University College London
Department of Computer Science
M.Sc Data Communications, Network and Distributed Systems

Extensible Secure Event and Report Toolkit

Group Report

Group Members
Shaohua Yan
Yang Wang
Xiaoyue She
Ping Liang

Supervisor
Dr. Saleem Bhatti

30 Aug 2005

Abstract

This report mainly describes the design and implementation of a toolkit based on the system architecture developed by last year's project group. Last year's system architecture was proved to be suitable to resolve the problem of crossing heterogeneous platforms among different IXPs. However, it is still not easy to achieve this in terms of software implementation such as retrieving data from different platforms, processing and parsing different kind of data, and designing general and flexible data transmission protocol. In order to resolve above problems, a compiler is created this year, which can generate common code for system components. In addition, the provided API functions for data processing can be invoked to meet different requirements from clients. Therefore, this combination of compiler and API provides a very easy and convenient way to create different network monitoring toolkits. Furthermore, this project uses Python and XML for program implementation and data encapsulation so that the toolkit is easy to be interacted, to cross platform and to process different type of data. Moreover, this project also exploits many significant aspects of project management such as iterative and incremental development method, progress monitoring and risk management. Finally, this report discusses the future work that lights the road for further development of this project.

Acknowledgement

We group members appreciate the dedicated work of our supervisor Dr. Saleem Bhatti sincerely. His invaluable suggestions, feedback, resources and ideas during the whole project work contribute significantly to the success of this project.

In addition, we would like to thank Felipe Huici for his kind assistance during all stages of this project.

Table of Contents

Abstract 2
Acknowledgement 3
Table of Contents 4
List of Figures 8
List of Tables 8
Chapter 1 Introduction 9
1.1 Purpose 9
1.2 Problem Space and Work Motivation 9
1.3 Objectives and Scope 10
Chapter 2 Background 11
2.1 Internet Exchange Point 11
2.2 Secure Socket Layer 12
2.3 Simple Network Management Protocol 12
2.4 ASN.1 13
2.5 XML 14
Chapter 3 Requirement 15
3.1 Functional Requirement 15
3.1.1 Represent similar data in a common format 15
3.1.2 Create a compiler to generate common classes and code 15
3.1.3 Provide a common APIs for data analysis and data representation 16
3.2 Non-Function Requirement 16
3.2.1 Implement the system using Python 16
3.2.2 Easy to use, deploy and extend 16
3.2.3 Secure access and communications 17
3.2.4 Release the software as open source 17
Chapter 4 System Architecture 18
4.1 System Architecture 18
4.2 Advantages 21
Chapter 5 Design and Implementation 22
5.1 Data Representation 22
5.2 Validation 24
5.3 Data Types 25

5.3.1 Basic data type 25
5.3.2 User-defined type 26
5.4 Binary Encoding 27
5.5 Glue 31
5.5.1 G-RS 31
5.5.2 G-RI 32
5.6 PoD 33
5.6.1 Event Infrastructure 33
5.6.2 Data transmission model 34
5.6.3 Multiple clients' connections 34
5.6.4 Implementation 36
5.6.5 Protocols 38
5.6.6 Configuration file 39
5.7 Security 39
5.7.1 Implementation 40
5.7.2 Certificate management in OpenSSL 41
5.8 PoDRegistry 42
5.8.1 Implementation 43
5.8.2 Protocols 44
5.9 Compiler 45
5.9.1 Architecture 45
5.9.2 ExSERT Schema 46
5.9.3 Implementation 46
5.9.4 Directory structure 47
5.10 API 47
5.10.1 ThresholdRTT 48
5.10.2 LinkStatus 50
5.10.3 LinkUtilization 51
5.10.4 StatisticRTT 52
5.11 Database 53
5.11.1 Table design 54
Chapter 6 Testing 56
6.1 Test Strategy 56
6.2 Test Case Summary 58

6.2.1 Resources involved 58
6.2.2 Single model/function test 58
6.2.3 Whole model integration test 58
6.2.4 Exceptional case test 59
6.2.5 Stress and performance test 59
Chapter 7 Project Management 60
7.1 Project Scope 60
7.2 Team Organization and Communication Plan 62
7.3 Project Schedule and Progress 63
7.4 Risk Management 66
7.5 Status Control and Monitoring 67
7.5.1 Version control 67
7.5.2 Defect code root cause tracking 68
7.5.3 Re-factoring code and program naming standard 68
7.5.4 Test process standardization 69
7.6 Report 69
Chapter 8 Development Tools & Technologies 71
8.1 Programming Languages 71
8.2 Tools 72
8.2.1 Fping 72
8.2.2 CVS 72
8.2.3 Pydoc 73
8.2.4 MySQL 73
8.2.5 OpenSSL 74
8.2.6 Twisted 74
8.2.7 Rational Rose 75
8.2.8 Visio 75
Chapter 9 Future work 77
9.1 Network Error Prediction 77
9.2 Future Development of the Compiler 77
9.3 Super PoD 78
9.4 Validation between DTD and XML schema 78
9.5 Separation of Glue and PoD 78
Chapter 10 Conclusion 80

References 83
Appendix A: Class Diagram 85
1 Basic Data Types 85
2 LinkMeasure RMF 86
3 Compiler 87
4 PoDRegistry 87
5 Data Analysing Function 88
6 Example VFClient 89
Appendix B: Sequence Diagram 90

List of Figures

Figure 2.1: SNMP System Architecture 13
Figure 4.1: System Architecture 18
Figure 5.1: Data Representation and Mapping 23
Figure 5.2: Class Diagram for the PoD 36
Figure 5.3: Class Diagram for the PoD and Glue 38
Figure 5.4: Mutual Authentication Architecture 39
Figure 5.5: Compiler Architecture 45
Figure 5.6: Directory Structure of Generated Files 47
Figure 5.7: Package Structure 48
Figure 6.1: Client Output 59
Figure 7.1: Phase 1 and Part of Phase 2 63
Figure 7.2: Phase 2 and Part of Phase 3 64
Figure 7.3: Phase 3 64
Figure 7.4: Part of Phase 4 64
Figure 7.5: Part of Phase 4 and Phase 5 65

List of Tables

Table 5.1: Basic Data Types 25
Table 5.2: Data Type 30
Table 5.3: Binary Encoding 30
Table 5.4: TLV 30
Table 5.5: Data Packet Architecture 30
Table 5.6: RP Protocol 44
Table 5.7: RC Protocol 44
Table 7.1: Risks Management 66

Chapter 1 Introduction

1.1 Purpose

This project is to design and implement an Extensible Secure Event and Report Toolkit (ExSERT) based on the system architecture developed by the last year's project group. Unlike the previous project, there is no static Remote Monitoring Function (RMF) that has been implemented. Instead, the new ExSERT provides a way that the developer can easily create a new RMF using the integrated library functions and data types. The purpose of this document is to provide a comprehensive description of the project, and to explain the principals that are vital to the future follower of this project. It aims to give a detailed account of the development processes adopted and decisions that were made by the group during different phases of the development lifecycle. Furthermore, this report includes a detailed outline of the requirements, design, implementation, and testing. The report also discusses the project management techniques employed. Finally, the report concludes with a section on an evaluation of whether the objectives have been met, and future enhancements to the system.

1.2 Problem Space and Work Motivation

As part of DCNDS weekly seminar, John Souter from the London Internet Exchange (LINX) gave a presentation on the issues of monitoring network traffic at the LINX. During this presentation it was apparent that many IXPs have similar network monitoring requirements and all have semantically similar tools that are developed in house. The tools often differ in the presentation of the information. For instance, they have different hardware logs for data source since each IXP runs different types of hardware. The IXPs also use different scripts for processing these logs. As a result, they retrieve, store, and analyse the traffic data in a variety of different ways. These differences make it difficult for the IXPs to share information directly, to use common information for troubleshooting, to make comparisons of multi-site data, or to perform analysis using this multi-site data. Most of IXPs in Europe recognise that this is a growing problem.

Last year's project group has already tackled the problems outlined above. Data representation and format has been standardised by defining a set of protocols for the

information exchanged between the IXPs. System architecture has been designed and a working prototype has been implemented to provide the proof of the concept for this architecture. A toolkit has also been created for data analysis purpose. This year's project therefore aims to re-engineer the ExSERT based on the system architecture, and to provide an easy way for the new RMF creation.

1.3 Objectives and Scope

The project has one main aim that is to re-engineer the ExSERT based on the system architecture developed by last year's project group, but to use completely different programming language from last year. The project aim has been identified to establish a definite goal. In order to reach the goal a number of objectives have been identified. These are smaller sub goals that are significant steps towards achieving the project aim. The project objectives are:

- To re-engineer all the system components based on the system architecture defined by the last year's project group by using Python.
- To implement a new toolkit that is easy to use, easy to deploy and easy to extend, that allows integration of existing data source such as ping, Fping, SNMP etc and proprietary such as log files.
- To create a compiler that allows the easy creation of the new RMFs.
- To add more data analysis functionalities to the existing toolkit.
- To provide secure data transmission between the IXP and its clients.
- To provide sufficient testing on the complete toolkit and to write down detailed report in order to lay a good foundation to the future project handlers.

As this project is suggested by LINX, the project mainly concentrated on the requirements of this client. Due to the time limitation and lack of access to the relevant resources, the testing and deployment of this toolkit was not performed in LINX or other IXPs. However, relevant documentation was created that gives detailed instructions on how to deploy and use this toolkit. The detailed instructions are included in the user manuals. More specific project scope can be found in Chapter 7 Project Management.

Chapter 2 Background

This chapter provides the reader with some background information on general and technological aspects that are relevant to this project. Since the project was carried out on behalf of the IXP, some background information on IXPs is included. In addition, an example network management tool and two standard data representation techniques have been evaluated and discussed.

2.1 Internet Exchange Point

According to Wikipedia, an Internet Exchange Point (IXP) is “a physical infrastructure that allows different Internet Service Providers (ISPs) to exchange Internet traffic between their autonomous systems by means of mutual peering agreements”. The IXP allows the ISPs to exchange domestic Internet traffic locally without having to send those messages across multiple international hops to reach their destinations. Normally there is a single physical connection from each ISP to the IXP. This infrastructure reduces the cost of the Internet connectivity and bandwidth, and improves the quality of the service, resulting in better and cheaper service for the end user.

Peering is “the interconnection mutual business arrangement between at least two ISPs whereby each directly exchanges traffic to and from each other’s clients” (AfNOG 2001 Meeting Presentation). An IXP does not normally take part in the negotiations of the peering agreements, and it only acts as a neutral interconnection point where ISPs are physically interconnected with each other and where traffic is forwarded to the peering partners. The peering relationship not only reduces cost and reliance on purchased Internet bandwidth and transit, but also lowers inter-AS traffic latency.

According to Wikipedia, an autonomous system (AS) is “the unit of router policy, either a single network or a group of networks that is controlled by a common network administrator (or group of administrators) on behalf of a single administrative entity”. The single entity is usually assumed to be an ISP, a university, or a business enterprise. Networks within an AS communicate routing information to each other using an Interior Gateway Protocol such as RIP or OSPF. An AS shares routing information with other ASs using the Exterior Gateway Protocol such as BGP.

2.2 Secure Socket Layer

The Secure Sockets Layer (SSL) is a protocol developed by Netscape for transmitting private documents via the Internet. In terms of the protocol stack, SSL sits directly above transport layer protocol and below application layer protocol, and provides a secure end-to-end connection. The functionality provided by SSL is intended to allow server authentication, client authentication, and encrypted connections. The server authentication allows a user to confirm a server's identity using standard public key mechanisms, and the client authentication allows a server to confirm a user's identity in a similar way. The encrypted connections allow all communications between the client and server to be encrypted and singed.

2.3 Simple Network Management Protocol

Simple Network Management Protocol (SNMP) is the Internet standard protocol developed to manage nodes (such as servers, routers, switches and hubs etc.) on an IP network. SNMP enables network administrators to manage network performance, find and solve network problems, and plan for network growth.

SNMP contains two primary elements: network management systems (NMSs) and agents. The NMS is the console through which the network administrator performs network management functions. Agents are the entities that interface to the actual device being managed. Routers, bridges, hubs, or network servers are examples of managed devices that contain managed objects. These managed objects might be hardware, configuration parameters, performance statistics that directly relate to the current operation of the device in question. These objects are arranged in what is known as a virtual information database, called a management information base (MIB). SNMP allows managers and agents to communicate for the purpose of accessing these objects. The following figure taken from Cisco's Internetworking Technology Handbook illustrates the system architecture of an example SNMP implementation:

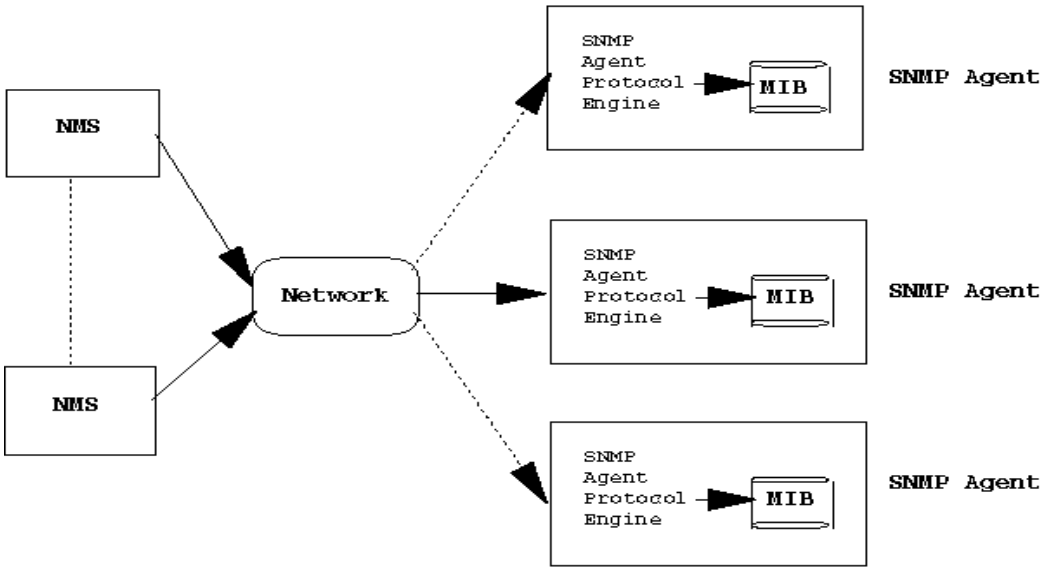


Figure 2.1: SNMP System Architecture

SNMP must account for and adjust to incompatibilities between managed devices. Different computers use different data representation techniques, which can compromise the capability of SNMP to exchange information between managed devices. SNMP uses a subset of Abstract Syntax Notation One (ASN.1) to accommodate communication between diverse systems.

2.4 ASN.1

Abstract Syntax Notation One (ASN.1) is an International Standards Organization (ISO) data representation format designed to describe the structure and syntax of transmitted information content. ASN.1 provides for the definition of the abstract syntax of a data element (or data type). The abstract syntax describes the syntactical structure and typed contents of data that are subsequently to be transmitted across some medium such as the Internet. The language is based firmly on the principles of type and value, with a type being a set of values. The type defines what values can be sent at runtime, and the value is what is actually transmitted across the medium at runtime.

The values are encoded before transmission using one of a number of different encoding mechanisms such as the Basic Encoding Rules (BER). The encoding rules specify how the values of the abstract data types are converted into byte string ready for transfer. The recipient must usually be aware of the type definition before receipt,

as this is not transferred but must be inferred from the context in which the message exchange takes place. During the transmission the data stream is never in a form readable by human operators. Only when it has been transformed into some local data display format, prior to encoding or after decoding, can it be easily read by humans.

2.5 XML

The Extensible Mark-up Language (XML) is a W3C-recommended general-purpose mark-up language for creating special-purpose mark-up languages. It is a simplified subset of Standard Generalized Mark-up Language (SGML) and is used widely for exchange of structured information over the Internet. XML supports the definition of a set of mark-up tags relating to the content of documents, thus delivering both extensibility and potential for validation. Mark-up tags can be defined for different type constructors so that complex data structures can be built. XML uses a Document Type Definition (DTD) or an XML Schema to describe the data.

XML is transferred in textual format with no binary encoding or compression, remaining in a constant human readable format throughout the transmission. The recipient has to examine every byte received in order to determine the end of a data value. Constraints can be imposed on the XML document structure with the provision of DTDs or XML schemas, which describe the allowed mark-ups that a conformant XML document can contain.

Chapter 3 Requirement

This chapter gives an overview of the requirements of the system that will be developed. Since last year's group have already implemented a working prototype to provide the proof of the concept for the system architecture, one key function requirement for this year is to re-engineer the system based on this system architecture. The functional requirements are concerned with representing similar data in a common format, generating the common code from ExSERT schema, providing a common APIs for data analysis and data representation, and securing the communication between the client and server. The non-function requirements have also been detailed, including implementing the system using Python and releasing the software as open source, and requirements that drives the development of this toolkit. For instance, the toolkit needs to be easy to use, to deploy and to extend as well as easy integration with the existing back-end tools such as Ping, Fping or SNMP.

3.1 Functional Requirement

3.1.1 Represent similar data in a common format

Last year's group has standardized the data representation for the network traffic exchanged between the IXPs. A set of protocols has been defined to provide communication over the network between different components in the system. These protocols also resolved the data heterogeneity generated by different back-end network tools. However, these protocols are not standardised by any standardisation organization, resulting in these protocols may not be accepted by other IXPs. Therefore a standard way to represent data is required to be adopted.

