" Error Number\t"+v_line+" Frequency\t"+v_line+" Error
                 Name\t\t\t"
228         print h_line
229 
230         for i in range(len(table_error)):
231             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])
232 
233             if file_fd != None:
234                 content = " %5d | %7s\t| %6s\t| %s\n" % ((i+1), table_error[i][0],
                            table_error[i][1], table_error[i][2])
235                 file_fd.write(content)
236 
237         print h_line
238 
239         for i in range(len(field)):
240             field_label[i][0] = (i+1)
241             field_label[i][1] = "%d" % field[i][0]
242             field[i][0] = (i+1)
243 
244         window = []
245 
246         pic_obj = gui_classes.Picture(col, window)
```

```
247         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\"")
248         pic_obj.mainloop()
249
250 ########################################################################
251
252 def start():
253
254     '''
255     Start the application.
256     '''
257     verbose = 0
258     col = 1
259     graph = 0
260     filus = None
261     configfile = ""
262     sql_host = None
263     sql_project = None
264     sql_error = None
265     sql_error_freq = None
266     date1 = None
267     date2 = None
268     time1 = None
269     time2 = None
270     ip = None
271     port = None
272     project = None
273     error = None
274
275     # evaluate parameters
276     try:
277         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='])
278         for opt, value in opts:
279             if opt in ('', '--nocolor'):
280                 col = 0
281             if opt in ('-h','--help'):
282                 msg = help_context(col)
283                 usage_exit(sys.argv[0], msg, col)
284             if opt in ('-c', '--config'):
285                 value = value.replace("=", "")
286                 configfile = os.getcwd()+"/"+value
287             if opt in ('-v', '--verbose'):
288                 verbose = 1
289             if opt in ('-g', '--graph'):
290                 graph = 1
291
292         for opt, value in opts:
293             if opt in ('', '--sql_host'):
```

```
294                 sql_host = 1
295             if opt in ('', '--sql_project'):
296                 sql_project = 1
297             if opt in ('', '--sql_error'):
298                 sql_error = 1
299             if opt in ('', '--sql_error_freq'):
300                 sql_error_freq = 1
301             if opt in ('', '--error'):
302                 error = value
303                 error = error.strip()
304                 error = error.strip(",")
305                 error = error.split(",")
306                 for i in range(len(error)):
307                     error[i] = error[i].strip()
308                     try:
309                         error[i] = int(error[i])
310                     except ValueError, e:
311                         #'given error is not valid'
312                         usage_exit(sys.argv[0], 'invalid literal for int()' , col)
313             if opt in ('', '--start_date'):
314                 date1 = value
315                 status = re.search('^[0-3][0-9].[0-1][0-9].[1-9][0-9]{3}', date1)
316                 if (None == status):
317                     usage_exit(sys.argv[0], 'given date is not valid', col)
318                 else:
319                     date1 = status.string[status.start():status.end()]
320                     if (0 == check_date(date1)):
321                         date1 = convert_date(date1)
322                     else:
323                         usage_exit(sys.argv[0], 'given date is not valid', col)
324             if opt in ('','--end_date'):
325                 date2 = value
326                 status = re.search('^[0-3][0-9].[0-1][0-9].[1-9][0-9]{3}', date2)
327                 if (None == status):
328                     usage_exit(sys.argv[0], 'given date is not valid', col)
329                 else:
330                     date2 = status.string[status.start():status.end()]
331                     if (0 == check_date(date2)):
332                         date2 = convert_date(date2)
333                         print "date 2: ", date2
334                     else:
335                         usage_exit(sys.argv[0], 'given date is not valid', col)
336             if opt in ('', '--start_time'):
337                 time1 = value
338                 status = re.search('^[0-2][0-9]:[0-5][0-9]:[0-5][0-9]', time1)
339                 if (None == status):
340                     usage_exit(sys.argv[0], 'given time is not valid', col)
341                 else:
342                     time1 = status.string[status.start():status.end()]
343                     if ( 0 == check_time(time1)):
344                         pass
345                     else:
```

```
346                     usage_exit(sys.argv[0], 'given time is not valid', col)
347             if opt in ('', '--end_time'):
348                 time2 = value
349                 status = re.search('^[0-2][0-9]:[0-5][0-9]:[0-5][0-9]', time2)
350                 if (None == status):
351                     usage_exit(sys.argv[0], 'given time is not valid', col)
352                 else:
353                     time2 = status.string[status.start():status.end()]
354                     if ( 0 == check_time(time2)):
355                         pass
356                     else:
357                         usage_exit(sys.argv[0], 'given time is not valid', col)
358             if opt in ('','--ip'):
359                 ip = value
360                 status = re.search('^[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}', ip
                        )
361                 if (None == status):
362                     usage_exit(sys.argv[0], 'given IP is not valid', col)
363                 else:
364                     ip = status.string[status.start():status.end()]
365                     if (0 == check_ip(ip)):
366                         print "ip: ", ip
367                     else:
368                         usage_exit(sys.argv[0], 'given IP is not valid', col)
369             if opt in ('', '--port'):
370                 port = int(value)
371                 if (port < 1024 or port > 50001):
372                     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)
373             if opt in ('','--project'):
374                 project = value
375             if opt in  ('', '--file'):
376                 filus = value
377     except getopt.error, e:
378         e = "%s" % (e)
379         usage_exit(sys.argv[0], e, col)
380     except ValueError, e:
381         e = "%s" % (e)
382         usage_exit(sys.argv[0], e, col)
383
384
385     # load config file or default values
386     if (configfile != ""):
387         # check if file exists
388         if(1 == os.path.exists(configfile)):
389             config = LoadConfig(configfile)
390         else:
391             # if file NOT exists terminate program
392             print "\n\nSorry, a given config file does NOT exist !\nPlease try again
                    !\n\n"
393             os._exit(-1)
```

```
394     else:
395         msg = "\nNo config file spezified !\n"
396         usage_exit(sys.argv[0], msg, col)
397 
398     if col == 1:
399         col_obj = gui_classes.Colour()
400 
401     gui = Display(config)
402 
403     if verbose == 1:
404         i = 1
405         d = config.iteritems()
406         while(1):
407             try:
408                 print i, ". ", d.next()
409                 i += 1
410             except:
411                 break
412 
413     if filus != None:
414         # save output in file
415         # check if file exists
416         try:
417             filus_fd = file(filus, 'r')
418             quest = "File \"%s\" already exists, overwrite file (y/n) ? -> " % filus
419             if col == 1:
420                 quest = col_obj.darkred(quest)
421             decision = raw_input(quest)
422             if decision == 'y' or decision == 'Y' or decision == 'yes' or decision ==
                    'Yes' or decision == 'YES':
423                 filus_fd.close()
424                 filus_fd = file(filus, 'w')
425             else:
426                 os._exit(0)
427         except IOError:
428             filus_fd = file(filus, 'w')
429 
430     #------------------------------------ SQL COMMANDS
        ------------------------------------#
431 
432     if (1 == sql_host):
433         print "sql_host: ", sql_host
434         data = gui.sql_host(col)
435 
436         if len(data) == 0:
437             text = "\n\nSorry, no data available for you request!"
438             if col == 1:
439                 text = col_obj.yellow(text)
440             print text
441             print "\n\n"
442             os._exit(0)
443 
```

```
444         h_line = "
                ------------------------------------------------------------------------
                "
445         v_line = "|"
446
447         if col == 1:
448             h_line = col_obj.yellow(h_line)
449             v_line = col_obj.yellow(v_line)
450
451         # table head
452         print h_line
453         print " Nr.\t"+v_line+"\tHost IP\t\t\t"+v_line+"     Host Name      "+v_line+
                "      Project "
454         print h_line
455
456         # table data
457         for i in range(len(data)):
458             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'])
459
460         print h_line
461
462     elif (1 == sql_project):
463         print "sql_project: ", sql_project
464         data = gui.sql_project(col)
465
466         if len(data) == 0:
467             text = "\n\nSorry, no data available for you request!"
468             if col == 1:
469                 text = col_obj.yellow(text)
470             print text
471             print "\n\n"
472             os._exit(0)
473
474         h_line = "----------------------------------------"
475         v_line = "|"
476
477         if col == 1:
478             h_line = col_obj.yellow(h_line)
479             v_line = col_obj.yellow(v_line)
480
481         # table head
482         print h_line
483         print v_line+" Nr.\t"+v_line+"\tProject\t\t\t"+v_line
484         print h_line
485
486         # table data
487         for i in range(len(data)):
488             print "%s %d\t%s\t%s\t\t%s" % (v_line, (i+1), v_line, data[i]['p_name'],
                    v_line)
489
```

```
490         print h_line
491
492
493     elif (1 == sql_error):
494
495         data = gui.sql_error(col, d1=date1, d2=date2, t1 = time1, t2 = time2, host =
                ip, project = project, error = error)
496
497         if len(data) == 0:
498             text = "\n\nSorry, no data available for you request!"
499             if col == 1:
500                 text = col_obj.yellow(text)
501             print text
502             print "\n\n"
503             os._exit(0)
504
505         # table head
506         h_line = "
                ----------------------------------------------------------------------------------------------------
                "
507         v_line = "|"
508         h_line_short = "-"
509         header = "  Nr.  |    Date    |   Time   |\t\t\t\tError String"
510         ln = "LN"
511         en = "EN"
512         ip = "IP"
513         pr = "PR"
514
515         brown_line = h_line
516         if col == 1:
517             brown_line = col_obj.brown(h_line)
518             h_line = col_obj.yellow(h_line)
519             v_line = col_obj.yellow(v_line)
520             h_line_short = col_obj.yellow(h_line_short)
521             header = col_obj.yellow(header)
522             ln = col_obj.yellow(ln)
523             en = col_obj.yellow(en)
524             ip = col_obj.yellow(ip)
525             pr = col_obj.yellow(pr)
526
527         print h_line
528
529         print header
530
531         if filus != None:
532             content = " Nr.\t|    Date    |   Time   | \t\t\t\tError String\t\t\t\t |
                    Line Number | Error Number |     Host IP     | Project\n\n"
533             filus_fd.write(content)
534         print h_line
535
536         # table data console
537         for i in range(len(data)):
```

```
538             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'])
539             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'])
540
541             # table data file
542             if filus != None:
543                 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'])
544                 filus_fd.write(content)
545             if len(data) > (i+1):
546                 print brown_line
547
548
549         print h_line
550
551         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)
552
553         if graph == 1:
554             if filus != None:
555                 gui.display_graph(data, col, file_fd = filus_fd)
556             else:
557                 if filus != None:
558                     filus_fd.close()
559                 gui.display_graph(data, col)
560
561     elif ( 1 == sql_error_freq):
562
563         data = gui.sql_error(col, d1=date1, d2=date2, t1 = time1, t2 = time2, host =
                ip, project = project, error = error)
564
565         if len(data) == 0:
566             text = "\n\nSorry, no data available for you request!"
567             if col == 1:
568                 text = col_obj.yellow(text)
569             print text
570             print "\n\n"
571             os._exit(0)
572
573
574         print "datasets: ", len(data)
575         #print data
576         if graph == 0:
577             field = []
578             field_label = []
```

```
579             table_error = []
580             for i in range(len(data)):
581                 index = find_item(int(data[i]['error.e_number']), field)
582                 if (None == index):
583                     field.append([int(data[i]['error.e_number']), 1])
584                     field_label.append([int(data[i]['error.e_number']), 1])
585                     table_error.append( [int(data[i]['error.e_number']), 1, data[i]['
                            error.e_name']])
586                    # print field
587                 else:
588                     count = field[index][1]
589                     count += 1
590                     field[index][1] = count
591                     field_label[index][1] = count
592                     table_error[index][1] = count
593 
594             field.sort()
595             field_label.sort()
596             table_error.sort()
597 
598             h_line = "
                    ---------------------------------------------------------------------------
                    "
599             v_line = "|"
600             header = "\nFrequency of Errors: \n"
601 
602             if col == 1:
603                 header = col_obj.yellow(header)
604                 h_line = col_obj.yellow(h_line)
605                 v_line = col_obj.yellow(v_line)
606 
607             print header
608             print h_line
609             print " Nr.   "+v_line+" Error Number\t"+v_line+" Frequency\t"+v_line+"
                    Error Name\t\t\t"
610             print h_line
611 
612             if filus != None:
613                 content = " Nr.   | Error Number\t| Frequency\t| Error Name\t\t\t\n\n
                        "
614                 filus_fd.write(content)
615 
616             # print table console
617             for i in range(len(table_error)):
618                 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])
619 
620                 # print table in file
621                 if filus != None:
622                     content = " %5d | %7s\t| %6s\t| %s" % ((i+1), table_error[i][0],
                            table_error[i][1], table_error[i][2])
623                     filus_fd.write(content)
```

```
624
625             print h_line
626
627             if filus != None:
628                 filus_fd.close()
629
630         elif graph == 1:
631             if filus != None:
632                 gui.display_graph(data, col, file_fd = filus_fd)
633             else:
634                 gui.display_graph(data, col)
635
636 if __name__ == '__main__':
637
638     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):
        '''
```

```
48          dye green
49          '''
50          return self.color['green']+text+self.color['reset']
51
52      def red(self, text):
53          '''
54          dye red
55          '''
56          return self.color['red']+text+self.color['reset']
57
58      def bold(self, text):
59          '''
60          dye bold
61          '''
62          return self.color['bold']+text+self.color['reset']
63
64      def teal(self, text):
65          '''
66          dye teal
67          '''
68          return self.color['teal']+text+self.color['reset']
69
70      def turquoise(self, text):
71          '''
72          dye turquoise
73          '''
74          return self.color['turquoise']+text+self.color['reset']
75
76      def fuscia(self, text):
77          '''
78          dye fuscia
79          '''
80          return self.color['fuscia']+text+self.color['reset']
81
82      def purple(self, text):
83          '''
84          dye purple
85          '''
86          return self.color['purple']+text+self.color['reset']
87
88      def darkred(self, text):
89          '''
90          dye darkred
91          '''
92          return self.color['darkred']+text+self.color['reset']
93
94      def darkblue(self, text):
95          '''
96          dye darkblue
97          '''
98          return self.color['darkblue']+text+self.color['reset']
99
```

```
100     def blue(self, text):
101         '''
102         dye blue
103         '''
104         return self.color['blue']+text+self.color['reset']
105 
106     def darkgreen(self, text):
107         '''
108         dye darkgreen
109         '''
110         return self.color['darkgreen']+text+self.color['reset']
111 
112     def yellow(self, text):
113         '''
114         dye yellow
115         '''
116         return self.color['yellow']+text+self.color['reset']
117 
118     def brown(self, text):
119         '''
120         dye brown
121         '''
122         return self.color['brown']+text+self.color['reset']
123 
124 
125 class Picture(Tkinter.Tk):
126     '''
127     This class provides functions around the "diplay diagrams" issues.
128     '''
129 
130     def __init__(self, color, windows):
131         '''
132         Constructor
133         '''
134         self._col = color
135         self._windows = []
136         # needed to close all windows properly
137         self._all_windows = windows
138         # needed for deactivate and activate all window buttons properly
139         self._all_windows.append(self)
140         # initialise tkinter
141         Tkinter.Tk.__init__(self)
142         # set min size
143         self.minsize(width=500, height=400)
144         #create frame, where the diagram is drawn later
145         self.framus = Tkinter.Frame(self)
146         # add frame to dialog
147         self.framus.grid(
148             column = 0,
149             row = 0,
150             columnspan = 7,
151             sticky = "news" #north east west south
```

```
152             )
153         # "QUIT" BUTTON
154         self.button_quit = Tkinter.Button(self, text="quit")
155         self.button_quit.grid(
156             column = 6,
157             row = 1,
158             columnspan = 1,
159             sticky = "e"
160                         )
161         # tooltips for "QUIT" button
162         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)
163         self.button_quit.configure(command = self.pre_shutdown)
164         # "SAVE AS" BUTTON
165         self.button_save = Tkinter.Button(self, text = "save as")
166         self.button_save.grid(
167             column = 5,
168             row = 1,
169             columnspan = 1,
170             sticky = "e"
171                         )
172         self.button_save.configure(command = self.save_as)
173         # status bar
174         self.status = Tkinter.Label(self)
175         self.status.grid(
176             column = 0,
177             row = 3,
178             columnspan = 7,
179             sticky = "w"
180                         )
181         # configure grid
182         self.grid_columnconfigure(0, weight = 1)
183         self.grid_rowconfigure(0, weight = 1)
184
185         # overwrite function
186         self.protocol("WM_DELETE_WINDOW", self.shutdown)
187
188         # saves as button hoover method
189         self.button_save.bind("<Enter>", lambda event, t="save diagram as postscript
                file": self._show_description_two(event, t))
190         self.button_save.bind("<Leave>", lambda event, t="": self.
                _show_description_two(event, t))
191         # tooltips for "SAVE AS" button
192         tooltips.ToolTip(self.button_save, follow_mouse = 1, text = "Please press \"
                save as\" to save the diagram as a postscript file.", delay = 3500)
193
194     def deactivate(self):
195         '''
196         This function deactivates all buttons.
197         '''
198         self.protocol("WM_DELETE_WINDOW", self._dummy)
```