3.1.2 Create a compiler to generate common classes and code

The compiler is required mainly to generate the common code for the system components. By having a compiler would certainly speed up the development of new RMF tools. Furthermore, regarding the VP protocol defined in last year's project, given a set of types to transmit, the functions that convert these types into the array of bytes to send would be written by hand. This problem is also outlined as a future work in last year's report. To improve this deficiency, this compiler is also required to generate these functions automatically according to the ExSERT schema which

acts as a report format for the compiler. In other words, IXPs could deal with their different data formats from back-end tools by using this complier rather than writing a fixed program. The report format specifies the information structure transferred across the network between different IXPs. As a whole, the compiler is used for the auto-generation of the protocol and information processing.

3.1.3 Provide a common APIs for data analysis and data representation

Last year's group has developed a set of functionality for data analysing based on the working prototype. Since the data representation for this year's toolkit is different from the previous one, all the functionality needs to be re-engineered according to the new data format. Furthermore, additional functionalities are required to be built in order to provide a complete toolkit. After the common code has been generated from the compiler, the code for data analysing can be added by the application programmer. By providing a common API, application programmers can insert these functionalities easily.

Another part the API should provide is the concrete implementation of the basic data types that are used to represent the network traffic, such as IP address. In terms of the system architecture, these concrete implementations would be used by both the IXP and its clients to process the data transmitted across the network.

3.2 Non-Function Requirement

3.2.1 Implement the system using Python

Most of IXPs prefer scripting language for their data processing and particularly LINX uses Python for its existing monitoring system. This is due to the fact that Python is interpreted language, which is easy to implement, easy to use, multi-platform, flexible, and efficient. Therefore the toolkit is required to be implemented in Python.

3.2.2 Easy to use, deploy and extend

As the toolkit will be used mostly by the site administrator who might not be an experienced application developer, the toolkit is required to be easy to use and easy to deploy. In addition, the toolkit is required to be extended easily, allowing the simple adding of new report and event types. The compiler tool mentioned in the functional requirement would certainly help to achieve this requirement.

3.2.3 Secure access and communications

As the data that LINX or other IXPs generate from monitoring their customers' traffic and their own hardware have to be kept confidential, there is a requirement to ensure its confidentiality through encryption as well as providing mutual authentication of the parties involved in the exchange of these data. Furthermore, the secure access (on the wire or site level) would be controllable by the site administrator.

3.2.4 Release the software as open source

This monitoring toolkit will be released as open source. Open source offers a radically different and exponentially better software development model. With many open source projects, a virtual community of developers grows around the software. Everybody contributes to produce a higher quality product than could have been produced independently. Furthermore, more programmers are better since the more people looking at a piece of code, the more likely one of them is to fined a bug before it gets to be a major problem.

Chapter 4 System Architecture

As the system architecture has been well developed in the last year's project, this year's project mainly focuses on the auto-generation of the system components and common APIs that provides the easy creation of the functionalities of the Remote Monitoring Function (RMF) or the toolkit. This chapter provides an overview of the system architecture, and a brief discussion on how this auto-generation is achieved.

4.1 System Architecture

The RMF consists of several components that are real resource, Glue, PoD, and Visualization (front-ends). The following figure shows a typical RMF, and the subsequent paragraphs outline a brief description of these components.

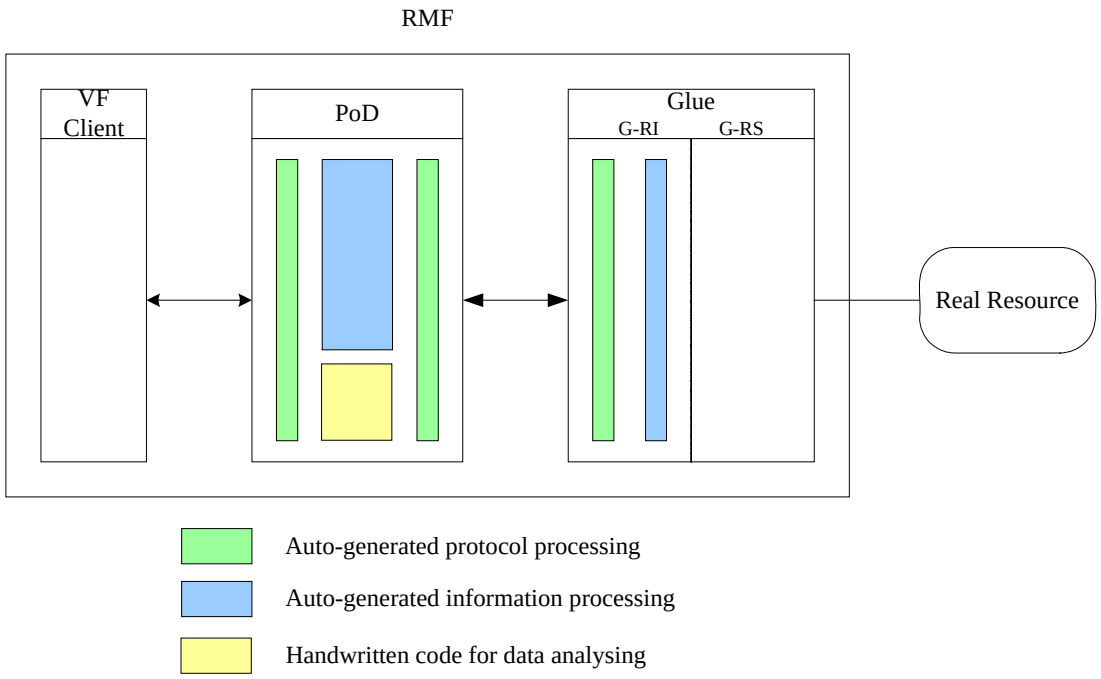


Figure 4.1: System Architecture

The real resource acts as an input for the RMF. It might come from the IXP, which uses some network monitoring tools to gather network information from its customers such as an ISP. It could also be the information generated from a switch or

a router. The real resource can be obtained either at run time or from the log file supported by a particular IXP in a certain time period.

The next component is called Glue as it provides the “glue” between the real resource and the rest of the system. There are two logical parts in the Glue namely, Glue resource specific part (G-RS) and Glue resource independent part (G-RI) respectively. G-RS is responsible for dealing with network information in a non-standard format obtained from the IXP, switches or routers and storing it. G-RI is responsible for parsing received data from the G-RS into a standard format and saving it to a file. XML has been adopted for the standard representation of the data, and will be discussed in more detail in the next chapter. For the ease of reading, XML is notated in the following discussion instead of the standard representation. The standardised data will be passed to the PoD for further data processing.

The next component, called PoD, performs three functions: it analyses the data supplied by the G-RI, acts as server that accepts connections from authorized clients, and pushes all the processed data to the clients. A set of functions contained in the API can be used for data analysing.

The last component is called visualization component. It can be a graphical user interface, a web page or just a plain text file. In a word, it is up to the end users to choose one of the display types, which can tailor their requirements perfectly. In fact, XML document is already well formatted and can be displayed in a current web browser without any changes. XML can also be combined with other layout languages (such as HTML based web documents) to produce highly readable output. Unlike last year’s project, no particular GUI has been implemented to display the information received from the PoD. However, a client demo has been implemented, which converts the received XML document to HTML format, which then can be displayed by the browser.

The whole process starts when one or more clients connect to the PoD server, which triggers the Glue to begin requesting and retrieving data from the real resource. While there is no client connected to the server, the Glue stops requesting and retrieving data. Of course, the PoD server is still running to accept new clients.

The compiler is used to auto-generate the code of protocol processing and data processing for system components, which has been shown on the above diagram. To the end users, what they need to do is just to modify the input ExSERT schema and the configuration file of the compiler initially, and then the complier will generate all

the common source files. After appending the code for retrieving the back-end data for the G-RS and the data analysing code in the PoD, a completed network monitor tool is generated and ready to use.

As discussed in the last year's report, the system needs to have some mechanisms whereby the clients can find the specific PoDs. In order to solve this problem, a PoDRegistry has been designed. Each IXP has its own PoDRegistry, which is empty at the very beginning. When a PoD starts running, it connects to the PoDRegistry and registers itself with the PoDRegistry by informing the registry of its host name, port number, types of PoD and a short description of what it does. The PoDRegistry runs at a well-known port 5555 and at a well-known address determined individually by each IXP, so that the PoD knows where to contact the registry. For the clients, they must initially connect to the PoDRegistry server rather than any PoD server. Then the clients can obtain a list of current registered and connected PoD servers from the PoDRegistry in which they can choose one and set up a connection with the specific PoD. After the connection has been established between the clients and the PoD server, the data transmitting from the PoD server to the clients have been built and kept running. PoD server will stop transmission while there is no client connected to the PoD server. The PoDRegistry can be in different locations with a well-known socket port number 5555 or use different mechanism such as a URL web site.

Since the data being transmitted is likely to be confidential, according to one of the client's requirements, all communication between the PoD servers and the clients are done using SSL and mutual authentication. Communications between the PoD server and PoDRegistry, and between the client and the PoDRegistry are unencrypted since the location and registry information of a PoD is not confidential. In fact, in order to established a SSL connection between a client and a PoD, a Certificate Authentication (CA) has to be set up by which both clients and PoD server can verify each other's certificate before encrypting and transmitting data. Due to the complexity, a CA is running in the same machine with the PoD server in order to simplify the CA operation. This part can be improved in further development.

4.2 Advantages

The most significant advantage of the compiler and API is that it allows the users to easily create a new RMF to meet their needs. To create a new RMF, the users only need to make slight modifications on the common code generated by the compiler. Without the compiler, the development of a new RMF normally takes more time since all the common code needs to be re-written for different RMFs. To further ease the creation of the new RMF, a common API is developed which provide both the functions used to analyze the data and the basic data types for the data representation. The developer can use the libraries in the API to add different functionalities of the RMF.

Chapter 5 Design and Implementation

This chapter provides a detailed discussion on the issues aroused during the design and implementation phase of the project, and many decisions made by the group. The design and implementation is based on the system architecture developed in the last year's project. A prototype as a simple RMF has been designed and implemented first in order to determine the structure and sources of the target that will be generated by the compiler. After the development of the compiler, more functionalities of the RMF have been built up in order to refine the compiler. These functionalities are included in a common API, which also contains the implementation for basic data types. This chapter also includes the discussion about the re-engineered PoDRegistry server and its communication protocols, the SSL used between the client and PoD server, and the binary encoding scheme.

5.1 Data Representation

A number of standardised protocols can be used to represent data for the network traffic exchanged between IXPs. Two most popular techniques XML and ASN.1 have been examined and detailed in the background chapter. XML was chosen for the toolkit because it has many advantages over other techniques.

Firstly, over the past couple of years, XML has become the preferred syntax for the transfer of information across the Internet. XML has received the wide spread backing and endorsement from major IT industry organisations like Sun Microsystems, IBM and Microsoft.

Secondly, constraints can be imposed on the XML document structure with the provision of Data Type Definition (DTD) or XML schema, thus XML is easier for validation. Validation is an important process for the recipient of XML documents that might be corrupted during the transmission. XML provides a set of element types, which serve to define types of documents. DTD or XML schema contains set of rules to control how documents and tags are structured, which elements are presented and the structural relationship between the elements for documents of a particular type.

Thirdly, XML is transferred in textual format with no binary encoding or compression, remaining in a constant human readable format throughout the transmission. Application programmers can build simple parsers to read XML data,

making it good format for interchanging data. XML is easier to debug since the data stream can be read without any special software tools. XML seeks to “achieve a compromise between flexibility, simplicity and readability by both humans and machines” (Emmerich 2000).

Finally, XML can be combined with other layout languages (such as HTML based web documents) to produce highly readable output.

On the other hand, XML presents a disadvantage as XML is very verbose, and consequently create large data streams during transmission. As a result, this might hinder the performance of the transfer and consequently, the effectiveness of displaying real-time data on the recipient side. A binary encoding scheme has been defined to solve this overhead, and the detailed description about how binary encoding works will be discussed in the following section. In addition, this encoding scheme also alleviates some of the performance penalties suffered from the use of SSL-encrypted communications.

XML is the abstraction or specification defining the data types and structured information transferred across the Internet. Concrete implementation is needed to allow the machine to understand and store the data. Due to the requirement, the implementation is done in Python. XML elements are mapped to the Python classes and each Python class has a method that converts the data to a XML format. The XML document can also be parsed and converted back to the Python implementation using the program from the standard Python library. The following figure illustrates the architecture of this two-way mapping:

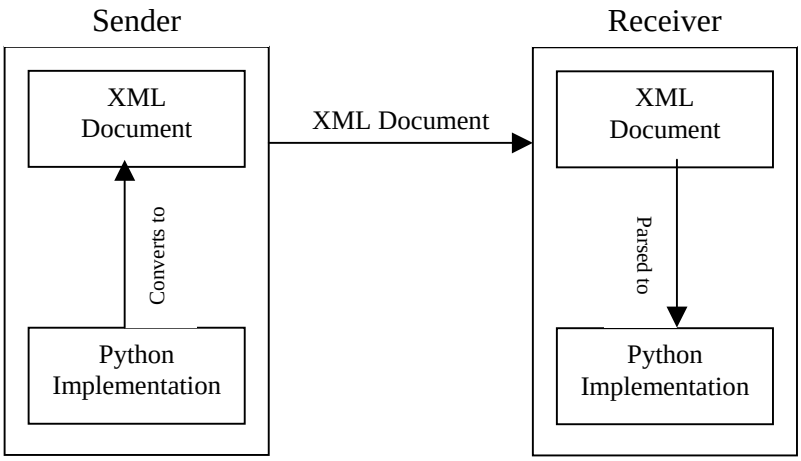


Figure 5.1: Data Representation and Mapping

5.2 Validation

Validation is the process to check whether the XML document is “well-formed” and “valid”. A "well-formed" XML document is a document that conforms to the XML syntax rules and a "valid" XML document is a "well-formed" XML document, which also conforms to the rules of a DTD or an XML schema. Most of high-level programming languages have built-in libraries for XML validation, and the validation process can be easily performed.

Relating to the system architecture, the validation process needs to be performed before the PoD sending the XML document and after the client receiving it. The DTD or XML schema is normally predefined and distributed to both the PoD and client before data transmission has been taken place. The XML document was created by the Glue and passed to the PoD for data processing, thus the first validation phase is required to check whether the XML document is valid. If it is not valid, the PoD will not process the data, and consequently the document will not be sent the client. The occurrence of the error in this stage is nearly impossible as the document was automatically generated by the Glue rather than being written by hand. The validation process is very important on the client side since the XML data might be corrupted or damaged during the transmission across the network. If the XML document is not valid, the client will ignore it to avoid unnecessary transformation for displaying its contents.

According to the W3C recommendation, there are two types of validation rules as discussed previously: DTD and XML schema. The purpose of DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in the XML document, or as an external reference. XML schema is an XML based alternative to DTD. Compared to DTD, XML schema has a number of advantages. One of the greatest strengths of XML schemas is the support for data types. With the support for data types, it is easier to define restrictions on data in programs using regular expressions. In other words, XML schema can check semantic errors as well as verifying syntactic mistake, whereas DTD only support the latter.

Although XML schema is more mature than DTD, DTD has been chosen for this toolkit instead. The reason is that one of the requirements is to implement the system using Python, which only supports the DTD validation in the current

implementation. Alternative way is to use third party tools for XML schema validation. This has not been adopted due to the fact that either the kinds of tools are too complex or lack of supported documentation, or the tools has copyright legitimacy.

5.3 Data Types

5.3.1 Basic data type

XML provides a set of element types, which serve to define types of documents. An XML element is everything from (including) the element's start tag to (including) the element's end tag. An element can have element content, mixed content, simple content, or empty content. Relating to the data representation of the toolkit, elements with a single content are used to represent the basic data types. The basic data type is the basic unit of the network information. For instance, an output from ping measurement look likes “64 bytes from 64.233.183.104: imcp_seq=0 ttl=238 time=29.8 ms”, and the IP address can be seen as a basic unit. The following table summarised the basic data types defined in the toolkit:

Basic Data Type	Description
Year	A string with a specified format, e.g. 2005
Month	A string with a specified format, e.g. 10
Day	A string with a specified format, e.g. 29
Hour	A string with a specified format, e.g. 20
Minute	A string with a specified format, e.g. 30
Second	A string with a specified format, e.g. 45
IPv4Address	A string with the format specified in IPv4
IPv6Address	A string with the format specified in IPv6
MACAddress	A string with the format specified in MAC address
RTT	A integer represents the round trip time
NumOfBytes	A integer represents the number of bytes transferred
PacketLoss	A percentage represents the packet loss rate

Table 5.1: Basic Data Types

There are two universal types String and Integer, from which all other data types (including basic and user-defined types) are derived. In particular, the Integer type is just a signed integer type, as the network information usually does not have the format of a negative integer. Year, Month, and Day are defined individually and combined together to represent date information. The reason why they are defined

individually is that it is easier for data processing in the case of a client requires certain monitoring information in a particular time period. Hour, Minute, Second are defined and combined together to represent time information in a similar way. RTT and NumOfBytes are not supposed to be classified as the base data types because they are just types of an integer with no special format. However, there is a limitation when using DTD since DTD only supports for character data. There is no way to define RTT as an Integer in DTD, thus RTT and NumOfBytes need to be predefined for the purpose of validation. IPv4Address, IPv6Address, MACAddress are three address formats commonly used in the network information.

As discussed in section 4.2, unlike XML schema, DTD only supports for checking the syntactic error for a given XML document. The semantic checking is also required to be performed in a way that the data has a correct meaning. As discussed, the abstract data types defined in XML/DTD have corresponding implementation Python classes; the semantic checking is also carried out in these Python classes. Regular expressions have been used for this checking since it is a powerful and standardized way of searching, replacing, and parsing text with complex patterns of characters.

5.3.2 User-defined type

The user-defined type is the data type constructed by the application programmer. It can be a structured type or just a type directly derived from String or Integer. The latter is often used to store the result calculated from other data types. The structured type can also include other structured types. The network information usually consists of a set of basic data types in the structural way. For instance, the Time information is composed of Hour, Minute, and Second, which are three basic data types. Time is referred to a user-defined type that is constructed using basic data types. In terms of an XML document, the structured type can be represented by elements with element contents. The following XML tag shows an example of a structured type:

```
<Time>
    <Hour></Hour>
    <Minute></Minute>
    <Second></Second>
</Time>
```

In terms of the concrete implementation, the structured type is also represented by a Python class that is composed of other classes.

5.4 Binary Encoding

This section describes the related concept of binary encoding, explains the reason why binary encoding is necessary and specifies how to use it.

XML is a useful, portable format for exchanging data between different applications. While XML solves the problem of data heterogeneity, XML's processing overhead, storage requirement, and bandwidth consumption become quite problematic when transaction volumes are high. In many cases wireless networks are slow (data throughput possibly as low as 4KB per second), passing XML documents between a server and a J2ME-enabled device might be too slow to be workable. Also, parsing large documents can easily cause out-of-memory errors on very constrained devices.

One approach to improve the performance of XML is to compress XML directly. Although compression may solve the bandwidth issues in the most straightforward approach to reducing the size of XML documents, it worsens the processing problem at both sides of the sender and recipient to apply compression technologies. Furthermore, compression formats like zip or base64 offer an "all or nothing" approach. So any marginal gains in network bandwidth are also lost in processing time. To resolve the limitations of all-or-nothing compression and its processing overhead, we developed the binary representation of XML.

The binary format was designed to allow compact transmission with no loss of functionality or semantic information. The binary format encodes the parsed physical form of an XML document, for instance, the structure and content of the document entities. Meta-information, including the document type definition and conditional sections, is removed when the document is converted to the binary format.

Furthermore, high throughput transaction processing systems, low bandwidth communications and low-power processors with small memory are not generally places where complex compression algorithms are worthwhile. So this encoding uses binary, rather than text-based, means for serializing and transmitting XML information. It promises to significantly alter the processing, bandwidth, and storage penalties that currently plague XML.

Moreover, binary encoding is faster than textual encoding. Also, the binary format is not proprietary and is supported by a lot of tools. The binary encoding is already standardized, mature and stable, high performance, and supported by large scale of commercial and free software. It reduces the message size and provides interoperability and self-describing format. In addition, it can be accessed randomly by using binary tags.

Therefore, encoding or transforming XML into a binary format is usually a better solution.

In order to understand binary encoding more, ASN.1 has to be introduced herewith. As described in the background chapter, ASN.1 is used to describe messages exchanged between communicating application programs. It provides a high-level description of messages that frees protocol designers from having to focus on the bits and bytes layout of messages.

One of the main reasons for the success of ASN.1 is that it is associated with several standardized encoding rules such as the Basic Encoding Rules (BER) - X.209, Canonical Encoding Rules (CER), Distinguished Encoding Rules (DER), Packed Encoding Rules (PER), and XER Encoding Rules (XER). These encoding rules describe how the values defined in ASN.1 should be encoded for transmission, regardless of machine, programming language, or how it is represented in an application program. ASN.1's encodings are more streamlined than many competing notations, enabling rapid and reliable transmission of extensible messages - an advantage for wireless broadband. Because ASN.1 has been an international standard since 1984, its encoding rules are mature and have a long track record of reliability and interoperability.

Closely associated with ASN.1 are sets of standardized encoding rules that describe the bits and bytes layout of message as they are in transit between communicating application programs. The encoding rules provide a means of going from the local concrete syntax to the transfer syntax and reverse.

The basic data structure format is called TLV that describes XML data as triplets <data Tag, data Length, data Value> by which specifications of information can be handled by high-level protocols with no loss of generality, regardless of software or hardware systems.

There are compact binary encoding rules (BER, CER, DER, PER, but not XER) are considered alternatives to the more modern XML. In this project, BER is considered based on the following analysis.

BER stands for Basic Encoding Rules. These rules describe how to encode and decode basic data types and composite data structures to and from TLV streams. A TLV hence may be primitive (atomic) or constructed (nested) where the value component contains other TLVs. The T is for the Tag a numeric type identifier; the L is for the length of the data carried in the third V component, the value. TLV structure must be maintained throughout. Unlike XML, BER tag and value are binary rather than textual. For instance, XML is like <tag>value</tag> (<IPv4Address>127.0.0.1</IPv4Address>) whereas BER is like tag/length/value (0x23/0x33/0xFE). Therefore, the XML data is encoded using the TLV-style BER for transmission - a binary format. The entire PDU (data and "header information") is a single BER entity.

In addition, endianness like big endian or little endian has to be indicated since some kind of data such as IPv4Address and RTT would take account of multi bytes. In case of the binary encoding, it is essential that which endian will be adopted. In fact, it does not matter either big endian or little endian is used as long as encoding and decoding are consistent. In other words, if encoding uses big endian, decoding has to use big endian too. In this project, big endian was used due to human reading habit.

In the following, detailed definition and design of tables such as data type table and encoding rules table are demonstrated respectively. There are currently three types of data such as integer, string and IPv4Address. It is obvious that more data types might be added in the future. At the moment, the length for encoding data type is four bits. This means the current encoding rules can deal with sixteen kinds of data at most.

In the case of concrete element data encoding in XML, the length of encoding bits for each element totally depends on the dedicated data type and data value range.

Number in decimal	Binary code	Type name
0	0000	Integer
1	0001	String
2	0010	IPv4
3	0011	IPv6
4	0100	MAC address

Table 5.2: Data Type

Data Field Name	Length (bits)	Range From	Range To
YEAR	6	000000	111111
MONTH	4	0001	1100
DAY	5	00001	11111
HOUR	5	00001	10111
MINUTE	6	000000	111011
SECOND	6	000000	111011
NUMOFBYTES	8	00000000	11111111
RTT	16	0x0000	0xFFFF
IPV4ADDRESS	32	0x0.0x0.0x0.0x0	0xFF.0xFF.0xFF.0xFF
THRESHOLD	2	00	11

Table 5.3: Binary Encoding

Tag	Length	Value

Table 5.4: TLV

Tag	Len	T	L	V	T	L	V	T	L	V

Table 5.5: Data Packet Architecture

In order to explain the binary encoding further, a concrete XML file in the following will be taken as an example. This XML file acts as the input of the binary coding function, and the output is an encoded binary string that will be written into a binary file before being sent over the network.

Input file:

```
<RTTMeasure>
    <Timestamp>
        <Date>
            <Year>2020</Year>
            <Month>10</Month>
```

```
            <Day>30</Day>
        </Date>
        <Time>
            <Hour>13</Hour>
            <Minute>35</Minute>
            <Second>29</Second>
        </Time>
    </Timestamp>
    <NumOfBytes>100</NumOfBytes>
    <IPv4Address>192.168.100.9</IPv4Address>
    <RTT>15</RTT>
    <Threshold>INFO</Threshold>
</RTTMeasure>
```

Result:

0001000000000110010100000000000100101000000000010111110000000000101011010000000011010001100000000011001110100000000100001100100001000100000011000000101010000110010000001001000000010000000000000000011111000100000001000

5.5 Glue

5.5.1 G-RS

The purpose of the G-RS is to process the network information in a non-standard format obtained from the IXP, switches or routers and to store it in the Python objects. The implementation of the G-RS is specific to IXPs since different IXP might have network information in a different format. If the real resource is the log file, the G-RS processes the file directly. If the real resource is the runtime data obtained from the IXP, the program that can acquire this data is integrated into G-RS. The command to run the program is usually called a back-end tool, which can be specified in the configuration file of this particular RMF. It is usually an ICMP or a SNMP command. The real resource is parsed line by line, and the corresponding Python objects are created to store the processed result. Here are five lines produced by an ICMP tool (fping) command:

```
[2005-07-08-144433.656091] www.google.com : [0], 84 bytes, 30.1 ms (30.1 avg, 0% loss)
[2005-07-08-144433.664449] www.google.com : [1], 84 bytes, 30.1 ms (30.1 avg, 0% loss)
[2005-07-08-144433.673309] www.google.com : [2], 84 bytes, 30.1 ms (30.2 avg, 0% loss)
[2005-07-08-144433.675252] www.google.com : [3], 84 bytes, 30.1 ms (30.1 avg, 0% loss)
[2005-07-08-144433.684907] www.google.com : [4], 84 bytes, 30.1 ms (30.1 avg, 0% loss)
```

For each line, the required information is extracted and stored in a Python object. This object may contain other Python objects. As discussed in the 5.1, these Python objects refer to the concrete implementation of the data types.

Since the back-end tool generates the output continuously, piping has been to pass the output to the G-RS. A pipe is essentially a channel in which one program can communicate with another by sending or receiving text string. Using the pipe, the back-end tool pushes the output into one end of the pipe, and the G-RS pulls the output from the other end of the pipe. However, one problem is found when using the pipe in Python. The problem is that the back-end tool does not push the output until the pipe buffer is full. The pipe buffer is 47 by default, and there is no way to modify the size of the buffer in Python. As the toolkit requires the continuous data, the delay produced by the pipe is not suitable. To solve this problem, the back-end tool has to terminate after producing a number of outputs and restarts. If the number of outputs is less than the pipe buffer size, the pipe will be forced to push the data to the G-RS. Therefore, this solution prevents the real time data from delaying with the tradeoffs of consuming more CPU usage, as terminating and restarting the back-end tool consumes the kernel usage. The number of outputs that the back-end tool produce can also be specified in the configuration file.

The interval between terminating and restarting the back-end tool also allows the G-RS to pass the result to the G-RI. Otherwise, threading technique needs to be used for the intercommunication between these two Glue components.

One alternative way to pass the generated output to the G-RS is to store the output in a file, which will be accessed later by the G-RS. This technique has not adopted since the file will not be read by the G-RS until the back-end tool finishes writing, resulting in the latency of the real time data.

5.5.2 G-RI

The purpose of the G-RI is to parse received data from the G-RS into a standard format and to save it to a file, which will be accessed by the PoD. As described before, the standard format is an XML document. A list of Python objects representation data types has been passed from the G-RS, and the G-RI process one object a time. The getXML() method in these objects are called to construct the XML representation of the data type. After building these data types, the root of the XML

document is added. Other XML parameters including the version, the encoding, and the DTD schema are also inserted to produce a well-formed XML document. These parameters are specified in the configuration file, and the constructed XML document is saved to file that will be accessed by the PoD later.

If the back-end tool does not work properly or there are errors in retrieving the real resource, it will generate an empty file. When the PoD detects an empty file, the PoD will notify clients that it cannot produce any monitoring information.

As the G-RS part is implementation specific, the G-RI provides a well-defined interface to handle diverse G-RS implementations. It is designed in a way that all G-RS implementations encapsulate their code for processing non-standard format data in a particular method that will be called later by the G-RI.

5.6 PoD

As described in Chapter 4, the PoD has three main responsibilities: analyses the data supplied by the G-RI, acts as server that accepts connections from authorized clients, and pushes all processed data to the clients. This section details the different design strategies for the PoD and the design decisions made by the group.

5.6.1 Event Infrastructure

The current toolkit has been implemented using push model and TCP connection. At the first beginning, an event infrastructure was considered. In case of event infrastructure, the PoD server pushes data into the event channel by which data will be forwarded to different clients using IP multicast technology. This form of asynchronous messaging is a far more scalable architecture than point-to-point alternatives such as message queuing, since message senders need only concern themselves with creating the original message, and can leave the task of servicing recipients to the messaging infrastructure. It is a very loosely coupled architecture, in which senders often do not even know who their subscribers are.

However, the above publish/subscribe architecture is not suitable for this project based on the following reasons. Firstly, unlike TCP, IP multicast uses UDP that provides unreliable data transmission between the server and client (the IXP and its clients in terms of the toolkit). Thus, it would be essential to add reliability if the event infrastructure is adopted. Secondly, data communication in event-driven architecture is asynchronous; the current network monitoring toolkit is real time and

synchronous. For instance, it will be meaningless if PoD server pushes data continuously without any client receives it simultaneously. In addition, it is not easy to implement a mediator server that accepts message from PoD server and multicasts it to eligible clients. In fact, this would add one more layer into original system architecture. Finally, in terms of data security, it might not be preferred by LINX since both PoD server and client do need to know each other in publish/scribe architecture. Obviously this could not be accepted according to requirements. Furthermore, implementing SSL is more complicated in event architecture since multicast is not compatible with SSL. Therefore, a point-to-point TCP connection and push model is adopted.

5.6.2 Data transmission model

There are two kinds of data transmission model named pull and push, which can be used between clients and the PoD. Both of them have their own advantages and disadvantages.

In general, the pull model means that clients send request to the server and get dedicated result from the server based on their requirements. As different clients may have different requirements, the server will deal with them differently. As a result, it might cause overhead on the server while thousands of clients connect to the server at the same time. The advantage of this model is that it can satisfy all the clients with different requirements.

On the other hand, the push model means the server pushes the same data to all connected clients simultaneously. This model improves the performance significantly although it cannot meet some clients' specific requirements. However, API functions can be used to make different kinds of PoD, which can satisfy different clients with diverse requirements. In other words, clients with different requirements can connect to different PoD servers. Therefore, inside one particular PoD server, the push model is more suitable than the pull model and preferred for the transmission between clients and the PoD.

5.6.3 Multiple clients' connections

TCP has been selected for the implementation of the push model since it requires reliable and continuous connections between clients and the PoD. The PoD server also needs to be a concurrent server to deal with multiple connections from the

clients. Three techniques can be used to design a concurrent server when using TCP: forking, multi-threading, and using non-blocking sockets.

The simplest way to write a concurrent server in Unix is to fork a child process to handle each client. When a connection is established, the server calls fork(), and the child process services the client and the parent process waits for another connection. The parent process closes the connected socket since the child handles the new client. If plenty of clients connect to the server simultaneously, the server needs to create a large number of processes, which will decrease the performance of server rapidly, and even worse, make the server crash. Therefore, the forking paradigm is not suitable since the PoD server needs to handle a great number of clients' connections.

Multi-threading is another way to implement a concurrent server. By multi-threading, it means the server has each client served by a separate thread of control. Multi-threading is far more advantageous than forking due to a number of reasons. Firstly, thread is sometimes called lightweight process since thread creation can be ten to one hundred times faster than process creation. Secondly, multi-threading is less expensive since all threads within a process share the same global memory, whereas in forking, the memory is copied from the parent to the child and all descriptors are duplicated in the child. However, the synchronization among different threads might reduce the server performance as well when there are a huge number of clients' connections.

By default, sockets are blocking; this means that the process will not resume execution until the socket call has completed. With non-blocking sockets, the socket call returns immediately, even if it cannot be completed. The non-blocking socket and select function often combines to serve as a concurrent server to handle multiple clients. When accepting incoming connections on the non-blocking socket, if the new connection is not available, the error is returned instead of putting the process to sleep with the blocking socket. The select function is used to determine when a socket descriptor is readable, and is often called within a loop. In other words, if this is a client ready for connecting, the server will establish this connection. If problem occurred during the establishment of this connection, the server process will not block but process the next client request.

The third technique has been chosen to handle multiple clients' connections, since it overcomes the overhead discussed in the previous paradigms. It is also

proved that this technique has better performance of simultaneous connections than using threads.

5.6.4 Implementation

Three classes have been designed for the PoD component: PoDServer, PoDProcesser and PoDDataAnalysier. PoDServer is responsible for registering the PoD with the PoDRegistry server and accepting connections from the clients. PoDProcesser is responsible for processing the XML document received from the Glue and pushing processed XML data to the connected clients. The processing is actually done by PoDDataAnalysier that invokes the API functions for data analysing. The class diagram shows the relationship between these classes, and a detailed version is included in Appendix B:

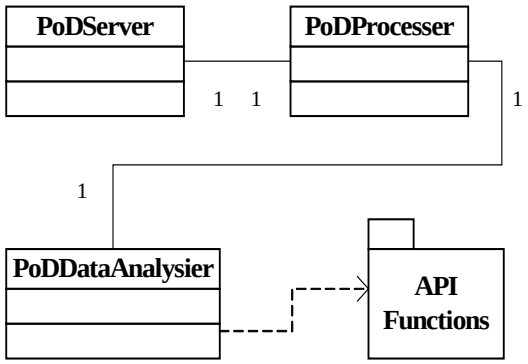


Figure 5.2: Class Diagram for the PoD

Firstly, the PoDServer initialises two sockets. One socket is used for sending the PoD information to the PoDRegistry, and the other is used for handling the client connections. After that, it uses a combination of non-blocking and select function to deal with multiple clients' connections. When the connection is detected by the select function, it is added to a list, which is then passed to the PoDDataProcesser object for data processing. When the PoDServer starts, it creates an object of the PoDProcesser, which will be running until the PoDServer shuts down. The PoDProcesser object is initialised only once and resides in the memory all the time. This strategy is better than using a separate thread to deal with the data processing because the thread requires the synchronization with the PoDServer. Another reason to use this method is that all the clients have the same requirements when using the push model.

When there is at least one client, the requestAndSend function in the PoDProceser class will be invoked with the connection list as a parameter. This function firstly triggers the Glue to return a formatted XML document, and then processes and pushes the new XML document to all connected clients. Instead of pushing the whole XML document, the PoDProcesser sends the file line by line. In this way, it avoids the delay introduced by the large XML document, providing a real time data to the clients. After pushing data successfully, the requestAndSend function returns an updated connection list to the PoDServer in which all clients are currently connecting. In other words, all disconnected clients have been deleted in the process of pushing the XML document. Normally, when calling socket functions such as send, an integer is returned indicating the success of this operation or other status. However, Python socket functions do not operate in this way. Thus the only way to determine the status of socket function in Python is to use exceptions. When an exception is caught in the process of pushing the XML document to the client, it is assumed that this client has a bad connection or even been terminated. This particular connection will be removed from the connection list. In addition, this exception catching technique has also been used in other places such as the situation while client or server crashes.