```
199         self.button_quit.configure(command = self._dummy)
200         self.button_save.configure(command = self._dummy)
201     if self._select_type == 'error' or self._select_type == 'date':
202             self.button_select.configure(command = self._dummy)
203 
204     def activate(self):
205         '''
206         This function activates all buttons.
207         '''
208         self.protocol("WM_DELETE_WINDOW", self.shutdown)
209         self.button_quit.configure(command = self.pre_shutdown)
210         self.button_save.configure(command = self.save_as)
211         if self._select_type == 'error':
212             self.button_select.configure(command = self._select_error)
213         elif self._select_type == 'date':
214             self.button_select.configure(command = self._select_date)
215 
216     def save_as(self):
217         '''
218         This function saves the diagram picture as postscript.
219         '''
220         # deactivate all buttons
221         try:
222             for i in range(len(self._all_windows)):
223                 self._all_windows[i].deactivate()
224         except Tkinter.TclError:
225             pass
226         # save as dialog
227         result = tkFileDialog.asksaveasfilename(filetypes = [('postscript', '*.ps')],
                title = 'Save graph as ... ')
228         # activate all buttons
229         try:
230             for i in range(len(self._all_windows)):
231                 self._all_windows[i].activate()
232         except Tkinter.TclError:
233             pass
234 
235         if result != '':
236             # save diagram in file
237             self.graph.canvas.postscript(file = result, colormode = 'color')
238             ###Graphs.canvas.postscript(file = result, colormode = 'color')
239 
240     def _dummy(self, event = None):
241         '''
242         This function is doing nothing, it serves as a dummy.
243         '''
244         return 'break'
245 
246     def show_barchart(self, window_name, listus, label, xlabel, ylabel, data,
            select_type=None, filus_fd = None, descript = None):
247         '''
248         This function shows a barchart diagram.
```

```
249         '''
250         self.title(window_name)
251         self._data = data
252         self._file_fd = filus_fd
253   self._select_type = select_type
254         # generate barchart diagram
255         line = Graphs.GraphBars(listus, color ='green', size = 6)
256         graphObject = Graphs.GraphObjects([line])
257         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)
258         self.graph.pack(side = Tkinter.LEFT, fill = Tkinter.BOTH, expand = Tkinter.
                YES)
259         self.graph.draw(graphObject, 'automatic', 'automatic')
260
261         # sort items for listbox
262         self.items = []
263         self.search_label = []
264         self.search_value = []
265         for i in range(len(label)):
266             self.items.append(label[i][1])
267             self.search_value.append(listus[i])
268             self.search_label.append(label[i])
269
270         self.create_listbox()
271
272     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):
273         '''
274         This functions shows a line chart diagram.
275
276         window_name = name of the new window
277         listus = value list
278         label = label list for x-axis
279         xlabel = description of x-axis
280         ylabel = description of y-axis
281         select_type = type of items are listed in listbox
282         data = dataset which comes from the database query
283         error = chosen error from listbox
284         labelamount = amount of possible labels for the x-axis
285         '''
286         self.title(window_name)
287         self._data = data
288         self._file_fd = filus_fd
289   self._select_type = select_type
290
291         values = []
292
293         # only draw a dot where is a real value
294         for i in range(len(listus)):
295             if listus[i][1] != 0 :
```

```
296                 values.append(listus[i])
297
298         dot = Graphs.GraphSymbols(values,  color = 'green', marker = 'dot', fillcolor
                = 'darkgreen')
299
300         if len(listus) > 1:
301             line = Graphs.GraphLine(listus, color='green', size=6)
302             graphObject = Graphs.GraphObjects([line, dot])
303         else:
304             graphObject = Graphs.GraphObjects([dot])
305
306         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)
307         self.graph.pack(side = Tkinter.LEFT, fill = Tkinter.BOTH, expand = Tkinter.
                YES)
308         self.graph.draw(graphObject, 'automatic', 'automatic')
309
310         if select_type == "date":
311
312             self.items = []
313             self.search_label = []
314             self.search_value = []
315             #rearrange labels for listbox
316             for i in range(len(label)):
317                 if listus[i][1] != 0 :
318                     self.items.append(label[i][1])
319                     self.search_value.append(listus[i])
320                     self.search_label.append(label[i])
321
322             # create listbox with dates
323             self.create_listbox(error)
324
325     def create_listbox(self,  error = None):
326         '''
327         This function creates a listbox with the given items.
328         '''
329          # listbox
330         list_scrollbar = Tkinter.Scrollbar(self, orient=Tkinter.VERTICAL)
331         list_scrollbar.grid ( row = 1, column = 1, columnspan = 1, sticky = "ns" )
332
333         self.listbox = Tkinter.Listbox(self, height = 4, cursor = "plus", bg = "#
                ffffff", bd = 1, highlightcolor = "#00ff00",  yscrollcommand=
                list_scrollbar.set)
334         self.listbox.grid(
335             column = 0,
336             row = 1,
337             columnspan = 1,
338             sticky = "news"
339                         )
340
```

```
341         self.listbox.bind("<Enter>", self._show_description)
342         self.listbox.bind("<Leave>",  lambda event, t="": self._show_description_two(
                event, t))
343 
344         list_scrollbar["command"] = self.listbox.yview
345 
346         # "PLOT" button
347         self.button_select = Tkinter.Button(self, text = "plot")
348         self.button_select.grid(
349             column = 3,
350             row = 1,
351             columnspan = 1,
352             sticky = "w"
353                         )
354 
355         self._the_error = error
356 
357         if self._select_type == 'error':
358             self.button_select.configure(command = self._select_error)
359         elif self._select_type == 'date':
360             self.button_select.configure(command = self._select_date)
361 
362         self.button_select.bind("<Enter>", lambda event, t="plot new diagram": self.
                _show_description_two(event, t))
363         self.button_select.bind("<Leave>", lambda event, t="": self.
                _show_description_two(event, t))
364         # tooltips for "PLOT" button
365         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)
366 
367         # OPTION (dropdown) menu
368         if self._select_type == 'error':
369             self._ldate = "%15s" % ("error")
370             tooltips.ToolTip(self.listbox, follow_mouse = 1, text = "Please select an
                    error and then press \"plot\" to view this error number only.")
371             tooltips.ToolTip(self.listbox, follow_mouse = 1, text = "Please select an
                    error and then press \"plot\" to view this error number only.")
372         elif self._select_type == 'date':
373             self._ldate = "%15s" % ("date")
374             tooltips.ToolTip(self.listbox, follow_mouse = 1, text = "Please select a
                   date and then press \"plot\" to view this date only.")
375         self._lfreq = "%12s" % ("frequency")
376         self.var = Tkinter.StringVar(self)
377         # activate a trace, which monitores the changes, so in case the drop down
                menu is used a function is called
378         self.var.trace('w', self.menu_change)
379         self.var.set(self._ldate) # initial value
380 
381         option = Tkinter.OptionMenu(self, self.var, self._ldate, self._lfreq)
382 
383         # binding for status bar
```

```
384         option.bind("<Enter>", self._show_dropdown_description)
385         option.bind("<Leave>", lambda event, t="": self._show_description_two(event,
                t))
386 
387         if self._select_type == 'error':
388             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
                    )." )
389         elif self._select_type == 'date':
390             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)." )
391 
392         option.grid(
393             column = 2,
394             row = 1,
395             columnspan = 1,
396             sticky = "w"
397                    )
398 
399         # SPACE LABEL
400         labelus = Tkinter.Label(self)
401         labelus.grid(
402             column = 4,
403             row = 1,
404             columnspan = 1,
405             sticky = "news"
406                     )
407 
408     def menu_change(self, name, index, mode):
409         '''
410         This function changes the order in the listbox according to the chosen item
                in the drop down menu.
411         '''
412         temp_listus = []
413         temp_search_label = []
414 
415         change = self.var.get()
416         # for dates
417         if change == self._ldate:
418 
419             if self._select_type == 'error':
420                 self.search_label.sort(second_string_to_int)
421                 self._dropdown_description = "change order in listbox, currently
                        ordered by \"error number\""
422             elif self._select_type == 'date':
423                 self.search_label.sort(second_string_only)
424                 self._dropdown_description = "change order in listbox, currently
                        ordered by \"date\""
425 
426             for i in range(len(self.search_label)):
```

```
427                 # save label
428                 temp = self.search_label[i][1]
429                  # search for corresponding label in label array
430                 for j in range(len(self.search_value)):
431                     if self.search_label[i][0] == self.search_value[j][0]:
432                         # save corresponding label
433                         temp_listus.append([i+1, self.search_value[j][1]])
434                 # adjust items
435                 temp_search_label.append([i+1, temp])
436 
437                 self.items[i] = "%s  (%s)" % (temp_search_label[len(temp_search_label
                        )-1][1], temp_listus[len(temp_listus)-1][1])
438 
439             self.search_value = temp_listus[:]
440             self.search_label = temp_search_label[:]
441 
442             # delete old items and write new items in listbox
443             self.listbox.delete(0, Tkinter.END)
444             for i in range(len(self.items)):
445                 self.listbox.insert(Tkinter.END, self.items[i])
446 
447         # for frequency
448         elif change == self._lfreq:
449 
450             self._dropdown_description = "change order in listbox, currently ordered
                    by \"frequency\""
451             self.search_value.sort(second)
452 
453             # rearrange order of array
454             for i in range(len(self.search_value)):
455                 #save value
456                 temp = self.search_value[i][1]
457                 # search for corresponding label in label array
458                 for j in range(len(self.search_label)):
459                     if self.search_label[j][0] == self.search_value[i][0]:
460                         # save corresponding label
461                         self.items[i] = "%s  (%s)" % (self.search_label[j][1], self.
                                search_value[i][1])
462                         temp_search_label.append([i+1, self.search_label[j][1]])
463                 # adjust number in value array
464                 self.search_value[i][0] = i+1
465                 self.search_value[i][1] = temp
466 
467             # rearrange label array description
468             self.search_label = temp_search_label[:]
469 
470             # delete old items and write new items in listbox
471             self.listbox.delete(0, Tkinter.END)
472             for i in range(len(self.items)):
473                 self.listbox.insert(Tkinter.END, self.items[i])
474 
475     def pre_shutdown(self):
```

```
476         '''
477         This function calls a message box and make sure the user really wants to
                shutdown.
478         '''
479         # deactivate all buttons
480         try:
481             for i in range(len(self._all_windows)):
482                 self._all_windows[i].deactivate()
483         except Tkinter.TclError:
484             pass
485         # queston message box
486         status = tkMessageBox.askquestion("Close Window", "Do you really want to
                close this and all child windows ?")
487         # activate all buttons
488         try:
489             for i in range(len(self._all_windows)):
490                 self._all_windows[i].activate()
491         except Tkinter.TclError:
492             pass
493         if status == 'yes':
494             self.shutdown()
495 
496     def shutdown(self):
497         '''
498         This function closes all open child windows and itself
499         '''
500 
501         if self._file_fd != None:
502             if self._all_windows[0] == self:
503                 # only main windows closes file
504                 self._file_fd.close()
505 
506         # destroy all cildren windows
507         for i in range(len(self._windows)):
508             try:
509                 self._windows[i].shutdown()
510 
511             except Tkinter.TclError:
512                 pass
513 
514         # destroy myself
515         try:
516             self.destroy()
517         except Tkinter.TclError:
518             pass
519 
520     def _select_error(self):
521         '''
522         This function get the selected item from the listbox
523         '''
524         try:
525             # get index of chosen listbox item
```

```
526             firstIndex = self.listbox.curselection()[0]
527         except IndexError:
528             firstIndex = None
529 
530         if firstIndex != None:
531 
532             # convert index to int
533             firstIndex = int(firstIndex)
534 
535             # print data
536             title = "Diagram Error %s \"Frequency - Date\"" % self.search_label[
                    firstIndex][1]
537 
538             field = []
539             field_label = []
540             data_new = []
541             # work up the given data and prepare for display
542             for i in range(len(self._data)):
543                 if (int(self._data[i]['error.e_number']) == int(self.search_label[
                        firstIndex][1])):
544                     data_new.append(self._data[i]) # get new dataset (only
                            interesting data is taken)
545                     index = find_item(self._data[i]['messages.m_date'], field)
546                     if (None == index):
547                         field.append([self._data[i]['messages.m_date'], 1])
548                         field_label.append([self._data[i]['messages.m_date'], 1])
549                     else:
550                         count = field[index][1]
551                         count += 1
552                         field[index][1] = count
553                         field_label[index][1] = count
554 
555             field.sort()
556             field_label.sort()
557 
558             # print result table
559             h_line = "-----------------------------------"
560             v_line = "|"
561             header = "\nFrequency of Error \"%s\":\n" % self.search_label[firstIndex
                    ][1]
562 
563             # write in file
564             if self._file_fd != None:
565                 content =  "\n"+header
566                 content += "\n\n Nr.    | Date\t\t| Frequency\n\n"
567                 self._file_fd.write(content)
568 
569             if self._col == 1:
570                 col_obj = Colour()
571 
572                 header = col_obj.yellow(header)
573                 h_line = col_obj.yellow(h_line)
```

```
574                 v_line = col_obj.yellow(v_line)
575
576             print header
577             print h_line
578             print " Nr.   "+v_line+" Date\t\t"+v_line+" Frequency"
579             print h_line
580
581             for i in range(len(field)):
582                 print " %5d %s %s\t%s %s" % ((i+1), v_line, field[i][0], v_line,
                        field[i][1])
583
584                 # write in file
585                 if self._file_fd != None:
586                     content = " %5d | %s\t| %s\n" % ((i+1),  field[i][0], field[i
                            ][1])
587                     self._file_fd.write(content)
588
589             print h_line
590
591             for i in range(len(field_label)):
592                 temp = field_label[i][0]
593                 field_label[i][0] = field_label[i][1]
594                 field_label[i][1] = temp
595
596             for i in range(len(field)):
597                 field_label[i][0] = (i+1)
598                 field_label[i][1] = "%s" % field[i][0]
599                 field[i][0] = (i+1)
600
601             field_label, field = complete_days(field_label, field)
602
603             pic_obj = Picture(self._col, self._all_windows)
604
605             self._windows.append(pic_obj)
606
607             title = "Diagram \"Frequency - Date\" - Error: %s" % self.search_label[
                    firstIndex][1]
608             descr = title+" - Range: "+field_label[0][1]+"  -  "+field_label[len(
                    field_label)-1][1]#+" )"
609             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" )
610
611             pic_obj.mainloop()
612
613         else:
614             # disable all buttons within the windows
615             for i in range(len(self._all_windows)):
616                 self._all_windows[i].deactivate()
617
618             tkMessageBox.showerror("Error", "No item selected !")
```

```
619             # enable all buttons within the windows
620             for i in range(len(self._all_windows)):
621                 self._all_windows[i].activate()
622 
623     def _select_date(self):
624         '''
625         This functions take a date and generates a new graph
626         '''
627 
628         try:
629             firstIndex = self.listbox.curselection()[0]
630         except IndexError:
631             firstIndex = None
632 
633         if firstIndex != None:
634 
635             firstIndex = int(firstIndex)
636 
637             # print data
638             title = "Diagram \"Frequency - Time\" - Date %s" % self.search_label[
                        firstIndex][1]
639 
640             field = []
641             field_label = []
642             data_new = []
643             for i in range(len(self._data)):
644                 if self._data[i]['messages.m_date'] == self.search_label[firstIndex
                            ][1] and int(self._the_error) == int(self._data[i]['error.
                            e_number']):
645 
646                     data_new.append(self._data[i])
647                     hour = self._data[i]['messages.m_time'].split(":")
648 
649                     hour[0] = int(hour[0])# hour
650 
651                     index = find_item(hour[0], field)
652                     if (None == index):
653                         field.append([hour[0], 1])
654                         field_label.append([hour[0], 1])
655                     else:
656                         count = field[index][1]
657                         count += 1
658                         field[index][1] = count
659                         field_label[index][1] = count
660 
661             field.sort()
662             field_label.sort()
663 
664             # rearrange arrays for use within the picture and graph class
665             for i in range(len(field_label)):
666                 temp = field_label[i][0]
667                 field_label[i][0] = field_label[i][1]
```