PoDDataAnalysier is responsible for analysing the data received from the Glue. In different RMFs, the data analysing is different. A number of API functions can be used for this data analysing, and will be discussed in details shortly. The PoDDataAnalysier object is also instantiated when the PoDProcesser object is created by the PoDServer.

The communication between the PoD and Glue is very flexible; several techniques can be taken based on the real situation. For instance, if both the PoD and Glue are in the same host, the communication can be done by sharing a common file or just embedding one process into another one’s main process. In addition, if they are separated in different locations, they have to communicate with each other by using sockets.

In the current system architecture, the PoD and Glue are located on the same machine. The synchronization between the PoD and the Glue is implemented by sharing a common file. When a PoD server starts, it creates one PoDProcesser object, which subsequently creates a G-RI object that triggers the back-end tool to start retrieving the output from real resource. The XML document can be obtained by

calling a dedicate function in G-RI, which writes a formatted XML document into a file. The calling function will return a value to indicate the success or failure of writing XML files. Afterwards, other functions will proceed. Thus, this mechanism guarantees that data access conflict will not happen during the whole process. In other words, the PoD cannot send the data until it has received it from the Glue, otherwise an error will be noted to both the PoD and clients. Furthermore, this synchronization method improves the performance of the PoD, and reduces the usage of CPU and memory. The following class diagram shows the relationships between these system components:

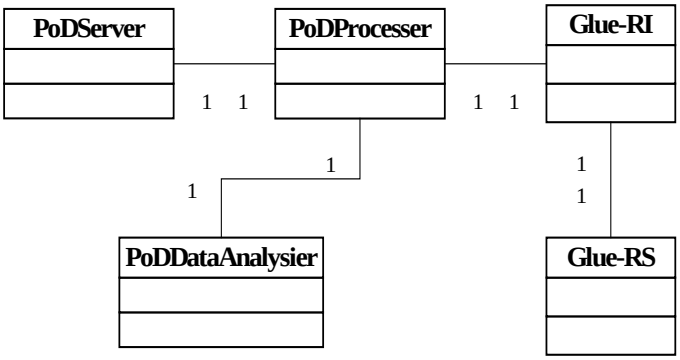


Figure 5.3: Class Diagram for the PoD and Glue

In order to avoid PoD server's overloading, the maximum connection number can be specified to restrain the number of connected clients. In fact, this is a very common way used by many other well known web servers like apache, otherwise the quality of service of the PoD is difficult to be guaranteed especially when system is running in a heavy mode.

5.6.5 Protocols

In order to notify clients of the current status of the PoD, two communication protocols have been defined. If the number of the current connections exceeds the limitation of the PoD, the PoD will send a "FULL" flag to the client. If the Glue cannot retrieve the back-end data or there is an exception raised by the back-end tool, the PoD will send an "EMPTY" flag to the client. All client implementations should focus on dealing with this flags as well as receiving the correct XML documents.

5.6.6 Configuration file

In general, the configuration file is a repository that contains all the information necessary to boot and run an application. The configuration file discussed here contains the configure information for the whole RMF. The PoD initially reads this configuration file and passes to the G-RI, which then hands over to the G-RS. The configuration includes three sections of information. One section includes the system information such as the name of the XML DTD schema. Another section contains the information to configure a PoD server such as the PoD description and location of SSL certificates. The final section contains the host and port information of the PoDRegistry. One of the advantages of the configuration file is that it allows the administrator to view the system specification; the other advantage is that the parameters should not be hard coded in the source code since the variable such as number of client, port number, etc. will be changed as the environment changes.

5.7 Security

This section describes the detailed application and implementation of SSL in Python with the support of third party libraries. One of the third party libraries is named OpenSSL that is actually a tool that can be used to generate self-signed certificate authentication, public and private keys for the communication parties using SSL. The other third party library is called Twisted that provides necessary functions used in SSL such as certificate verification and data encryption.

In contrast to single authentication, mutual authentication is preferred since it can allow both of server and client verify identities each other. Of course, mutual authentication is more complicated than the single authentication in terms of certificate verification and management.

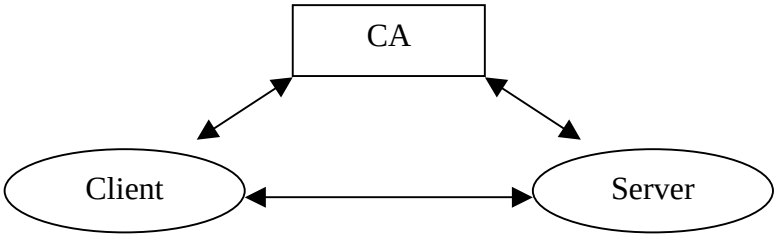


Figure 5.4: Mutual Authentication Architecture

This figure describes the architecture of mutual authentication and the relationship within three parties including CA, client and server that will be explained in the following. CA is a short of certificate authentication, which will sign certificates for communication parties by using their public keys and verifying their certificates at the beginning of SSL connection. In other words, the client will send server's certificate to CA for verification and vice versa as long as they get each other's certificate. Therefore, SSL connection can only be set up when both communication parties' certificates are verified successfully, otherwise an error indicating bad certificates will be notified to both communication party, and consequently the SSL connection between client and server will be closed.

5.7.1 Implementation

According to the client's requirements, data transmission over the public network should be encrypted since it is confidential. In addition, the client must ensure that the PoD could authenticate the clients before pushing any data and vice versa. Therefore, a SSL tunnel should be set up between the PoD and client. In the case of communication between PoD and PoDRegistry, again between client and PoDRegistry will not be necessary to use SSL since information between them is not considered confidential such as the location of PoD. CA currently resides in the server side so that it simplified the implementation significantly. Otherwise the whole system architecture will be more complicated since the CA in this project is a self-signed certificate authority and an official recognized CA centre is not necessary currently.

As described in the requirement session, this project will be written in Python. But the problem we met is that Python has no enough packages to support SSL. Therefore, the third party libraries have to be used to accomplish mutual authentication.

In fact, Python only supports SSL client side, whereas other third party libraries support SSL server side. Currently, there are four popular third-party libraries namely, M2Crypto, PyOpenSSL, TLSlite and Twisted. These libraries have different advantages and disadvantages since they are written by thousands of programmers all over the world. Most of the third party libraries do not provide specific and sufficient documents. Therefore, a comparison among them has to be done through testing the libraries one by one. Eventually, Twisted library seems more suitable to this project

since it has relative more documentation to explain the SSL interfaces and it is not very difficult to be implemented in the program either.

Unlike java key tool, another third party tool named OpenSSL is used to generate necessary certificates, public and private keys. OpenSSL is a toolkit which implements SSL v2, v3 and TLS protocols with full-strength cryptography worldwide. The whole process of using OpenSSL includes creating self-signed CA certificate, creating public and private keys and signing certificates. At the first place, a self-signed CA certificate will be created, which means that CAKey.pem and CACert.pem files are generated with x.509 option. Then a server key file named ServerKey.pem and a server certificate request file named ServerReq.pem will be created. The latter one will be signed to become a sever certificate file named ServerCert.pem by CA. Before signing certificates, the ServerReq.pem must be cat into a file named Server.pem for later certificate management and search. The same process happens in the client side. Eventually, both client and server will use its own private key file and certificate file for communicate with each other.

During the application processing, the server must indicate where CA is, so that CA can help client and server to verify certificates. Before SSL communication starts, both client and server must have their own private keys and certificates signed by CA based on their public keys.

Once the certificates have been verified successfully, a SSL connection will be started immediately between a client and a server. In order to make sure the transferred data have already been encrypted, both TCPdump and ethereal tools in Unix can be used to capture the data and see whether the data is encrypted or not.

5.7.2 Certificate management in OpenSSL

The Secure Socket Layer protocol was created by Netscape to ensure secure transactions between web servers and browsers. The protocol uses a third party, a Certificate Authority (CA), to identify one end or both ends of the transactions. The paragraph below details how CA works.

A certificate contains information about the owner of the certificate, such as e-mail address, owner's name, certificate usage, duration of validity, resource location or Distinguished Name (DN) which includes the Common Name (CN) (web site address or e-mail address depending of the usage) and the certificate ID of the person who certifies (signs) this information. It contains also the public key and finally a

hash to ensure that the certificate has not been tampered with. Usually the web browser or application has already loaded the root certificate of well known Certification Authorities (CA) or root CA Certificates. The CA maintains a list of all signed certificates as well as a list of revoked certificates. A certificate is insecure until it is signed, as only a signed certificate cannot be modified. Signing a certificate using itself, it is called a self-signed certificate. All root CA certificates are self-signed. A self-signed CA was created in this project in order to sign certificates for communication parties. A commercial PKI is alternative for self-signed CA that will be explained below.

The Public Key Infrastructure (PKI) is the software management system and database system that allows to sign certificate, to keep a list of revoked certificates and to distribute public key etc. In order to secure individual applications, a well-known commercial PKI as their root CA certificate is most likely to be inside a browser or an application. The problem is that it is so expensive.

At the moment there are two choices between a commercial PKI and our own PKI. The commercial PKI were created at the beginning to enable secure commerce over the Internet, basically securing HTTP. The pricing of certificates was calculated on a per host basis. The cost is more expensive than that for a domain name because of the costs to identify the owner of the certificate. Unfortunately when this project needs thousands of certificates, costs start to skyrocket as well as the administrative burden to register all these certificates to the Certificate Authority increases dramatically. Therefore, building our own Certificate Authority is preferred at the moment although it has certain limitations for users. This allows flexible management of certificates. In addition, a global PKI will be expected in the future.

5.8 PoDRegistry

As described in requirements, the PoDRegistry as a registry server accepts the registry information from PoDs which may reside anywhere. Therefore, clients can then contact the PoDRegistry to obtain the registered PoDs' profiles. In other words, the PoDRegistry will act as an agent that can also provide clients a very convenient way to find their preferred PoD servers. In order to achieve this, the PoDRegistry has to store PoD servers' information into a log file currently. Furthermore, the registry information can also be saved into MySQL database since the necessary tables have

been drawn, and in the future standard SQL statements can be used to retrieve data from the large tables.

5.8.1 Implementation

The current implementation of the PoDRegistry is written in Python. In addition, there are two additional communication protocols between the PoDserver and the PoDRegistry, again between clients and PoDRegistry have been done so that the data can be analysed correctly in both ends.

The PoDRegistry runs on well-known port 5555 when it starts. When a PoD server starts, it connects to the PoDRegistry and registers its server type, server hostname, server socket number and a short server description that indicates what the server does. In detail, the PoDRegistry accepts connections from both clients and PoD servers using the same socket. That means it is up to the client and PoD server to tell the PoDRegistry the identification through adding connection type in the data packet. For instance, in the first element of registry information of the PoD, it will store information like “I am a server”, so that the PoDRegistry can know that a server wants to register. However, it is not allowed for one PoD server to register twice in the same PoDRegistey. Therefore, before adding new PoDRegistry information into the log file, the PoDRegistry will check whether the PoD server has already registered. If yes the server will not be added. Meanwhile, the PoDaRegistry can accept any client’s connection since the PoDRegistry has started. Therefore, when a client connects to the registry server, the client will tell the registry server whom he or she is by using the same way as server such as “I am a client and I want to find a PoD server”. Then the registry server will know that a client wants to collect PoD server information, and consequently send the PoD server profiles to the client.

Another important technique that the program uses is that the PoDRegistry spawns a new thread to detect the disconnected links with PoD servers timely, and consequently updates the registry information in log files. In detail, the separate thread tries to send idle message to the PoD servers timely and catch exceptions whenever the idle message is not sent successfully. Then, the PoDRegistry will know the disconnection status while the PoD server cannot accept message or has closed the connection. However, it is so difficult to locate the reason why the PoD server disconnects from the registry server since Python is a high-level programming language although we can guess some possible reasons such as a network crash or

computer restart. Whereas in low level programming language C, the error can be located by matching the socket result value. Due to the uncertain disconnection reasons, a PoD deregister is not developed. Instead, PoDResgitry will delete the dedicated PoD server information in the log file when a thread detects a disconnected TCP link. The thread repeats this process for all the entries in the registry log file and then goes to sleep for a constant number of milliseconds, currently set at 1000. Of course, the time interval can be changed to be any value based on the real system. The separate thread performs this operation as long as the PoDRegistry is running.

Unlike the connection between PoD server and PoDRegistry, the connection between client and PoDRegistry is not necessary to be maintained after finish transmitting data. This is simply because that client will normally only connect to PoDRegistry only to get PoD information. In other words, the connection between client and the PoDRegistry will be closed when information is transmitted successfully.

5.8.2 Protocols

The PoDRegistry protocol is designed to provide communication over a network between the different components defined in the system architecture. This PoDRegistry protocol here is served for clients, the PoDRegistry and a PoD server. Therefore, it includes two protocols, one is for the PoDRegisetry and a PoD server named RP protocol, and the other is for the PoDRegisetry and clients named RC protocol. Both RP and RC are simple and convenient communication protocols.

Field description	Field type
Protocol version ID = 1	Integer
Identify message	String
PoD type	String
Socket number	Integer
Host name	String
Socket description	String

Table 5.6: RP Protocol

Field description	Field type
Protocol version ID = 1	Integer
Identify message	String
Searching key	String

Table 5.7: RC Protocol

5.9 Compiler

5.9.1 Architecture

Generally speaking, a compiler is a computing tool that reads a program written in a first (source) language and translates it into second (target) language, which is closer to the machine architecture; it is often assembler or some machine-oriented language. The compilation usually involves four stages: lexical analysis, parsing, semantic analysis, and target language code generation. The purpose of compiler in the toolkit is to generate the user-defined types and the common code of system components, which include the code for the Glue and PoD. The following diagram explains the architecture of this compiler:

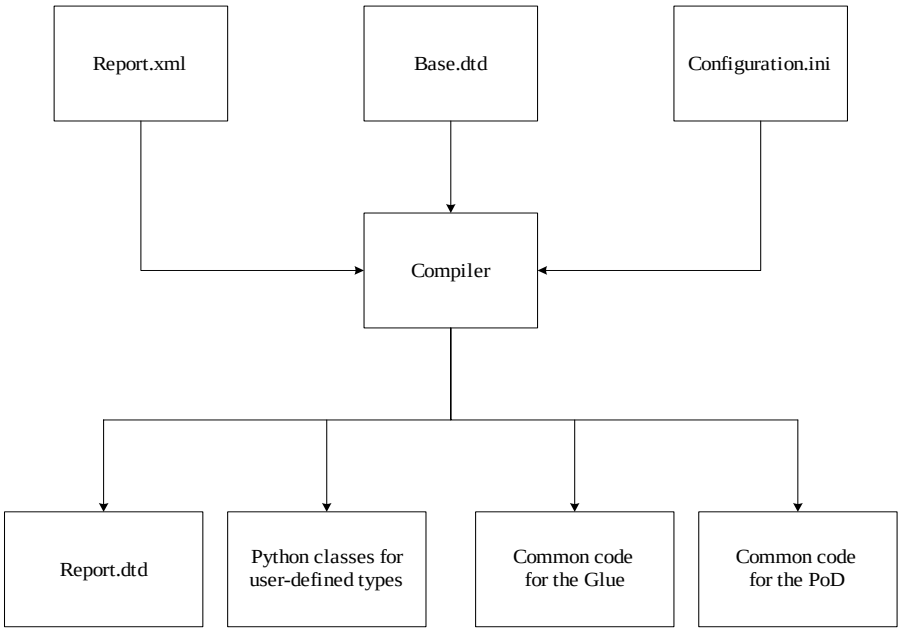


Figure 5.5: Compiler Architecture

There are three input files Report.xml, Base.dtd, and Configuration.ini for the compiler. Report.xml is a XML file that acts as an ExSERT schema containing the definition of the information format transferred across the network. The schema consists of a number of user-defined types, which may be composed of other user-defined types or basic data types. Base.dtd is a DTD file that contains the basic data types. The implementation of basic data types is stored in the package of the API which will be discussed in the following section. Configuration.ini is the configuration file, in which the user can set the location of the Report.xml, Base.dtd,

and the package of basic data types. The user can also define the name of a generated monitoring tool in this configuration file.

There are four types of output files as shown in the diagram. Report.dtd is a DTD file that will be used for the validation purpose by both the PoD and Client. Report.dtd is constructed in the way that the relationship of the elements is taken from the Report.xml and the types of the elements are obtained from Base.dtd. Python classes for the user-defined types are generated according to the generated Report.dtd. Before creating these classes, a checking on the Base.dtd is performed since Report.dtd also includes the basic data types which do not need to be generated. Common code for the Glue contains a complete generation of G-RI part. The implementation for G-RS would not be generated by the compiler since the output from diverse back-end tools is different, requiring specific processing. Most implementations of the PoD are generated by the compiler except for the data analysing part.

5.9.2 ExSERT Schema

There are a number of reasons why XML has been chosen for the specification of input ExSERT schema. One obvious reason is that it directly mapped the transferring syntax described in Section 5.1; thus it provides an overview of the report format. Subsequently, XML has a very simple syntax, which can be easily handled by the user when creating the ExSERT schema.

To further ease the creation of the ExSERT schema, a graphical development environment could be used for defining this input. For instance, GUI components like “textbox” could be applied to represent an XML tag. Drag and drop operations could be adopted for manipulating elements within the GUI. Due to the time constraint, this graphical environment has not implemented in this toolkit, but it can be potentially useful when the toolkit has more users.

5.9.3 Implementation

The iterative and incremental development method has been adopted for the implementation of the compiler. As it is quite difficult to determine the common code for the system components, it is not easy to find out the correct target for the compiler at one time. The initial version of the compiler is implemented based on the common code derived from the example RMF, which is implemented prior to the

compiler based on the system prototype defined by the last year group. Every time a new type of RMF was developed, the common code is revised, and then the compiler is refined in order to meet this change. Sometimes, the compiler also needs to be refined when the implementation of the system components has modified.