```
668                 field_label[i][1] = temp
669
670             for i in range(len(field)):
671                 field_label[i][0] = (i+1)
672                 field_label[i][1] = "%s" % field[i][0]
673                 field[i][0] = (i+1)
674
675             field_label, field = complete_hours(field_label, field)
676             field_label, field = complete_ticks(field_label, field)
677
678             h_line = "------------------------------"
679             v_line = "|"
680             header = "\nFrequency on Date \"%s\":\n" % self.search_label[firstIndex
                    ][1]
681
682             # write in file
683             if self._file_fd != None:
684                 content = "\n"+header
685                 content += "\n\n Time of Day\t| Frequency\n\n"
686                 self._file_fd.write(content)
687
688             if self._col == 1:
689                 col_obj = Colour()
690                 print col_obj.yellow(header)
691                 h_line = col_obj.yellow(h_line)
692                 v_line = col_obj.yellow(v_line)
693
694             print h_line
695             print " Time of Day\t"+v_line+" Frequency"
696             print h_line
697
698             for i in range(len(field)):
699                 print " %2s h - %2s h\t%s %s" % (i, i+1, v_line, field[i][1])
700
701                 if self._file_fd != None:
702                     content = " %2s h - %2s h\t| %s\n" % (i , i+1, field[i][1])
703                     self._file_fd.write(content)
704
705             print h_line
706
707             pic_obj = Picture(self._col, self._all_windows)
708             self._windows.append(pic_obj)
709
710             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  )
711             pic_obj.mainloop()
712
713         else:
714
715             for i in range(len(self._all_windows)):
716                 self._all_windows[i].deactivate()
```

```
717             tkMessageBox.showerror("Error", "No item selected !")
718             for i in range(len(self._all_windows)):
719                 self._all_windows[i].activate()
720 
721     def _show_description(self, event):
722         '''
723         This function displays the description for the listbox in the status bar.
724         '''
725         if self._select_type == 'error':
726             self.status.config(text = "listbox: error number (frequency) -> select
                    error to zoom", anchor = "w")
727         if self._select_type == 'date':
728             self.status.config(text = "listbox: date (frequency) for the choosen
                    error -> select date to zoom", anchor = "w")
729         self.status.update_idletasks()
730 
731     def _show_dropdown_description(self, event):
732         '''
733         This function shows a short description for the dropdown menu in the status
                bar.
734         '''
735         self.status.config(text = self._dropdown_description, anchor = "w")
736         self.status.update_idletasks()
737 
738     def _show_description_two(self, event, msg):
739         '''
740         This function shows a short description (msg) in the status bar.
741         '''
742         self.status.config(text=msg, anchor = "w")
743         self.status.update_idletasks()
```

D.3.3 Module `gui_utils.py`

LISTING D.11: Module `gui_utils.py`

```
1  #!/usr/bin/env python
2  
3  '''
4  This module provides small utility methods that are used by the gui_classes.py and
       gui.py.
5  
6  Reading University
7  MSc in Network Centered Computing
8  a.weise - a.weise@reading.ac.uk - December 2005
9  '''
10 import os, time, ConfigParser, string
11 import gui_classes
12 import calendar
13 
```

```
14 def LoadConfig(file_name, config={}):
15     """
16     returns a dictionary with key's of the form
17     <section>.<option> and the values
18 
19     source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/65334
20     """
21     config = config.copy()
22     cp = ConfigParser.ConfigParser()
23     cp.read(file_name)
24     for sec in cp.sections():
25         name = string.lower(sec)
26         for opt in cp.options(sec):
27             config[name + "." + string.lower(opt)] = string.strip(cp.get(sec, opt))
28     return config
29 
30 def usage_exit(progname, msg = None, color = 0):
31     '''
32     This function gives usage help and exits script.
33     '''
34     if msg:
35         if 1 == color and msg != None:
36             color_obj = gui_classes.Colour()
37             print color_obj.red(msg)
38         else:
39             print msg
40         print # lf cr
41 
42     text = "usage: python %s -c config_file [optional commands] \n\n" % progname
43     if 1 == color:
44         print color_obj.red(text)
45     else:
46         print text
47     os._exit(-1)
48 
49 def check_time(timus):
50     '''
51     This functions checks if a given time with the format hour:minute:second
           (12:45:46) is valid.
52     '''
53     timus = timus.split(':')
54     if int(timus[0]) < 24 and int(timus[1]) < 60 and int(timus[2]) < 60:
55         return 0
56     else:
57         return -1
58 
59 def check_date(datus):
60     '''
61     This function checks if a given date is valid.
62     '''
63     datus = datus.split(".")
64     tup1 = (int(datus[2]), int(datus[1]), int(datus[0]), 0, 0, 0, 0, 0, 0)
```

```
65      try:
66          date = time.mktime (tup1)
67          tup2 = time.localtime (date)
68          if tup1[:2] != tup2[:2]:
69              return -1
70          else:
71              return 0
72      except OverflowError:
73          return -1
74      except ValueError:
75          return -1
76
77  def convert_date(datus):
78      '''
79      This function converts a date like 01.10.2005 into database conform date like
            2005-10-01.
80      '''
81      datus = datus.split(".")
82      return "%s-%s-%s" % (datus[2], datus[1], datus[0])
83
84  def convert_date_readable(datus):
85      '''
86      This function converts a date like 2005-10-10 into are readable format
            01.10.2005.
87      '''
88      datus = datus.split("-")
89      return "%s.%s.%s" % (datus[2], datus[1], datus[0])
90
91  def check_ip(ip):
92      '''
93      This function checks if a given IP is valid.
94      '''
95      try:
96          ip = ip.split(".")
97      except AttributeError:
98          return -1
99
100     for i in range(len(ip)):
101         check =  ip[i].find("0", 0, 1)
102         if -1 != check and 1 < len(ip[i]):
103             return -1
104         try:
105             ip[i] = int(ip[i])
106         except ValueError:
107             return -1
108         if ip[i] >= 0 and ip[i] <= 255:
109             pass
110         else:
111             return -1
112
113     return 0
114
```

```
115 def find_item(search, listus):
116     '''
117     This function find an item within a list (2 dimensional)
118     '''
119     for i in range(len(listus)):
120         if 1 == len(listus[i]):
121             if listus[i] == search:
122                 return i
123         elif 2 == len(listus[i]):
124             if listus[i][0] == search:
125                 return i
126         elif 3 == len(listus[i]):
127             if listus[i][0][0] == search:
128                 return i
129     return None
130 
131 def help_context(color):
132     '''
133     This function provides the help context.
134     '''
135 
136     color_obj = gui_classes.Colour()
137     msg = ''
138     if color == 1:
139         msg += color_obj.green("\n-----------   Help   ----------\n\n\n")
140     else:
141         msg += "\n-----------   Help   ----------\n\n\n"
142 
143     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"
144     if color == 1:
145         msg += color_obj.purple(note)
146     else:
147         msg += note
148 
149     msg += "-c or --config\t\t\t-> defines config file, if no config file given,
         default values are used\n"\
150         "-v or --verbose\t\t\t-> activates printing of messages [debug option]\n"\
151         "-h or --help\t\t\t-> print this help\n"\
152         "-g or --graph\t\t\t-> show output additionally as a diagram\n"\
153         "--nocolor\t\t\t-> no colored console output\n"\
154         "--file <string>\t\t\t-> dump output into a file (file name has to be given)\
             n"
155 
156     if color == 1:
157         msg += color_obj.green("\n-----  database commands  -----\n\n")
158     else:
159         msg += "\n-----  database commands  -----\n\n"
160     msg += "--sql_host\t\t\t-> show all hosts\n"\
161         "--sql_project\t\t\t-> show all projects\n"\
```

```
162         "--sql_error\t\t\t-> show errors (additional parameters possible)\n"\
163         "--sql_error_freq\t\t-> show only frequency of errors (additional parameters
                possible)\n"
164     if color == 1:
165         msg += color_obj.green("\n-----  additional parameters  -----\n")
166     else:
167         msg += "\n-----  additional parameters  -----\n"
168     msg += "\n--start_date <date>\t\t-> start date (e.g. 23.12.2005)\n"\
169         "--end_date <date>\t\t-> end date (e.g. 23.01.2006)\n"\
170         "--start_time <time>\t\t-> start time (e.g. 23:12:19)\n"\
171         "--end_time <time>\t\t-> end time (e.g. 23:12:59)\n"\
172         "--ip <ip>\t\t\t-> host IP (e.g. 127.0.0.1)\n"\
173         "--project <string>\t\t-> specify a certain project\n"\
174         "--error <int,int...>\t\t-> specify a certain error (comma seperated list)\n"
175     if color == 1:
176         msg += color_obj.green("\n-----  examples  -----\n\n")
177         msg += color_obj.blue("python gui.py -c config_gui.ini --sql_project\n")
178         msg += color_obj.yellow("\t-> show all projects\n\n")
179
180         msg += color_obj.blue("python gui.py -c config_gui.ini --sql_host\n")
181         msg += color_obj.yellow("\t-> show all host and the corresponding project\n\n
                ")
182
183         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")
184         msg += color_obj.yellow("\t-> show all errors of localhost between 01.01.2005
                 and 01.03.2005\n\n")
185
186         msg += color_obj.blue("python gui.py -c config_gui.ini --sql_error --
                start_date 01.01.2005  --project mySRBproject\n")
187         msg += color_obj.yellow("\t-> show all errors between 01.01.2005 and now for
                the project \"mySRBproject\"\n\n")
188
189         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")
190         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")
191
192         msg += color_obj.blue("python gui.py -c config_gui.ini --sql_error_freq --
                error -1023 --ip 127.0.0.1 -g \n")
193         msg += color_obj.yellow("\t-> show error frequency for the error -1023 from
                host 127.0.0.1 and display diagram\n\n")
194
195
196     else:
197         msg += "\n-----  examples  -----\n\n"
198         msg += "python gui.py -c config_gui.ini --sql_project\n"
199         msg += "\t-> show all projects\n\n"
200
201         msg += "python gui.py -c config_gui.ini --sql_host\n"
202         msg += "\t-> show all host and the corresponding project\n\n"
```

```
203
204         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"
205
206         msg += "python gui.py -c config_gui.ini --sql_error --start_date 01.01.2005
                --project mySRBproject\n"
207         msg += "\t-> show all errors between 01.01.2005 and now for the project \"
                mySRBproject\"\n\n"
208
209         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"
210         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"
211
212         msg += "python gui.py -c config_gui.ini --sql_error_freq --error -1023 --ip
                127.0.0.1 -g \n"
213         msg += "\t-> show error frequency for the error -1023 from host 127.0.0.1 and
                 display diagram\n\n"
214
215     msg += "\n"
216
217     return msg
218
219 def complete_hours(label, field):
220     '''
221     This function completes the missing hours within an array
222     '''
223     new_hours = []
224     new_values = []
225
226     count = 0
227     for i in range(len(label)):
228         temp = "%d" % count
229         while(count < 24):
230             if temp == label[i][1]:
231                 break
232             new_hours.append([count, temp])
233             new_values.append([count, 0])
234             count += 1
235             temp = "%d" % count
236
237         new_hours.append([count, label[i][1]])
238         new_values.append([count, field[i][1]])
239
240         count += 1
241
242     # last hours
243     while(count <= 23):
244         temp = "%d" % count
245         new_hours.append([count, temp])
```

```
246         new_values.append([count, 0])
247         count += 1
248 
249     return new_hours, new_values
250 
251 def complete_days(days, value_field):
252     '''
253     This function completes the missing dates within an array.
254     '''
255     if len(days) == 1:
256         # if only one day in the field
257         return days, value_field
258 
259     new_days = [] # new array with the completed days
260     new_values = []
261 
262     for i in range(len(days)):
263         day1 = days[i][1].split("-")
264         day2 = days[i+1][1].split("-")
265 
266         for x in range(len(day1)):
267             day1[x] = int(day1[x])
268             day2[x] = int(day2[x])
269 
270         if day1[0] == day2[0] and day1[1] == day2[1] and (day1[2]+1) == day2[2]:
271             # save day1 in new array
272             temp1 = "%d" % day1[2]
273             temp2 = "%d" % day1[1]
274             date = "%d-" % day1[0]
275             if len(temp2) == 1:
276                 date += "0%d-" % day1[1]
277             else:
278                 date += "%d-" % day1[1]
279 
280             if len(temp1) == 1:
281                 date += "0%d" % (day1[2])
282             else:
283                 date += "%d" % (day1[2])
284 
285             if len(new_days) == 0:
286                 number = 1
287             else:
288                 number = int(new_days[len(new_days)-1][0])+1
289             new_days.append([number, date])
290             new_values.append([number, value_field[i][1]])
291         else:
292             # not the following day
293             if day1[0] == day2[0] and day1[1] == day2[1]:
294                 #year and month the same
295                 new_days, new_values = complete_d(new_days, day1, day2, new_values,
                        value_field[i][1])
296 
```

```
297             elif day1[0] == day2[0]:
298                 # year the same
299                 new_days, new_values = complete_m(new_days, day1, day2, new_values,
                        value_field[i][1])
300
301             else:
302                 # year change
303                 new_days, new_values = complete_y(new_days, day1, day2, new_values,
                        value_field[i][1])
304
305         if (i+2) == len(days):
306             break
307
308     # add last date
309     temp1 = "%d" % day2[2]
310     temp2 = "%d" % day2[1]
311     date = "%d-" % day2[0]
312
313     if len(temp2) == 1:
314         date += "0%d-" % day2[1]
315     else:
316         date += "%d-" % day2[1]
317
318     if len(temp1) == 1:
319         date += "0%d" % day2[2]
320     else:
321         date += "%d" % day2[2]
322
323     if len(new_days) == 0:
324         number = 1
325     else:
326         number = int(new_days[len(new_days)-1][0])+1
327     new_days.append([number, date])
328     new_values.append([number, value_field[i+1][1]])
329
330     return new_days, new_values
331
332 def complete_d(new_array, start_date, end_date, new_field, value):
333     '''
334     Add missing dates within a month
335     '''
336
337     month = calendar.monthcalendar(start_date[0], start_date[1])
338     # run through matrix and save all dates between day1 and day2 in new_days array
339     found = 0
340     terminate = 0
341     for x in range(len(month)):
342         if terminate != 0:
343             break
344         for y in range(len(month[x])):
345             # go to current day1
346             if terminate != 0:
```

```
347                 break
348             if start_date[2] == month[x][y] and found == 0:
349                 #save date1 in new_days
350                 temp1 = "%d" % start_date[2]
351                 temp2 = "%d" % start_date[1]
352 
353                 date = "%d-" % start_date[0]
354 
355                 if len(temp2) == 1:
356                     date += "0%d-" % start_date[1]
357                 else:
358                     date += "%d-" % start_date[1]
359 
360                 if len(temp1) == 1:
361                     date += "0%d" % start_date[2]
362                 else:
363                     date += "%d" % start_date[2]
364 
365                 if (0 < len(new_array)):
366                     number = int(new_array[len(new_array)-1][0])+1
367                 else:
368                     number = 0
369                 new_array.append([number, date])
370                 new_field.append([number, value])
371 
372                 found = 1
373             elif found == 1:
374                 # add new dates
375                 if end_date[2] == month[x][y]:
376                     terminate = 1
377                 else:
378                     # save dates
379                     temp1 = "%d" %  month[x][y]
380                     temp2 = "%d" % end_date[1]
381 
382                     date = "%d-" % end_date[0]
383 
384                     if len(temp2) == 1:
385                         date += "0%d-" % end_date[1]
386                     else:
387                         date += "%d-" % end_date[1]
388 
389                     if len(temp1) == 1:
390                         date += "0%d" %  month[x][y]
391                     else:
392                         date += "%d" %  month[x][y]
393 
394                     # get next entry number in arrays
395                     if (0 < len(new_array)):
396                         number = int(new_array[len(new_array)-1][0])+1
397                     else:
398                         number = 0
```

```
399                     new_array.append([number, date])
400                     new_field.append([number, 0])
401             else:
402                 pass
403
404     return new_array, new_field
405
406 def complete_m(new_array, start_date, end_date, new_field, value):
407     '''
408     This function adds missing dates within a year.
409     '''
410     start_month = start_date[1]
411     end_month = end_date[1]
412
413     #current month
414     month = calendar.monthrange(start_date[0], start_date[1])
415     temp_date2 = [start_date[0], start_date[1], month[1]]
416
417     new_array, new_field = complete_d(new_array, start_date, temp_date2, new_field,
            value)
418
419     start_month += 1
420
421     # month in between
422     while(start_month < end_month):
423
424         month = calendar.monthrange(start_date[0], start_month)
425         temp_date2 = [start_date[0], start_month, month[1]]
426         temp_date1 = [start_date[0], start_month, 1]
427
428         new_array, new_field = complete_d(new_array, temp_date1, temp_date2,
                new_field, 0)
429
430         start_month += 1
431
432     # last month
433     temp_date1 = [start_date[0], start_month, 1]
434
435     new_array, new_field = complete_d(new_array, temp_date1, end_date, new_field, 0)
436
437     return new_array, new_field
438
439 def complete_y(new_array, start_date, end_date, new_field, value):
440     '''
441     This function adds missing dates within many years
442     '''
443     start_year = start_date[0]
444     end_year = end_date[0]
445
446     # current year
447     temp_date2 = [start_date[0], 12, 31]
```