5.9.4 Directory structure

The compiled files are structured in a way that the user can easily find it and make modifications. The following diagram shows the directory structure of generated files:

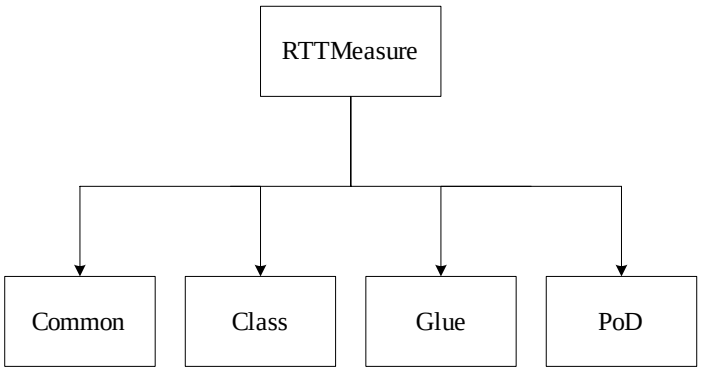


Figure 5.6: Directory Structure of Generated Files

RTTMeasure is the root of all compiled files, and it is also the name of the tool specified in the Configuration.ini. RTTMeasure contains four sub folders which are Common, Class, Glue, and PoD. Common folder includes Report.dtd and Configuration.ini which are shared between the PoD Server and Client. Class folder contains the Python implementation for the user-defined types. Glue and PoD folders contain the implementation of the common Glue and PoD source files respectively.

5.10 API

The API contains two packages: Class that includes the implementation of all basic data types and Function that includes functions which allow the developer to add the functionality of the PoD. The implementation of the data types have been discussed in Section 5.3, and this section mainly concentrates on the discussion of provided functions in details. The following figure shows the structure of the package:

Lib/			Top-level package
	__init__.py		Initialise the Lib package
	Class/		Sub package for basic data types
		__init__.py	
		Integer.py	
		String.py	
		Year.py	
		Month.py	
		Day.py	
		Hour.py	
		Minute.py	
		Second.py	
		IPv4Address.py	
		IPv6Address.py	
		MACAddress.py	
		RTT.py	
		NumOfBytes.py	
		PacketLoss.py	
	Function/		Sub package for functions
		__init__.py	Initialise the Function package
		ThresholdRTT.py	
		LinkStatus.py	
		Utilization.py	
		StatisticRTT.py	
		BinaryEncoding.py	

Figure 5.7: Package Structure

The "__init__.py" files are required to make Python treat the directories as containing packages.

5.10.1 ThresholdRTT

The purpose of this function is to provide the user to create a RMF that monitors the network's health in terms of the status of round-trip times (RTTs). A number of thresholds between different status zones can be defined, which makes the RTTs fall into these status zones. As far as the RMF developed by last year's group, four thresholds exist, making RTTs fall into one of four categories: "INFO", "WARNING", "ERROR" and "CRITICAL". One refinement has been made compared to the function in the ThresholdRTT PoD developed by the previous group. The improvement is that the new function allows the thresholds to be configurable, which means the user can specify the value of thresholds, whereas there are four

static threshold values in the previous RMF. This improvement provides a very general and flexible way to set up the thresholds that might not be fixed in different IXPs. For example, 60 milliseconds may fall into the “INFO” zone at one IXP, but may fall into the “ERROR” zone at another one. The threshold values can be defined in the RMF configuration file, and the following code shows how it can be done:

```
threshold : 500, 1000, 1500, 2000
```

Threshold values are defined as a string separated with commas, and this string will be passed as a parameter to the function.

When this function is called in the PoD, an unprocessed XML file is also passed as a parameter. In fact, all the functions discussed here take a raw XML as the input. As discussed before, the XML file is created by the G-RI. After this XML file has been parsed, the function takes each RTT value and assigns a dedicated status to it based on its value, and then a XML document is returned. The following code block shows the processed XML document with threshold for each RTT:

```
<TextToXML>
    <RTTMeasure>
        <Timestamp>
            <Date>
                <Year>2005</Year>
                <Month>08</Month>
                <Day>15</Day>
            </Date>
            <Time>
                <Hour>14</Hour>
                <Minute>25</Minute>
                <Second>30</Second>
            </Time>
        </Timestamp>
        <IPv4Address>66.102.7.104</IPv4Address>
        <NumOfBytes>84</NumOfBytes>
        <RTT>450</RTT>
        <Threshold>INFO</Threshold>
    </RTTMeasure>
    <RTTMeasure>
        <Timestamp>
            <Date>
                <Year>2005</Year>
                <Month>08</Month>
                <Day>15</Day>
            </Date>
```

```
            <Time>
                <Hour>14</Hour>
                <Minute>25</Minute>
                <Second>31</Second>
            </Time>
        </Timestamp>
        <IPv4Address>66.102.7.104</IPv4Address>
        <NumOfBytes>84</NumOfBytes>
        <RTT>800</RTT>
        <Threshold>WARNING</Threshold>
    </RTTMeasure>
</TextToXML>
```

5.10.2 LinkStatus

The purpose of this function is to provide the user to create a RMF that monitors the status of each interface. This function is built on top of the ThresholdRTT function, but instead of assigning a status to each RTT, it counts the number of each status and shows the percentage of different statuses for a particular interface. For example, for a given interface, a link status looks like:

INFO50.00% WARNING50.00% ERROR0.00% CRITICAL0.00%

When “ERROR” or “CRITICAL” zones take a large percentage, it means the current network performance is poor, and consequently, the network administrator needs to pay attention to this measurement. As the back-end can generate RTTs for a number of interfaces constantly, the function can also deal with multiple interfaces at the same time. The user can specify how often to refresh the link status information in the RMF configuration file according their needs.

Like ThresholdRTT function, this function also takes an XML file and a threshold string as inputs, and returns a new XML document with link status. The following code block shows a processed XML document. This link status is calculated every five round trip times:

```
<TextToXML>
    <LinkMeasure>
        <IPv4Address>66.102.7.104</IPv4Address>
        <RTT>300</RTT>
        <LinkStatus>
            INFO60.00%WARNING0.00%ERROR50.00%CRITICAL0.00%
        </LinkStatus>
```

```
        </LinkMeasure>
        <LinkMeasure>
            <IPv4Address>66.102.7.104</IPv4Address>
            <RTT>350</RTT>
            <LinkStatus>
                INFO60.00%WARNING0.00%ERROR50.00%CRITICAL0.00%
            </LinkStatus>
        </LinkMeasure>
        <LinkMeasure>
            <IPv4Address>66.102.7.104</IPv4Address>
            <RTT>400</RTT>
            <LinkStatus>
                INFO60.00%WARNING20.00%ERROR20.00%CRITICAL0.00%
            </LinkStatus>
        </LinkMeasure>
        <LinkMeasure>
            <IPv4Address>66.102.7.104</IPv4Address>
            <RTT>600</RTT>
            <LinkStatus>
                INFO60.00%WARNING20.00%ERROR20.00%CRITICAL0.00%
            </LinkStatus>
        </LinkMeasure>
        <LinkMeasure>
            <IPv4Address>66.102.7.104</IPv4Address>
            <RTT>1200</RTT>
            <LinkStatus>
                INFO60.00%WARNING20.00%ERROR20.00%CRITICAL0.00%
            </LinkStatus>
        </LinkMeasure>
</TextToXML>
```

5.10.3 LinkUtilization

The purpose of this function is to provide the user to create a RMF that monitors the utilization of each interface. Again, the link utilization is another parameter for measuring network performance. The link utilization is calculated from the packet loss rate over a period of time, and the packet loss rate can be generated by the back-end tool such as fping. For a given link, if a remote switch or router is monitored by the fping tool ten times and packet loss happens three times, no matter how many packets lost each time, the utilization is 70% for this link. Similar to LinkStatus function, this function can also deal with multiple interfaces, and the refresh time can be specified in the RMF configuration file as well.

Unlike the previous two functions, this one only takes an XML as input, and returns a new XML as output. The following code block processed one XML document, and this link utilization is calculated based on five packet losses:

```
<TextToXML>
    <LinkMeasure>
        <IPv4Address>66.102.7.104</IPv4Address>
        <PacketLoss>0%</PacketLoss>
        <LinkUtilization>80%< /LinkUtilization>
    </LinkMeasure>
    <LinkMeasure>
        <IPv4Address>66.102.7.104</IPv4Address>
        <PacketLoss>0%</PacketLoss>
        <LinkUtilization>80%< /LinkUtilization>
    </LinkMeasure>
    <LinkMeasure>
        <IPv4Address>66.102.7.104</IPv4Address>
        <PacketLoss>20%</PacketLoss>
        <LinkUtilization>80%< /LinkUtilization>
    </LinkMeasure>
    <LinkMeasure>
        <IPv4Address>66.102.7.104</IPv4Address>
        <PacketLoss>0%</PacketLoss>
        <LinkUtilization>80%< /LinkUtilization>
    </LinkMeasure>
    <LinkMeasure>
        <IPv4Address>66.102.7.104</IPv4Address>
        <PacketLoss>0%</PacketLoss>
        <LinkUtilization>80%< /LinkUtilization>
    </LinkMeasure>
</TextToXML>
```

5.10.4 StatisticRTT

The purpose of this function is to provide the user to create a RMF that provides statistic measurements including maximum, minimum and average of RTTs. This function is very useful because it gives the user the range of RTT values, alerting the administrator that there might be a network problem.

This function only takes an XML as input, and returns a new XML with maximum, minimum, and average of RTTs for each RTT. The following code block shows the processed XML document:

```
<TextToXML>
    <StatisticRTT>
        <IPv4Address>66.102.7.104</IPv4Address>
```

```
            <RTT>500</RTT>
            <MinRTT>500< /MinRTT>
            <MaxRTT>700</MaxRTT>
            <AvgRTT>600</AvgRTT>
        </StatisticRTT >
        <StatisticRTT >
            <IPv4Address>66.102.7.104</IPv4Address>
            <RTT>550</RTT>
            <MinRTT>500< /MinRTT>
            <MaxRTT>700</MaxRTT>
            <AvgRTT>600</AvgRTT>
        </StatisticRTT >
        <StatisticRTT >
            <IPv4Address>66.102.7.104</IPv4Address>
            <RTT>600</RTT>
            <MinRTT>500< /MinRTT>
            <MaxRTT>700</MaxRTT>
            <AvgRTT>600</AvgRTT>
        </StatisticRTT >
        <StatisticRTT >
            <IPv4Address>66.102.7.104</IPv4Address>
            <RTT>650</RTT>
            <MinRTT>500< /MinRTT>
            <MaxRTT>700</MaxRTT>
            <AvgRTT>600</AvgRTT>
        </StatisticRTT >
        <StatisticRTT >
            <IPv4Address>66.102.7.104</IPv4Address>
            <RTT>700</RTT>
            <MinRTT>500< /MinRTT>
            <MaxRTT>700</MaxRTT>
            <AvgRTT>600</AvgRTT>
        </StatisticRTT >
</TextToXML>
```

5.11 Database

This section describes the detailed database design that will be beneficial for future data analysis and processing. Given the operating system platform and data flow size, MySQL database was chosen since it has following advantages.

Firstly, MySQL database server is the world's most popular open source database. Secondly, Python provides plenty of libraries to support operations on MySQL database. Therefore, it is a better combination between Python and MySQL. In addition, MySQL is an attractive alternative to higher-cost, more complex database technology. Its award-winning speed, scalability and reliability make it the right choice for the current medium-size project.

5.11.1 Table design

This table is used to store data retrieved from XML file. The reason to save those data is to make it as historic data and examples that will be necessary for future statistic processing using complex mathematic algorithms.

FPING TABLE:

Field	Type	Null	Key	Default	Extra
YEAR	INT(11)	YES		NULL	
MONTH	INT(11)	YES		NULL	
DAY	INT(11)	YES		NULL	
HOUR	INT(11)	YES		NULL	
MINUTE	INT(11)	YES		NULL	
SECOND	INT(11)	YES		NULL	
NUMBER_OF_BYTES	INT(11)	YES		NULL	
IP_V4_ADDRESS	VARCHAR(40)	YES		NULL	
RTT	INT(11)	YES		NULL	
THRESHOLD	VARCHAR(10)	YES		NULL	
PACKETLOSS	VARCHAR(10)	YES		NULL	
UTILIZATION	VARCHAR(10)	YES		NULL	
MINRTT	INT(11)	YES		NULL	
AVGRTT	INT(11)	YES		NULL	
MAXRTT	INT(11)	YES		NULL	

This table is used to save registry information from PoD servers. Therefore, the function of searching PoD information can be implemented by using a searching key such as PoD address or PoD type in a SQL statement. This will significantly improve the efficiency in terms of thousands of PoD records in the database.

POD_REGISTRY TABLE:

Field	Type	Null	Key	Default	Extra
POD_TYPE	VARCHAR(10)	YES		NULL	
POD_IP_ADDRESS	VARCHAR(50)				
POD_SOCKET_PORT	INT(11)			0	
POD_DESCRIPTION	VARCHAR(50)	YES		NULL	

In the case of certificate management, database is an easy and effective mechanism since it is capable of providing account authority mechanism and certificate search or modification service by using SQL statement.

SSL_CA:

Field	Type	Null	Key	Default	Extra
NAME	VARCHAR(20)	YES		NULL	
CERT_DIR	VARCHAR(50)				
CERT_TYPE	VARCHAR(10)				
EXPIRE_DATE	DATE				
DESCRIPTION	VARCHAR(50)	YES		NULL	

SSL_CLIENT:

Field	Type	Null	Key	Default	Extra
NAME	VARCHAR(20)	YES		NULL	
PRIVATE_KEY_DIR	VARCHAR(50)				
CERT_DIR	VARCHAR(50)				
CERT_TYPE	VARCHAR(10)				
EXPIRE_DATE	DATE				
DESCRIPTION	VARCHAR(50)	YES		NULL	

SSL_SERVER:

Field	Type	Null	Key	Default	Extra
NAME	VARCHAR(20)	YES		NULL	
PRIVATE_KEY_DIR	VARCHAR(50)				
CERT_DIR	VARCHAR(50)				
CERT_TYPE	VARCHAR(10)				
EXPIRE_DATE	DATE				
DESCRIPTION	VARCHAR(50)	YES		NULL	

Chapter 6 Testing

This chapter details the type of testing that was carried out on the system. A test document has been produced which contains a selection of test cases, and the expected outcome and the actual outcome of the tests. The test document is included in a separate file.

6.1 Test Strategy

There are a number of techniques available that are used to discover program faults and to show that system meets its requirements.

One form of testing is defect testing. Sommerville (2001) states “The goal of defect testing is to expose latent defects in a software system before the system is delivered.” The emphasis of defect testing is on showing the presence of program faults and not the absence.

Sommerville (2001) suggests that exhaustive testing, which is “where every possible program execution sequence is tested” is impractical. Therefore testing of a subset of cases is needed. This is effectively functional testing or black box testing. With this form of testing the tester is only concerned with the functionality of the software. The inputs and related outputs are studied. According to (Sommerville 2001) black box testing can be applied just as well to systems that are organized as functions or as objects. The functional or black box testing approach will suit the testing of the system because the system has functional features and the system is organized as class, which is the model from which an object is created. As a result, black box testing will be one of the techniques used to test the system.

As stated above black box testing is carried out before the system is delivered. However, as the system was being developed the author has already carried out testing. This was done after the completion of every class. Sommerville (2001) writes, “Programmers take responsibility for testing their own code”. These tested components were then integrated to build the larger system. This is known as integration testing. The integration testing was based on the use. After completion of these classes they were integrated together to complete the use case. This effectively creates a module to test.

Once this module was tested the next use case was analyzed resulting in another module to be tested. Sommerville (2001) states this forms an incremental

approach to system integration and testing. The purpose of this incremental approach is that it is easier to locate errors. This is desirable because due to the interactions between system components it is sometimes difficult to find the error. By using the incremental approach this difficulty will be minimized.

Exceptional case test includes exceptional conditions that occur in a system that are not expected or are not a part of normal system operation. When the system handles these exceptional conditions improperly, it can lead to failures and system crashes. Exception failures are estimated to cause two thirds of system crashes and fifty percent of computer system security vulnerabilities. Exception handling is especially important in embedded and real-time computer systems because software in these systems cannot easily be fixed or replaced, and they must deal with the unpredictability of the real world. Robust exception handling in software can improve software fault tolerance and fault avoidance, but no structured techniques exist for implementing dependable exception handling. However, many exceptional conditions can be anticipated when the system is designed, and protection against these conditions can be incorporated into the system. Traditional software engineering techniques such as code walkthroughs and software testing can illuminate more exceptional conditions to be caught, such as bad input for functions and memory and data errors. However, it is impossible to cover all exceptional cases. It is also difficult to design a dependable system that can tolerate truly unexpected conditions. In these cases, some form of graceful degradation is necessary to safely bring down the system without causing major hazards.

The last but not least test case is the performance test. The main purpose of performance test is to reduce the application response time without incurring additional hardware costs, to improve customer satisfaction due to prompt response of the organization's application, and to fix bottlenecks before costly problems occur in production. Load and Stress Testing services include testing an application for normal load, heavy load (stress), sudden increase in load (spike) and sustained load (endurance). Monitors are deployed on all the systems being tested and vital data is collected during these tests. We analyse the data collected to identify the performance bottlenecks.

6.2 Test Case Summary

This section details a selection of test cases that were carried out on the system. Please refer to the testing manual for each test case, the expected outcomes and actual outcomes.

6.2.1 Resources involved

We developed and run our program on Red Hat Enterprise Linux WS software related:

Python version 2.3.3 used for coding

CVS: Used for version management

Besides the basic library in the installed Python, there are some extra library needed, such as OpenSSL, Twisted.Python, Twisted.internet, wisted.internet.protocol, MySQL,etc.