```
448     new_array, new_field = complete_m(new_array, start_date, temp_date2, new_field,
            value)
449
450     start_year += 1
451
452     # years in between
453     while(start_year < end_year):
454
455         temp_date1 = [start_year, 1, 1]
456         temp_date2 = [start_year, 12, 31]
457         new_array, new_field = complete_m(new_array, temp_date1, temp_date2,
                new_field, 0)
458         start_year += 1
459
460     # last year
461     temp_date1 = [start_year, 1, 1]
462     new_array, new_field = complete_m(new_array, temp_date1, end_date, new_field, 0)
463
464     return new_array, new_field
465
466 def complete_ticks(label, values):
467     '''
468     This function adds bins, so that the dot in the time diagram are between two
            hours.
469     '''
470     new_label = []
471     new_values = []
472
473     count = 0
474     for i in range(2*len(label)):
475         if (i%2) != 0:
476             new_values.append([i, values[count][1]])
477             new_label.append([i, ""])
478             count += 1
479         else:
480             new_label.append([i, label[count][1]])
481
482     return new_label, new_values
483
484 def second(t1, t2 ):
485     '''
486     This function works with sort and the field gets sorted descending, but the
            second value within the array is taking into account !!!
487     '''
488     # sort descending
489     return t2[1] - t1[1]
490
491 def second_string_to_int(t1, t2):
492     '''
493     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.)
```

```
494     '''
495     # sort ascending
496     return int(t1[1]) - int(t2[1])
497
498 def second_string_only(t1, t2):
499     '''
500     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.)
501     '''
502     # sort ascending
503     return cmp(t1[1], t2[1])
```

D.4 Remote Controller

LISTING D.12: Module `admin_server.py`

```
#!/usr/bin/env python
2
'''
This module can be used to administer the server (daemon).

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


# config parsing
12 import ConfigParser, string

#misc
import os, getopt, sys, re

17 # connection issues
from M2Crypto.m2xmlrpclib import Server, SSL_Transport
from M2Crypto import SSL

def LoadConfig(file, config={}):
22     """
    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
27     """
    config = config.copy()
    cp = ConfigParser.ConfigParser()
    cp.read(file)
    for sec in cp.sections():
```

```
32          name = string.lower(sec)
            for opt in cp.options(sec):
                config[name + "." + string.lower(opt)] = string.strip(cp.get(sec, opt))
        return config


37
class Colour:
        '''
        This class uses the ANSI escape sequences to color the output !
        '''
42      color = {"reset":"\x1b[0m",
                 "bold":"\x1b[01m",
                 "teal":"\x1b[36;06m",
                 "turquoise":"\x1b[36;01m",
                 "fuscia":"\x1b[35;01m",
47               "purple":"\x1b[35;06m",
                 "blue":"\x1b[34;01m",
                 "darkblue":"\x1b[34;06m",
                 "green":"\x1b[32;01m",
                 "darkgreen":"\x1b[32;06m",
52               "yellow":"\x1b[33;01m",
                 "brown":"\x1b[33;06m",
                 "red":"\x1b[31;01m",
                 "darkred":"\x1b[31;06m"}

57      def __init__(self):
            '''
            Constructor
            '''
            pass
62
        def green(self, text):
            '''
            dye green
            '''
67          return self.color['green']+text+self.color['reset']

        def red(self, text):
            '''
            dye red
72          '''
            return self.color['red']+text+self.color['reset']

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

        def teal(self, text):
82          '''
            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):
```

```
        '''
137     dye brown
        '''
        return self.color['brown']+text+self.color['reset']


142 class Admin:
    '''
    This is manager class for the Remote Controller application.
    '''

147     def __init__(self, config):
        '''
        Constructor
        '''
        workingpath = os.getcwd()
152
        # 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("/")
157     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)):
162             # first character "/"
                pass
            else:
                self.__client_certificate_path = workingpath+"/"+self.
                    __client_certificate_path

167     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
172     else:
            self.__client_ca = self.__client_ca.strip()
            if (-1 != self.__client_ca_path.find("/", 0, 1)):
                # first character "/"
                pass
177         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
182         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)
187         os._exit(-1)

    def connect_to_server(self, server, port):
        '''
        This function establishs the connection to the server.
192     '''
        serverus = server

        ctx = self.create_ctx()
        # connect to server via SSL using the created context
197     urladdress = "https://%s:%d" % (serverus, port)
        server = Server(urladdress, SSL_Transport(ctx))
        # return server object
        return server

202 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
207     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
212

# ------------------ additional functions --------------------

def usage_exit(progname, msg = None, color = 1):
    '''
217 This function gives usage help and exits the module.
    '''
    if msg:
        if 1 == color and msg != None:
            color_obj = Colour()
222         print color_obj.red(msg)
        else:
            print msg
        print # lf cr
```

```
227     text = "usage: python %s -c config_file [optional commands] \n\n" % progname
        if 1 == color:
            print color_obj.red(text)
        else:
            print text
232     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.
237     '''

        try:
            ip = ip.split(".") # split in 4 number
        except AttributeError:
242         return -1

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

257     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".
262     '''
        for i in range(len(listus)):
            if 1 == len(listus[i]):
                if listus[i] == search:
                    return i
267         elif 2 == len(listus[i]):
                if listus[i][0] == search:
                    return i
            elif 3 == len(listus[i]):
                if listus[i][0][0] == search:
272                 return i
        return None

    def help_context(color):
        '''
```

```
277     This function provides the help context.
        '''

        color_obj = Colour()
        msg = ''
282     if color == 1:
            msg += color_obj.green("\n-----------   Help   ----------\n\n\n")
        else:
            msg += "\n-----------   Help   ----------\n\n\n"

287     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
292
        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:
297         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"\
302         "--enable_rpc\t\t\t-> enable rpc calls (on server side)\n"\
            "--shutdown\t\t\t-> shutdown server\n"\
            "--interval_status\t\t-> change parsing interval of 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"\
307         "--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"
312     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"\
317         "--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
    stop = 0

    # 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'):
                print "no color"
                col = 0
            if opt in ('-h','--help'):
                stop = 1
            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)
412 except ValueError, e:
        e = "%s" % e
        usage_exit(sys.argv[0], e, col)

    if stop == 1:
417     msg = help_context(col)
        usage_exit(sys.argv[0], msg, col)

    # load config file or default values
    if (configfile != ""):
422     # check if file exists
        if(1 == os.path.exists(configfile)):
            config = LoadConfig(configfile)
        else:
            # if file NOT exists terminate program
427         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)
432

    gui = Admin(config)

    if col == 1:
437     col_obj = Colour()

    #-------------------- SERVER COMMANDS --------------------#

    if (1 == rpc_status):
442     # 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)
447         print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_status()
452             if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
457             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:
```

```
512                 text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
517                 answer = serv_object.enable_rpc_calls()
                    if col == 1:
                        answer = col_obj.green(answer)
                    print "\nserver -> %s" % answer
                except AssertionError:
522                 text = "Server is down !"
                    if col == 1:
                        print col_obj.red(text)
                    else:
                        print text
527             except:
                    text = "Could not connect to server \"%s:%d\"." % (ip, port)
                    if col == 1:
                        print col_obj.red(text)
                    else:
532                     print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
537         else:
                print text

    elif (1 == shutdown):
        # shutdown the server
542     if (None != ip and None != port):
            text = "Shutdown server \"%s:%d\"." % (ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
547         print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.stop_server()
                if col == 1:
552                 answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
557                 print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
562             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, 3)
                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
712     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:
```

```
762                     print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
767             else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
772             print col_obj.red(text)
            else:
                print text

    elif (None !=  delete_keyword):
777     # 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)
782         print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_update_keyword_file(delete_keyword, 0)
787             if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
792             if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
797             text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
802     else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
807             print text

    elif (1 ==  interval_status):
        # get current parsing interval time
        if (None != ip and None != port):
```

```
812             text = "Get parsing interval time (in minutes) from server \"%s:%s\"." %
                    (ip, port)
                if col == 1:
                    text = col_obj.yellow(text)
                print
                print text
817             serv_object = gui.connect_to_server(ip, port)
                try:
                    answer = serv_object.rpc_interval_status()
                    if answer != -2:
                        answer = "every %d minutes" % answer
822                 else:
                        answer = "RPC disabled"
                    if col == 1:
                        answer = col_obj.green(answer)