6.2.2 Single model/function test

Every single function, such as Fping, write file, read file, output user interface, TCP connection, TCP disconnection and error communication etc, should have results printed on screen.

For example, one unit test is to test the single function of PoDRegistry server. The output below shows the situation while the PoD server connects to the PoDRegistry server.

```
[pliang@skinner PoDRegistry]$ Python PoDRegistry.py
PoDRegistry server is waiting for connections...
Closed-connection-detection thread is running...
Connected by ('127.0.0.1', 1289)
A PoD server has connected.
PoD server info has been saved.
PoDServer(localhost, 5000) has closed its connection.
```

6.2.3 Whole model integration test

The purpose of this test is to integrate all functions included in the programs and run the whole programs by using different input.

In order to test whether the whole programs can work smoothly or not, we have made a client interface to get the final transmitted data.

Link Status Measurement

Year	Month	Day	Hour	Minute	Second	IPv4Address	NumberOfBytes	RTT	LinkStatus
2005	06	22	13	58	18	127.0.0.1	84	90	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	18	66.249.93.99	84	30000	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	19	66.249.93.99	84	29800	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	19	127.0.0.1	84	50	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	20	66.249.93.99	84	30000	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	20	127.0.0.1	84	50	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	21	66.249.93.99	84	29900	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	21	127.0.0.1	84	50	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	22	66.249.93.99	84	30000	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%
2005	06	22	13	58	22	127.0.0.1	84	90	INFO:99.99% WARNING:0.00% ERROR:0.00% CRITICAL:0.00%

Figure 6.1: Client Output

6.2.4 Exceptional case test

The exceptional cases are mainly happened during the interaction with applications/files out of programs and network communication. Although it is impossible to catch all the exceptions, we still need to check the exception messages in order to make our program more users friendly.

6.2.5 Stress and performance test

Based on the unit test and integration test, stress and performance test check the system performance while making system load as heavy as possible.

Chapter 7 Project Management

7.1 Project Scope

The scope for this particular project has roughly been defined in previous chapter of this report, which introduced the objectives. In this chapter, we detail the scope in the view of project management in order to make project assignment and tasks more clear.

Internet Exchange Points (IXPs) have been developing network-monitoring toolkits that are specifically suited to their needs, requirements and infrastructure. However, this approach has rarely been possible for one IXP to use a monitoring tool developed by another IXP. Many IXPs have similar monitoring requirement, and have semantically similar tools. The tools often differ in the presentation of information, such as different data structures, different coding for processing data and logs, and different visualisation front ends, etc. IXPs recognize that this is a growing problem in the aspects such as sharing information directly, making comparison of multi-site data, using common information for troubleshooting, and performing analysis using multi-site data. The last year's LINX network-monitoring project, conducted with the collaboration of the London Internet Exchange Point, addressed precisely to this problem.

The reason why we will continue to do this project is to refine the system, to improve coding efficiency by using code auto-generation and to increase more useful and dedicated functions in order to make our achievements more valuable and realistic for the LINX and other IXPs.

This project is mainly the re-engineering and further developments based on the last year's project. The importance of the last year's project is that they developed a new system architecture on which future network monitoring tools can be based, so that new tools can be deployed on Internet Exchange Points (IXPs) which have different hardware and software infrastructures. The following extensions will be beneficial from the underlying architecture, which has been proved and achieved.

First of all, the project needs to be recapped by the scripting language Python rather than previous Java or C++. This takes fully advantages of Python namely, powerful, neat and independent. Like most scripting languages, Python features

excellent text and file manipulation capabilities. Yet, unlike most scripting languages, Python sports a powerful object-oriented environment with a robust platform API for network programming, threads, and graphical user interface development. It can be extended with components written in C and C++ with ease, allowing it to be connected to most existing libraries. To top it off, Python has been shown to be more portable than other popular interpreted languages, running comfortably on platforms ranging from massive parallel connection machines to personal digital assistants and other embedded systems. The efficiency of Python programming greatly saves the time of coding.

Second, this project made a compiler that takes basic input – XML file and generates auto Python codes. Since the heterogeneous raw data gained by using different network commands, such as SNMP, Fping etc have different format, we can convert those disparate raw data into a common form through using this tool regardless of the underlying hardware and software infrastructure. This also means it will make crossing IXPs applications easier than before. In order to achieve this, an XML and XML schema is necessary. And Python is an excellent choice for XML programming and distributed application development. The creation of compiler effectively save the time of future programmer while there is a new requirement

Third, Binary encoding was added into this year's project. While XML solves the problem of data heterogeneity, XML's processing overhead, storage requirement, and bandwidth consumption become quite problematic when transaction volumes are high. Our approach for improving the performance of XML is to compress XML directly. Although compression may solve the bandwidth issue in the most straightforward approach to reducing the size of XML messages, it worsens the processing problem at both sides of the sender and recipient to apply compression technologies. Furthermore, compression formats like zip or base64 offer an "all or nothing" approach. So any marginal gains in network bandwidth are also lost in processing time. To resolve the limitations of all-or-nothing compression and its processing overhead, we developed the binary representation of XML. This encoding uses binary, rather than text-based, means for serializing and transmitting XML information. It promises to significantly alter the processing, bandwidth, and storage penalties that currently plague XML.

Forth, in order to make network controlling more secure and easy to be managed, the registry PoD is needed to make itself as a registry server which can let

different remote monitors to watch the network as long as they can pass the secure checking of the registry server. This registry POD can provide necessary information for the registered clients, such as the IP address and network port number.

The last but not least function is Secure Socket Layer (SSL). The network communication between PoD server and clients should be as secure as possible. SSL uses the public-and-private key encryption system from RSA, which also includes the use of a digital certificate. When an SSL session is started, the server sends its public key to the user's browser, which the client uses to send a randomly generated private key back to the server in order to have a secret key exchange for that session. We use database to store the certificates of clients this year.

Finally, this project will discuss future work that could be carried out to improve on the achievements of this project due to there have been potential valuable and as yet underdeveloped idea.

7.2 Team Organization and Communication Plan

Every Tuesday there is a project meeting with our supervisor. Before and after the meeting the meeting agenda and meeting minutes will be sent by project manager respectively in order to make project topics and weekly work plan more organized.

In general the group met and worked together on a daily basis. Part of the planning phase of this project was to design and agree on a weekly work schedule, which roughly outlined the working hours as well as what general tasks needed to be accomplished in the week. The work break down was planed just after the weekly project meeting. So the tasks were made clear and performed by assigning one or two owners according to group member’s interest and experience. In addition, the schedule called for a stop on every Wednesday or Thursday while the respected owners do have issues to complete his or her tasks. After the first two weeks’ cooperation, each member in the group had a rough learning about each member’s strength and weakness, so the strategy called work in pair was adopted. The pair programming not only makes people learn from each other more conveniently, but also is very helpful to facilitate the schedules while one member is absent.

The communication plan includes Emails of general information such as function requirement and specific instructions forthcoming, web blob of meeting agenda and minutes and MSN messenger, etc.

The weekly project meeting provided the basic working guidelines for the team and resolved the issues with the help of supervisors. The first half time of the meeting is mainly for weekly work review and issue resolution, and the second half time of the meeting is for scheduling new weekly tasks. The web blob (http://spaces.msn.com/members/dcnds-groupB) records every meeting agenda and minutes, which provides the basic information of work progress.

7.3 Project Schedule and Progress

The ideal project goes through a short initial development phase, followed by years of simultaneous production support and refinement, and finally graceful retirement when the project no longer makes sense. For our project, the lifecycle does not include the future refinement and maintenance because of the time constraint. Basically our project lifecycle includes 5 phases, investigation, initial simple prototype, complete coding, testing and documentation.

Phase 1: Investigation and learning stage, around two weeks

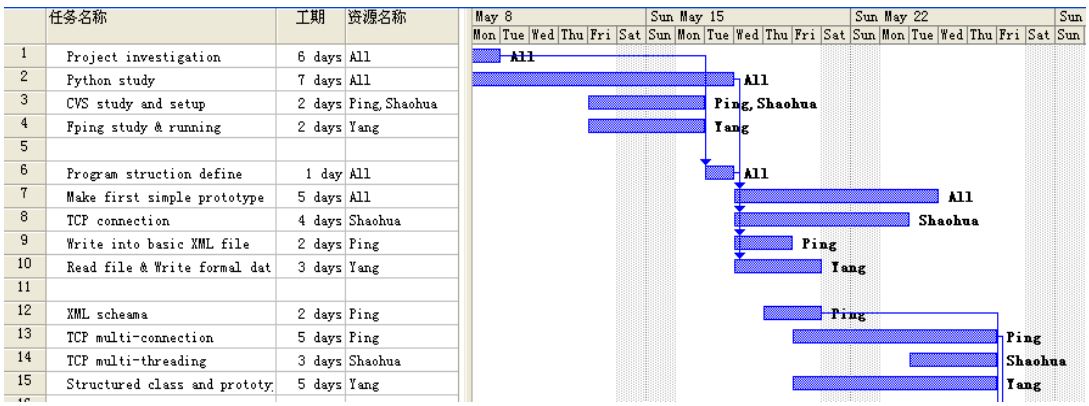
Phase 2: First simple proto type, around two weeks

Phase 3: Develop structured program and unit test, around one and a half month

Phase 4: Integration test, around two weeks

Phase 5: Documentation, around one month

In every phase, we have detailed work break down and schedules, which shows the progress of our work.



	任务名称	工期	资源名称
1	Project investigation	6 days	All
2	Python study	7 days	All
3	CVS study and setup	2 days	Ping,Shaohua
4	Fping study & running	2 days	Yang
5			
6	Program struction define	1 day	All
7	Make first simple prototype	5 days	All
8	TCP connection	4 days	Shaohua
9	Write into basic XML file	2 days	Ping
10	Read file & Write formal dat	3 days	Yang
11			
12	XML scheama	2 days	Ping
13	TCP multi-connection	5 days	Ping
14	TCP multi-threading	3 days	Shaohua
15	Structured class and prototy	5 days	Yang

Figure 7.1: Phase 1 and Part of Phase 2

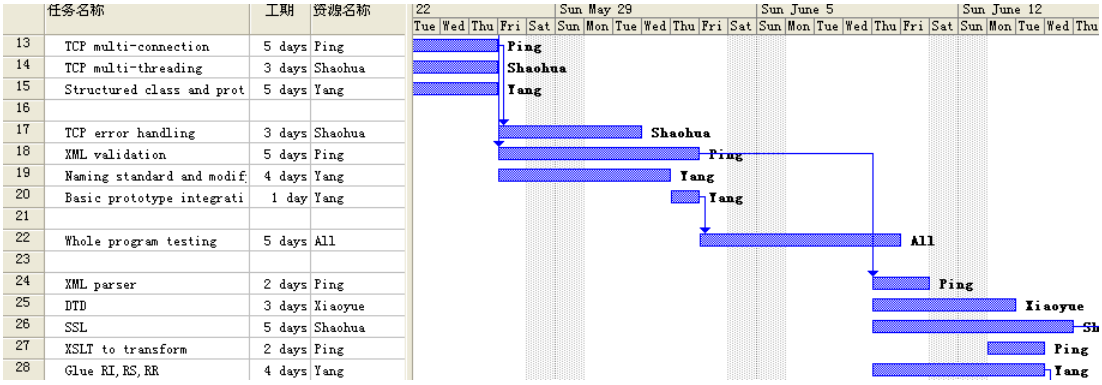


Figure 7.2: Phase 2 and Part of Phase 3

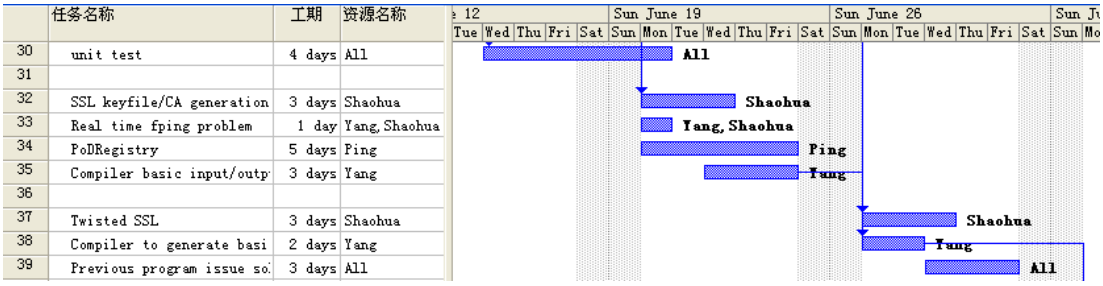


Figure 7.3: Phase 3

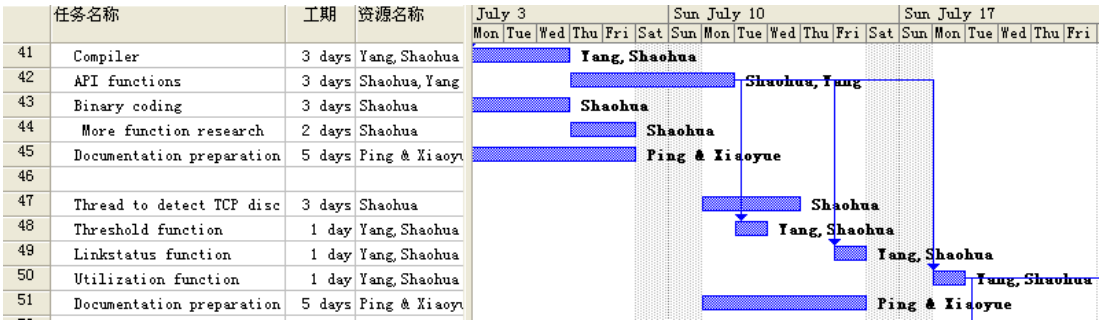


Figure 7.4: Part of Phase 4

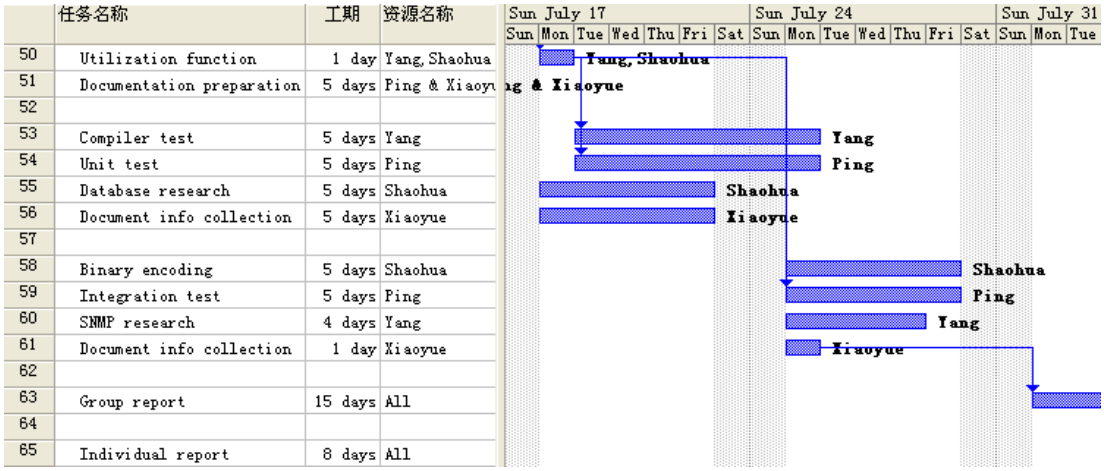


Figure 7.5: Part of Phase 4 and Phase 5

Milestones:

- ※ 2nd of May: Kick off meeting with supervisor
- ※ 11th of May: Project management presentation.
- ※ 3rd of June: Fist simple prototype complete.
- ※ 30th of June: Structured prototype complete
- ※ 29th of July: Coding and test complete
- ※ 1st of August: Begin documentation and report.
- ※ 19th of August: Complete group documentation.
- ※ 31st of August: Complete individual documentation.
- ※ 2nd of Sep: Project submission

This process results in a project schedule that can be tracked and monitored. The information identifies what resources are needed, when the resources are needed and for how long. It defines the timeframes and dates for key project deliverables and for project completion. It sets expectations for project progress. Work is broken down into small, more manageable pieces and reduces the overall complexity of the project. Creating a project schedule will also provide a tool for performing critical path assessments and more effective analysis of problem areas. Without a project schedule, the project manager won't know all that must be completed, who needs to complete it, how to make effective adjustments to get things completed, nor when to expect the project to be complete.

7.4 Risk Management

Throughout the project life cycle, we have to face several potential risks which might impact the success of this project, and several strategies were adopted to minimize risks. Since this project was dealing with a real client, it was apparent that many risks would be directly associated with the client. This chapter gives an explanation of the project risks identified and the actions taken to address them.

Each risk was classified to have low, medium or high seriousness and likelihood. A few of these risks are in the table below:

Risks	Probability	Impact
No enough testing/monitoring resource	M	H
Misunderstanding of customer requirement /	M	H
Demo not ready at deadline	M	H
Continuing stream of requirements changes	M	H
Real-time performance shortfalls	H	H
Unrealistic schedules and budgets	L	H
Functions and properties of application not precise	M	M
User interface not handy	M	M
Fail unit-testing repairing	M	M
Version control confusion	H	M

Table 7.1: Risks Management

After tacking and measuring risks, we need to take actions to mitigate risks. The Extreme Programming methodology adopted for this project proved to be very useful, as it mitigated many of these risks. For example, pair programming ensured that an individual was assisted by a second member of the team while he or she was facing a difficult task. Working in pairs reduced the risk of not being able to accomplish tasks and also reduced the likelihood of bugs in the code. Another

helpful method is the weekly feedback meetings. It was particularly useful in addressing any new issues during programming and to assess existing ones to see if they had been overcome or not.

Secondly, version control is also on top of anything for either coding or documentation. We use CVS to manage our programs and documents. CVS means Concurrent Versions System. It keeps track of all work and all changes in a set of files, typically the implementation of a software project, and allows potentially widely separated developers to collaborate. Every Monday there is a weekly release in a new CVS folder in order to keep the relatively stable programs. The modification and addition programs/files will be done in a temporary folder first in order not to impact the whole integration of programs.

Thirdly, careful task plan and proper communication plan is top of anything. The importance and details of task plan were already introduced in the section of schedule. Communication plan was also mentioned in the section of team organization. Frequently contact with our supervisor proofs very useful to our project. Furthermore, emergency plan needs to take into consideration. For all those risks which are deemed to be significant, emergency plan was put in place before it happens. For example, in the case of server shut down, we have all files/programs backup and are able to run our program in another system which has the similar environment as the server.

Lastly, learning to drive is the key to manage and reduce risks. The drive of a software project is the customer. If the software doesn't do what they want to do, this project is failed. As a programmer, the job is to give the customer a steering wheel and give them feedback exactly where we are on the road.

7.5 Status Control and Monitoring

7.5.1 Version control

CVS was used for version control for both coding and documentation. Using CVS, we can record the history of our source files. Instead of save every version of every file, CVS stores only the differences between the versions. CVS helps a lot for working in a group on the same project: CVS merges the work when each developer has done its work and avoids concurrent modification conflicts.

7.5.2 Defect code root cause tracking

The root cause tracking of the defect code provides reports highlighting priority issues, showing the most recently reported bugs, and summarizing problems by categories. And most importantly, we need a solution that saves us valuable hours by automating the existing internal processes for escalating and resolving program concerns.

The submitter of issue could include additional helpful information such as:

What is the priority (in the submitter's opinion) for this issue?

Is this issue on the critical path to meeting another milestone?

How was this issue found?

Has the submitter already found any work-around?

Name	Sub Function	Defected Number	Defect Cause	Estimated Date	Scheduled Date	Impact /Priority

7.5.3 Re-factoring code and program naming standard

In software engineering, the term re-factoring is often used to describe modifying source code without changing its external behaviour, and is sometimes informally referred to as "cleaning it up". Re-factoring is often practiced as part of the software development cycle: developers alternate between adding new tests and functionality and re-factoring the code to improve its internal consistency and clarity. Testing ensures that re-factoring does not change the behaviour of the code.

Re-factoring is the part of code maintenance which does not fix bugs or add new functionality. Rather it is designed to improve the understand ability of the code or change its structure and design, and remove dead code, to make it easier for human maintenance in the future. In particular, adding new behaviour to a program might be difficult with the program's given structure, so a developer might re-factor it first to make it easy, and then add the new behaviour.

An example of a trivial re-factoring is to change a variable name into something more meaningful, and to make the variable name standard. The class name, file name, variable name, const name, etc, need to be defined according to the naming standard.

Re-factoring is done as a separate step, to simplify testing. At the end of the re-factoring, any change in behaviour is clearly a bug and can be fixed separately from the problem of debugging the new behaviour. Re-factoring is an important aspect of extreme programming.

7.5.4 Test process standardization

Test-driven development (TDD) is a programming technique heavily emphasized in Extreme Programming. Essentially the technique involves writing your tests first then implementing the code to make them pass. The goal of TDD is to achieve rapid feedback and to implement the "illustrate the main line" approach to constructing a program.

It is good to emphasize the fact that TDD is not a method of testing, but a method of designing software. While we are running the test, several steps must strictly abide, such as test manual follow up, test result tracking, bug cause tracking etc. The detailed test process can refer to our testing manual.

7.6 Report

In our web blob, weekly meeting minutes are basically a kind of report which records the weekly work assignment, completeness and issues, and final documentation.

The purpose of the final report is to provide a comprehensive description of our entire project. The intended audience for the document should be the client, which may include people who are not intimately familiar with all aspects of our particular project. Because of this, we should make sure that the background and context for the project (in the introduction section) are clearly explained and that the motivation for the project is clear.

The following items should be included in our document; the details of each section correspond to the chapters in our final report:

- *Abstract*. The abstract might be used by someone who wanted a quick overview of the project
- *Introduction to the Project*. This section gives a high-level description of the project, in narrative form. It includes the background and motivation for the project.

- *Requirements and Specifications.* This section corresponds to the main body of our Software Requirements Specification. It includes both functional and non-functional requirements.
- *Design or Solution Approach.* This section corresponds to the main body of our Software Design Document. It includes graphical and/or formal representations of your design, such as UML diagrams, use case scenarios, database table layouts, sequence diagrams, state transition diagrams, data flow diagrams, etc.
- *Implementation Details and Results.* This section provides additional information about our implementation and/or results. Some possible items to include are:
    - Design decisions regarding language and/or tool used
    - Design decisions regarding writing vs. using code.
- *Project Progression/Project management. This section* includes flexible scope, possibly with identified milestones or a number of optional items.
- *Conclusions and Future Directions.* For many projects it will be useful to include lessons learned.

Chapter 8 Development Tools & Technologies

8.1 Programming Languages

Due to the requirement, Python was selected for the implementation of the toolkit. Python is a scripting language and very easy to use and easy to implement. The properties such as multi-platform, interpreted, interactive, object-oriented programming language make Python very suitable and powerful for this project. Python is fully dynamically typed and uses automatic memory management. Through the successful carrying out of this project, it is proved that Python is very suitable for the implementation of the toolkit, which is also the reason why the LINX or other IXPs prefer it. A number of detailed advantages of Python have been discussed below:

The implementation of the toolkit requires plenty of shell scripting (such as text and file manipulation), and network programming. Like most scripting languages, Python features excellent text and file manipulation capabilities. Unlike most scripting languages, Python sports a powerful object-oriented environment with a robust platform API for network programming and threads. Without Python, developers were forced to rely on a variety of tools used in awkward combination with one other. One scenario would be using Perl for text and file manipulation and using Java for network programming.

In addition, Python's power and ease of use combine to make it an excellent choice for writing programs that process XML data. As a high-level language, Python includes many powerful data structures as part of the core languages and libraries. The more recent versions include excellent support for Unicode and an impressive range of encoding, as well as an excellent and fast XML parser that provides character data from XML as Unicode strings. Python standard library also contains implementations of the industry standard DOM and SAX interfaces for working with XML data.

Of course, there are many languages capable of doing what can be done with Python, but it is rare to find all the qualities of Python in any single language. For instance, Java has a powerful API’s for processing XML and network programming, but it is not a good choice for text and file manipulation. These qualities do not so much make Python more capable, but they make it much easier to apply, reducing

programming hours. This allows more time to be spent finding better ways to solve real problems or just allows the programmer to move on to the next problem.

8.2 Tools

8.2.1 Fping

Fping is a ping like program, which uses the Internet Control Message Protocol (ICMP) echo request to determine whether a host is up. Fping is different from ping in that you can specify any number of hosts on the commanded line, or specify a file containing the lists of hosts to ping. Instead of trying one host until it timeouts or replies, Fping will send out a ping packet and move on to the next host in a round-robin fashion. If a host replies, it is noted and removed from the list of hosts to check. If a host does not respond within a certain time limit and/or retry limit it will be considered unreachable. Unlike ping, Fping is meant to be used in scripts and its output is easy to parse.

In this project, we used a modified version of Fping. The new tool adds a precise timestamp to the output of each request, and this timestamp is very useful for the monitoring information.

8.2.2 CVS

Concurrent Versions System (CVS) is a powerful method of allowing many developers to work on the same source code. It is used extensively within the project in order to make every user accessible to all the files and avoid modification conflicts

The work process of CVS is also not complicate. Each developer checks out a copy of the current version of the source code from CVS and then is able to work on their own personal copy separately from other developers. When they have made changes, they commit them back to the CVS repository. The CVS server is then able to merge all the changes that the developer has commit back. Sometimes this merging is not always successful, the developers are notified and they will have to manually fix any possible conflicts that arise before trying to commit their changes again.

CVS was proved to be very helpful for version control for both coding and documentation and become one of the most important tools for project management.

8.2.3 Pydoc

The Pydoc module automatically generates documentation from Python modules. The documentation can be presented as pages of text on the console, served to a web browser, or saved to HTML file.

It is imperative to provide clear and detailed documentation to IXP in case that they want to modify the APIs and develop more functionality. Pydoc is easy to use and can provide a description of the classes, When Pydoc generates documentation; it uses the current environment and path to locate modules. The usage of Pydoc is the shell command line outside of Python. We used the pydoc.py written by Ka-Ping Yee. The command is “python pydoc.py –w filename”. Filename is the Python file name.

Glue, PoD, Client sample codes are all easy to read the check the classes and data through the HTML generated by PyDoc.

8.2.4 MySQL

The MySQL database server is the world's most popular open source database. The Structured Query Language (SQL) is a very popular database language, and its standardization makes it quite easy to store, update and access data. One of the most powerful SQL servers out there is called MySQL and surprisingly enough, it is free.

Since we are using SSL to secure the data transmission between PoD server and client, the certificate management becomes a problem while there are more than hundreds of clients. We have considered about the different solutions to manage certificates. The first choice is to store certificate into files in a fixed directory. The advantage of storing as files is that it is fast to get data from files, but the trade off is the scalability. When there are too many clients, it will be difficult to search a certain file for a certain user. The second choice is to store files into database. Although the database programming is comparatively complicated compared with file storing, the biggest advantage is that it solved the problem of scalability. While there are hundreds or thousands of users, it is quite efficient to search by using database. So finally we decide to use database. MySQL is a stable, fast, easy to learn relational database that runs across a wide variety of operating systems and is available under both Open Source and more conventional licenses. MYSQL is multi threaded and multi user. It is arguably the most popular database used for adding and processing

large amounts of data on the Internet. And MySQL has very friendly interface with Python. The use of MySQL proved to be very successful during our programming.

8.2.5 OpenSSL

OpenSSL is a free toolkit that is used to generate self-signed certificate authentication, public and private keys for the communication parties who are interested in SSL connection.

One of the reasons that we use OpenSSL is just because it is free. The OpenSSL toolkit is licensed under an Apache-style licence, which basically means that it is free to get and use it for commercial and non-commercial purposes subject to some simple license conditions. OpenSSL is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation.

Another reason to use OpenSSL is that it can provide the correct format of key files and certificates that is compatible with Python SSL interface. Python has very friendly SSL interface, it just need the .pem format files that OpenSSL can generate. OpenSSL can create both server and client key files and certificate request files. Then it can sign certificate for both server and client.

In addition to the SSL implementation, OpenSSL toolkit includes utilities for certificate management. It also includes a public key implementation which may be used outside the United States. In the United States, RSARef or BSAFE3.0 must be used due to patent requirements. OpenSSL offers an inexpensive way to get started with SSL.

So in our ExSERT project, OpenSSL is installed in order to generate key files and certificates in both server and client in this project. This tool was downloaded from OpenSSL organization’s web link.

8.2.6 Twisted

Python native SSL package only supports SSL in client side rather than server side. So we have to find a third party library that provides more network application functions to provide SSL server side securities.

Twisted package is a third party library that provides plenty of network application functions including SSL for Python language. The most useful advantage for our project is that Twisted package support SSL in both server and client side.

Twisted is an open source networking framework, implemented in Python which can run on multiple operating systems and platforms.

Twisted is an attempt to build a framework capable of supporting the needs of modern network applications, from simple custom protocol development to large-scale multi-protocol integrated systems. It provides:

- Multiple levels of abstraction, starting with a low-level platform-specific networking event loop, and moving up to generic networking code, common protocol implementations, and high-level frameworks.
- Cross platform (Windows and Unix) support while still allowing platform-specific code (as opposed to Java's design decision to provide only the lowest common denominator).
- Integration and ease of use.
- “Batteries included." The framework includes everything necessary to build a network application.

Twisted is divided into several packages such as twisted.internet, twisted.application and twisted.protocols, each providing different services. Both twisted.internet and twisted.protocol are used in this project in order to implement SSL between clients and PoD server.

8.2.7 Rational Rose

Rational Rose is a tool that automates and simplifies the creation and modification of UML designs. It supports two essential elements of modern software engineering: component based development and controlled iterative development. Models created with Rose can be visualized with several UML diagrams.

The tool was used to create the packet hierarchy, structure and dependencies of classes, state diagram. So in our ExSERT project, classes in Glue, Compiler, PoD were illustrated by using Rational Rose.

8.2.8 Visio

Visio is the leading business-drawing tool for creating diagrams, flowcharts, and schematic drawings. Learn to create timelines, office layout plans, workflow diagrams, organizational charts, maps, flowcharts and more with speed and precision. In addition, Visio helps us communicate our ideas visually. While Rational Rose

provides excellent UML diagram, Visio was used to draw any other diagram used in our report.

Chapter 9 Future work

Due to time and resource constrains of this project, especially in terms of the toolkit design and implementation; there are several interested and valuable functionalities are worth of being refined or improved in the future such as network error prediction, compiler, super PoD, XML validation methods and separating of PoD and Glue.

9.1 Network Error Prediction

The purpose of the network error prediction is to alarm the network administrator before network error or disaster happens. Until now an effective and suitable algorithm still has not been found to forecast the network error or disaster. In fact, in order to achieve this, a combination of a set of algorithms rather than a single one have to be made since the prediction may depend on multiple kinds of network data. Furthermore, it is obvious that the prediction algorithm must have a higher accuracy and less complexity. At the moment, all historic data generated by fping tool can be saved in the MySQL database. Then, further analysis can be done based on the raw data in the database. In addition, it will be significantly useful if this error prediction algorithm can locate the dedicated network error and provide correspond and effective solutions as well.

9.2 Future Development of the Compiler

The aim of the compiler is to generate basic classes and common code in Glue and PoD. It had not been started until the first prototype of the project was finished, which can provide concrete input and target of the compiler. Therefore, the development lifecycle of the compiler should be an iterative and incremental process that will be improved and modified based on different kinds of data analysis and user requirements. In other words, a general and flexible compiler can only be made by considering more and more requirements from users and system.

In addition, more data types such as Ethernet address and IPv6 address should be generated as a basic class beforehand in order to satisfy heterogeneous hardware and operating system platforms.

9.3 Super PoD

A super PoD is a PoD that collects data from different individual PoD on which future data analysis and comparison can be done. It provides a high level prospective for the network administrator by using distributed monitor architecture. In other words, this is a communication among different PoDs rather than clients. Therefore, pull model seems suitable since it guarantees the super PoD get desirable data from the preferred PoDs.

In addition, if historic data in each PoD is saved in a database, the data share or exchange between the super PoD and individual PoD should be implemented according to the table structures. However, it cannot be guaranteed that different PoD has same table structures such as data field name and type. Therefore, data analysis will be a big issue in future.

9.4 Validation between DTD and XML schema

Due to lack of enough libraries of supporting XML schema in Python, DTD is chosen in this project although it does not provide data restriction, which has to be tackled in our own program. Some third party packages also can solve this problem, but they are so large and lack of necessary document. Therefore, the best way of solving this is that there are Python libraries that can support validation for XML schema, which is obviously not practical at the moment but may be truth in the future.

9.5 Separation of Glue and PoD

Currently, both Glue and PoD are in the same location. However, they are very possibly separate in different locations due to the different system architecture requirements. In fact, it is relative easy to accomplish this in the internal network. The problem is that SSL has to be used over the public network, as the data is confidential. Of course, SSL does not need to be used in the case of data transmission only happens within internal network of an IXP and bandwidth should not be a problem as well.

However, some issues have to be considered when data is transmitted over the public network such as packet delay, packet loss and so on.

Finally, other possible extensions may be developed by cooperating with other universities or organizations in near future such as looking at routing information with university of Cambridge, checking the problem space in more detail with IXP community and deploying the monitoring more widely with Euro-IX network.

Chapter 10 Conclusion

The primary objective of this project is to re-engineer the ExSERT project based on last year's system architecture. Basically last year's project already solved the problems such as data heterogeneity between different ISPs. The main purpose of this year's project is to design and implement a toolkit in order to achieve more objectives such as easy creation of RMFs, excellent extension ability, and easy deployment. This project is also different from other DCNDS projects since it will have final customer such as LINX and many other potential users such as other member IXPs of Euro-IX. So as an integrated project, the whole project lifecycle from design to testing and report was under the control of project management.

One of the most import novel parts of this year is to develop a toolkit including a compiler, which is able to generate basic classes and common code. This toolkit provides a more convenient way to build RMF. At the beginning, it is difficult to make sure the input of the compiler although XML is the optional candidate of input. And the output of the compiler is also not very clear until the initial prototype was created. Then, after a long iterative and incremental development process, the compiler was implemented successfully and generated the main components such as PoD and Glue near perfectly. Generally, the compiler has fulfilled its original goals such as generality, flexibility and scalability. In a word, the design and implementation of the compiler is the biggest achievement of this year's project.

The other accomplishment is that XML is used to represent the similar data and to transmit in the network in a common way. This resolves the data heterogeneity generated by different back-end network tools effectively. Compared to the data communication protocols developed by last year, XML provides an enterprise standard way that makes different kind of data embedded into dedicated XML tags defined by end users and uses XML schema or DTD to verify the different data type. This brings much convenience for both data transmission and data analysis especially in the case of dealing with complicated data types. Moreover, XML is a common standard recognized in the filed of industries. This will be a huge support for XML, and consequently beneficial to our project.

Another breakthrough of this project is the use of Python. Python is an open source and cross platform scripting language, which is easy to use and implement, effectively saving the programming time and resource. It is a complete object

oriented language that is beneficial to the programming and code management. More significantly, unlike java or c++, the Python program is very clean during the whole running process without any other intermediate files.

Moreover, binary encoding is added into this project. It converts the XML file into a binary file and transmits it over the public network. Meanwhile XML solves the problem of data heterogeneity, XML's processing overhead, storage requirement, and bandwidth consumption become quite problematic when transaction volumes are high. The encoding uses binary, rather than text-based, means for serializing and transmitting XML information. It promises to significantly alter the processing, bandwidth, and storage penalties that currently plague XML. That will be helpful for data transmission among heterogeneous hardware and operation system platforms. Especially in the scenarios such as wireless network, binary encoding provides great efficiency of data transmission.