827                 print "\nserver -> %s" % answer
                except AssertionError:
                    text = "Server is down !"
                    if col == 1:
                        print col_obj.red(text)
832                 else:
                        print text
                except:
                    text = "Could not connect to server \"%s:%d\"." % (ip, port)
                    if col == 1:
837                     print col_obj.red(text)
                    else:
                        print text
            else:
                text = "\nNo IP or port given !\n"
842             if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
847         text = "No command given !\nUse option -h or --help to display the help."
            usage_exit(sys.argv[0], text, col)


if __name__ == '__main__':
852
    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:
```

```
43          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
48          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
53          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)
58
    def get_keywords(filus):
        '''
        This function extracts keyword from a give file!
        '''
63      keys = []

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

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

73      file_fd.close()

        content = remove_item(content, "#") # remove comments
        content = remove_item(content, "\n")# remove linebreaks

78      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])):
83              keys.append(content[i][a])

        for i in range(len(keys)):
            keys[i] = keys[i].strip() # remove whitespace
            keys[i] = keys[i].split(":")
88
        return keys

    def remove_item(listus, item):
        '''
93      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
```

```
193     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)):
198         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")
203     if keyword != None:
            keyword = keyword.strip()
        if keyword_path != None:
            keyword_path = keyword_path.rstrip("/")
        if(keyword_path == '' or keyword_path == None):
208         keyword_path = workingpath
        else:
            if (-1 != keyword_path.find("/", 0, 1)):
                # first character "/"
                pass
213         else:
                keyword_path = workingpath+"/"+keyword_path

        keyword_list = get_keywords(keyword_path+"/"+keyword)

218     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())
223
        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()
228     os.chdir(path_srb_gz)
        if (0 < len(gz_list)):
            try:
                for x in range(len(gz_list)):
                    print "\n"
233                 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])
238                 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

The enclosed Compact Disc (CD) contains all the software and documentation related to the project. Table E.1 presents a general overview of the content. **Not** every single file is listed. A few HTML pages were developed to lead the user through all the files. By executing `index.html` with a browser (Opera[1] or Firefox[2] are recommended) the self-explanatory CD menu is started. From there all files and documentation can be easily accessed.

TABLE E.1: CD Content Overview

Folder or File	Explanation
	top level
`download`	contains the software
`doxygen`	contains the complete Doxygen documentation
`images`	contains pictures which are used for the CD menu
`index.html`	start file for the CD menu
`doxygen.html`	file which is part of the CD menu, it gives access to the complete Doxygen documentation
`download.html`	file which is part of the CD, it gives access to the complete software and further documentation
`readme.html`	file which is part of the CD, it gives a short introduction how to setup the monitoring system

Continued on next page

[1] Opera - http://www.opera.com
[2] Firefox - http://www.mozilla.com/firefox

Table E.1 CD Content Overview - *continued from previous page*

Folder or File	Explanation
`style.css`	style file for the CD menu
second level `download`	
`additional_software`	contains additional software package, required by the monitoring system
`monitoring_tools`	contains the monitoring system itself
`dissertation.pdf`	complete dissertation as file
second level `doxygen`	
`client`	contains the complete HTML Doxygen documentation of the Client
`server`	contains the complete HTML Doxygen documentation of the Server
`virtualiser`	contains the complete HTML Doxygen documentation of the Virtualiser
`remote_controller`	contains the complete HTML Doxygen documentation of the Remote Controller
`gz_parser`	contains the complete HTML Doxygen documentation of the GZ Parser
second level `additional_software`	
`m2crypto-0.13.zip`	M2Crypto package
`sqlite-2.8.16.tar.gz`	SQL database
`pysqlite-1.0.1.tar.gz`	Python SQL database interface
second level `monitoring_tools`	
`GZ_Parser.zip`	complete GZ_Parser software
`Remote_Controller.zip`	complete Remote_Controller software
`Server.zip`	complete Server software
`Virtualiser.zip`	complete Virtualiser software
`readme.txt`	monitoring system setup guide
`test_modules.py`	module to test the correct package installation

CD directory structure (overview):

```
.
|--- download
|--- download.html
| |--- additional_software
| | |--- m2crypto-0.13.zip
| | |--- pysqlite-1.0.1.tar.gz
| | |--- sqlite-2.8.16.tar.gz
| |--- dissertation.pdf
| |--- monitoring_tools
| | |--- Client.zip
| | |--- GZ_Parser.zip
| | |--- Remote_Controller.zip
| | |--- Server.zip
| | |--- Virtualiser.zip
| | |--- readme.txt
| | |--- test_modules.py
|--- doxy.html
|--- doxygen
| |--- client
| | |--- *
| |--- gz_parser
| | |--- *
| |--- remote_controller
| | |--- *
| |--- server
| | |--- *
| |--- virtualiser
| | |--- *
|--- images
| |--- Back.png
| |--- BackBottom.png
| |--- BackTop.png
| |--- Puntik.png
| |--- doxygen.png
|--- index.html
|--- readme.html
|--- styles.css
```

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.* [54] *It was held February 2nd and 3rd at SDSC in San Diego.* [54]

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.

* 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.

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.

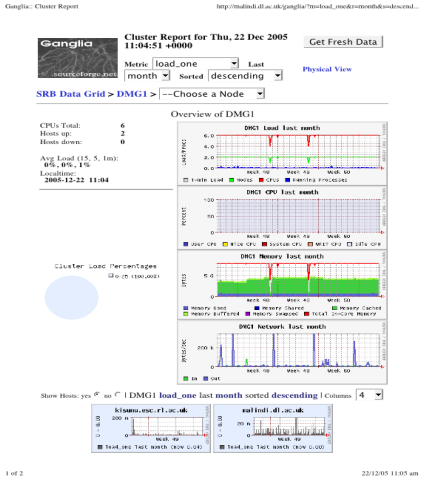

Figure 1: Ganglia web page displaying usage for a test SRB server.

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.

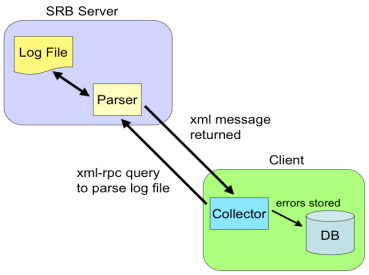

Figure 2: A simple schematic showing the log file parser system.

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.

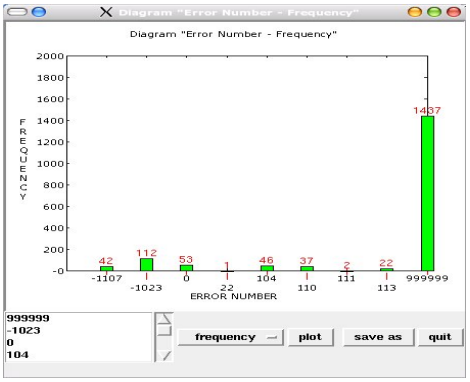

Figure 3: Screenshot of the error message numbers extracted from an SRB log file.

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.

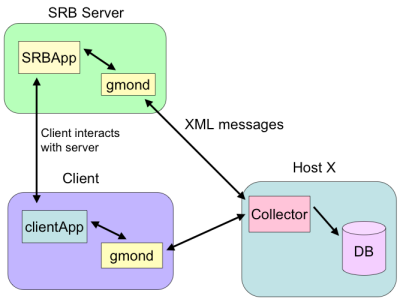

Figure 4: Schematic of the framework for performance measurements.

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.

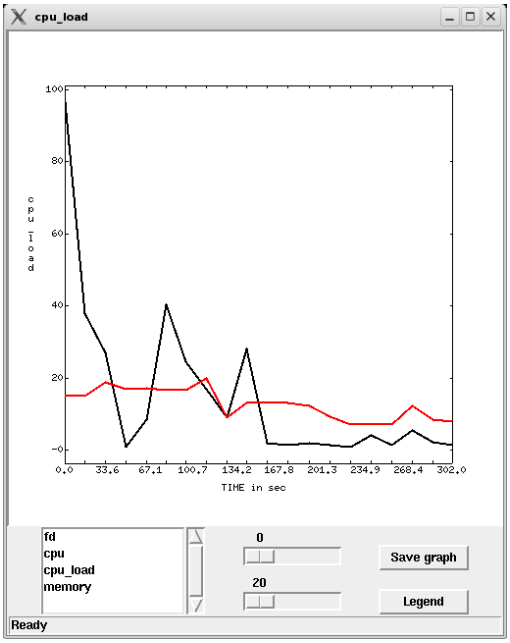

Figure 3: Graph of cpu-load produced by the Displayer application.

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, 13 March 2006

Andrea Weise