The secure communication between IXP and clients is one of the core requirements of this project. SSL is implemented with X.509 certificate to provide secure communication between client and server. A third party tool named OpenSSL is used together with a third party library named Twisted. It is not a hard issue to understand SSL; however, it is difficult to master the third party libraries or tools because of the various options and the lack of help menus. Therefore, plenty of research was done in order to choose the most suitable one from a serial of third party libraries and to test the relative functions provided by the third party libraries.

On the other hand, the successful completeness of the project doesn't mean that we did not meet any challenges along the way. Firstly, it took a long time to do research including scripting language Python, the validation method comparison between DTD and XML schema and the architecture of compiler. Secondly, it is also a big challenge to choose one library from some of the third party libraries. Thirdly, an effective algorithm still has not been found to predict the network error although some research has been done. Lastly, although we realize that the application would benefit aesthetically from a more attractive user interface as a demo to the final user, we opted to focus on issues such as functionality rather than user interfaces.

Fortunately, those challenges make us know better how important the teamwork is and the importance of project management. Great team work with professional project management laid good foundation to the current success of our project. Everyone in our group feels the team spirit deeply that drives out project go

forward constantly, especially when we are facing difficult problems and need to decide what the next step is.

In conclusion, it is a big success to re-engineer the ExSERT project of building an easy and fast toolkit based on previous proved successful system infrastructure. All the team members contributed hugely to gain the achievement of the project. We are fully confident that this project will be deployed very soon and operate smoothly on LINX. We also give the best wishes to the future groups that continue the future works of our project.

References

AfNOG 2001 Meeting Presentation, *Peering and Internet Exchange Points* [online], Available at: http://www.afnog.org/2001/peering-and-internet-exchange-points.ppt

Beck K., 1999, *Extreme Programming Explained: Embrace Change*, Addison-Wesley

Bradley N., 2003, *The XML Schema Companion*, Addison Wesley

Cisco, *Internetworking Technology Handbook* [online], Available at: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/index.htm

Collins G., *Lecture slides* [online], Available at: http://www.cs.ucl.ac.uk/external/g.collins/postgrad

Dubuisson O., 2000, *ASN.1 Communication between Heterogeneous Systems*, Morgan Kaufmann

Emmerich W., 2000, *Engineering Distributed Objects*, 1st Edition, John Wiley & Sons

Farthing D. W., Software Project Management [online], Available at: http://www.comp.glam.ac.uk/pages/staff/dwfarthi/projman.htm

Goerzen John, 2004, *Foundations of Python Network Programming*, Apress

Jones C. A. and Drake F. L., 2002, *Python and XML*, 1st Edition, O'Reilly

Larmouth J., 1999, *ASN.1 Complet*e, Morgan Kaufmann

OpenSSL Home [online], Available at: http://www.openssl.org

Perkins D. T. and McGinnis E., 1996, *Understanding SNMP MIBs*, Prentice Hall

Pilgrim M., 2004 *Dive Into Python*, Apress

Project Management Schedule [online], Available at: http://www.dof.ca.gov/HTML/IT/PMM/OPTIMIZED/PM3.4%20Planning%20Project%20Schedule.pdf

Priestley M., 2003, *Practical object-oriented design with UML*, 2nd Edition, McGraw-Hill

Royce W., 1998, *Software Project Management: A Unified Framework*, 1st Edition, Addison Wesley

Schmelzer, R. D., 2002, *Understanding Schemas and DTDs Presentation*, ZapThink LLC

Souter J., *IXP Network Monitoring Tool Portability Presentation* [online], Available at: http://www.cs.ucl.ac.uk/teaching/dcnds/seminars/2003-11-25-linx.pdf

Stallins W., 1999, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, 3rd Edition, Addison Wesley

Stevens R. W., Fenner B., and Rudoff, M. A., 2003, *Unix Network Programming, Vol.1: The Sockets Network API*, 3rd Edition, Addison Wesley

Summerville I., 2001, *Software Engineering*, 6th Edition, Addison-Wesley

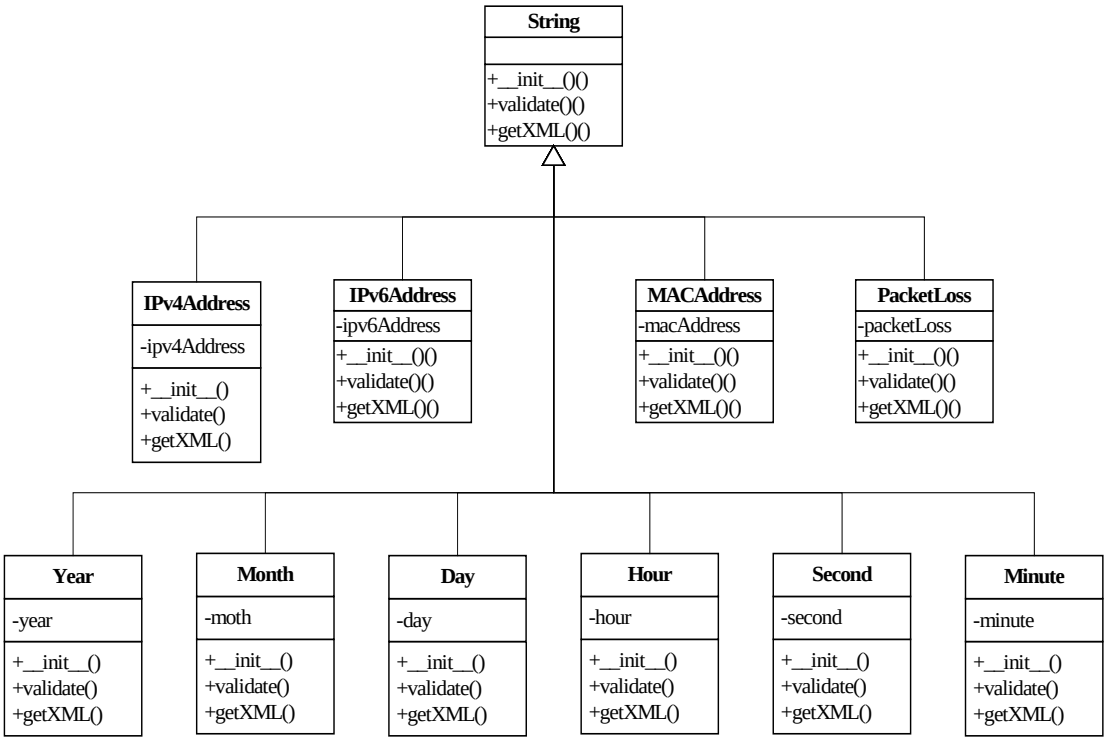
W3C (World Wide Web Consortium) Schools [online], Available at: http://www.w3schools.com/

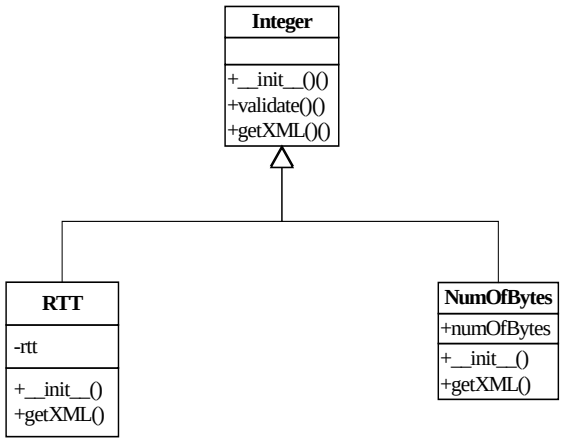
Wiegers K., *Peer Reviews in Software: A Little Help From Your Friends* [online], Available at: http://www.informit.com/articles/article.asp?p=24370

Wikipedia (the free encyclopedia) [online], Available at: http://en.wikipedia.org/wiki/Main_Page

Appendix A: Class Diagram

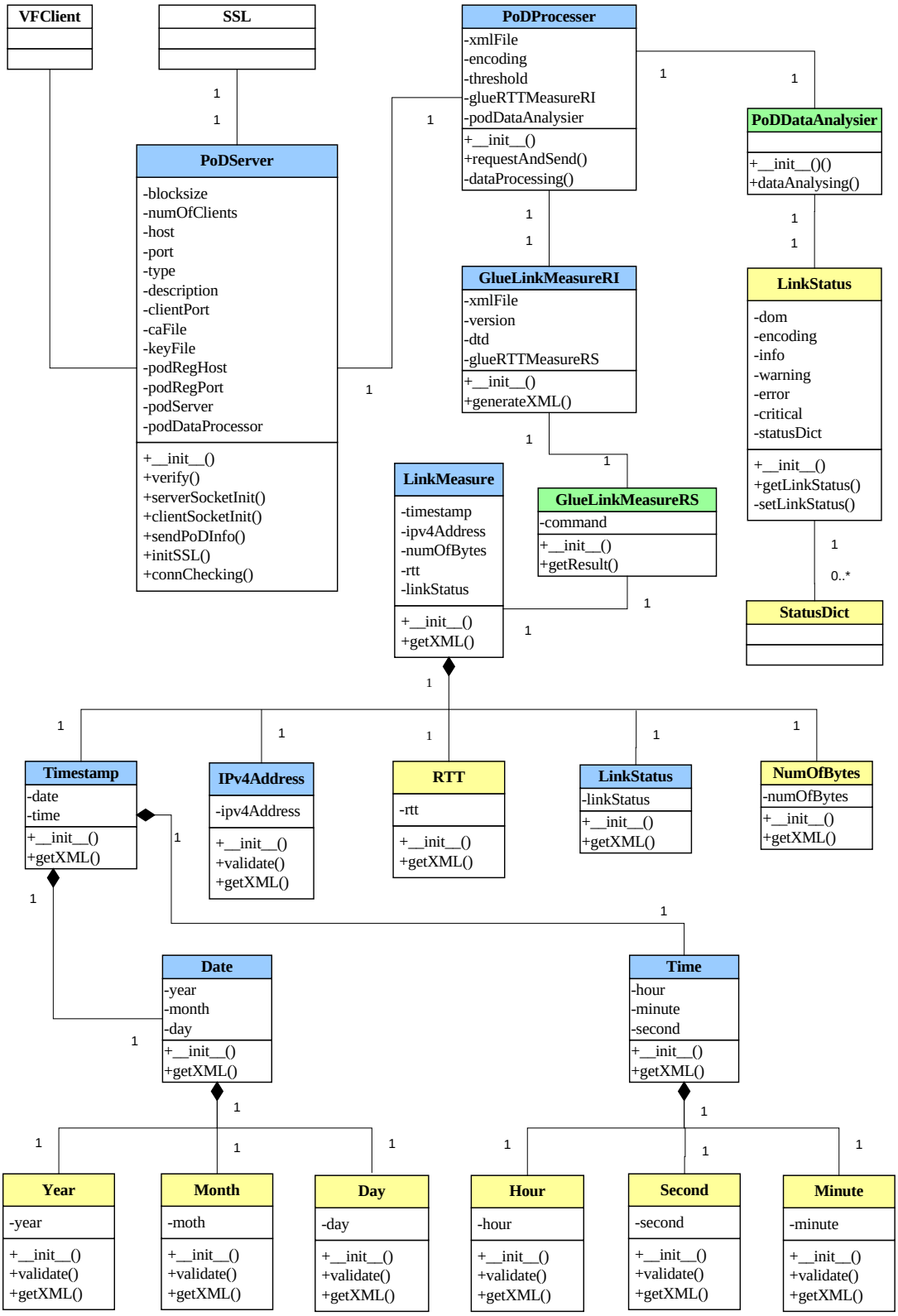
1 Basic Data Types



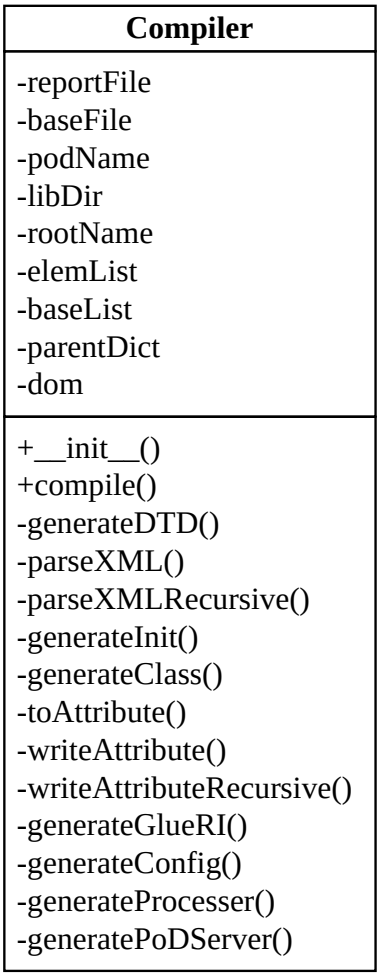


2 LinkMeasure RMF

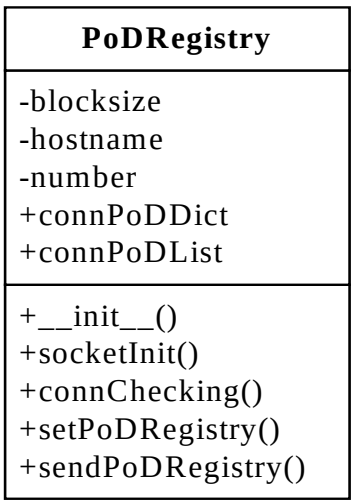
The light-blue classes are generated by the compiler, the light-green classes are added by the user, and the light-yellow classes are provided by the API.

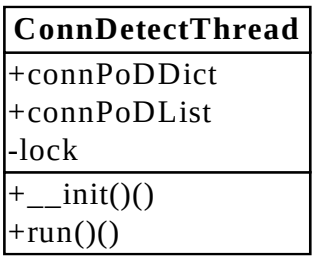


3 Compiler

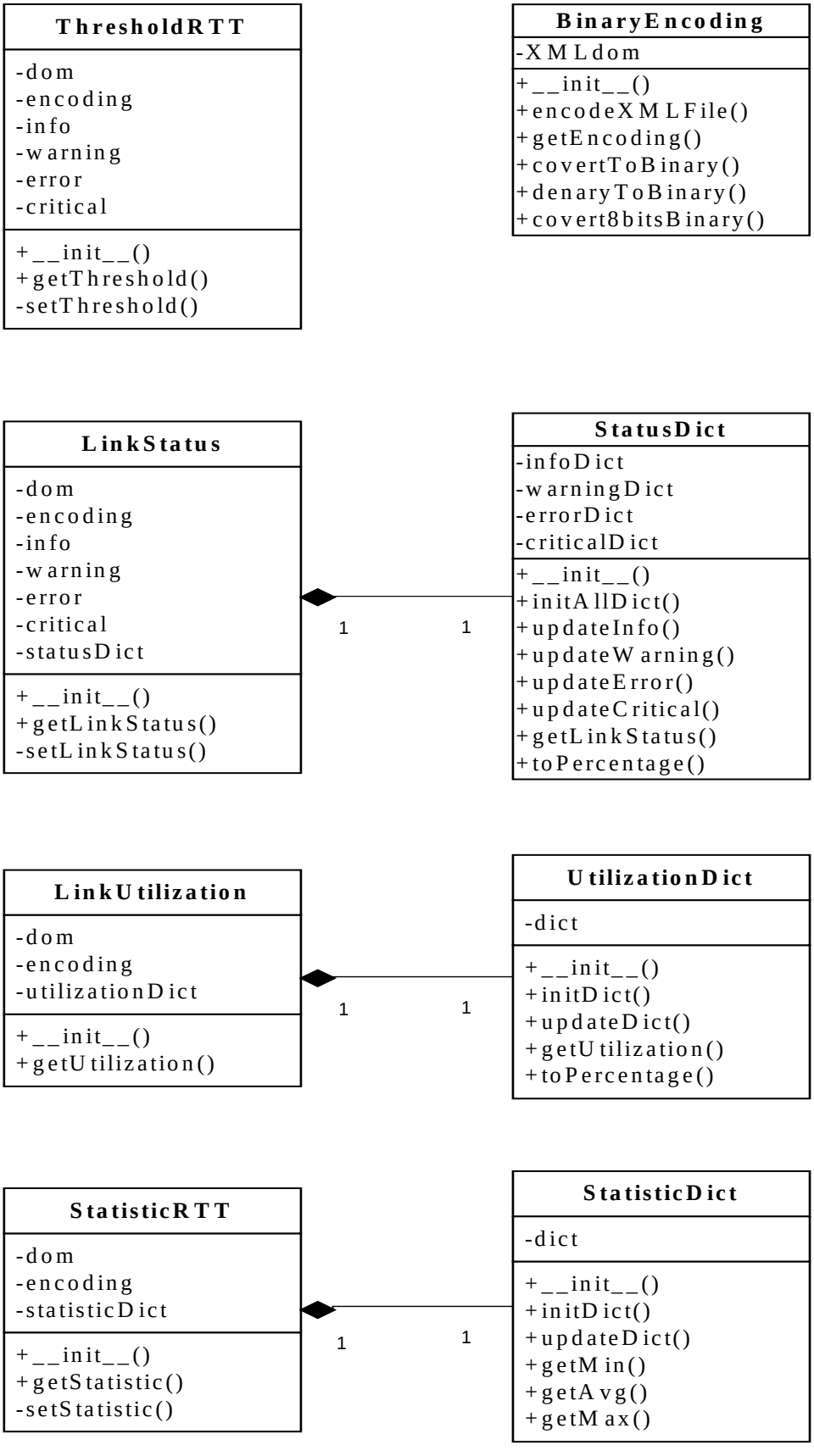


4 PoDRegistry

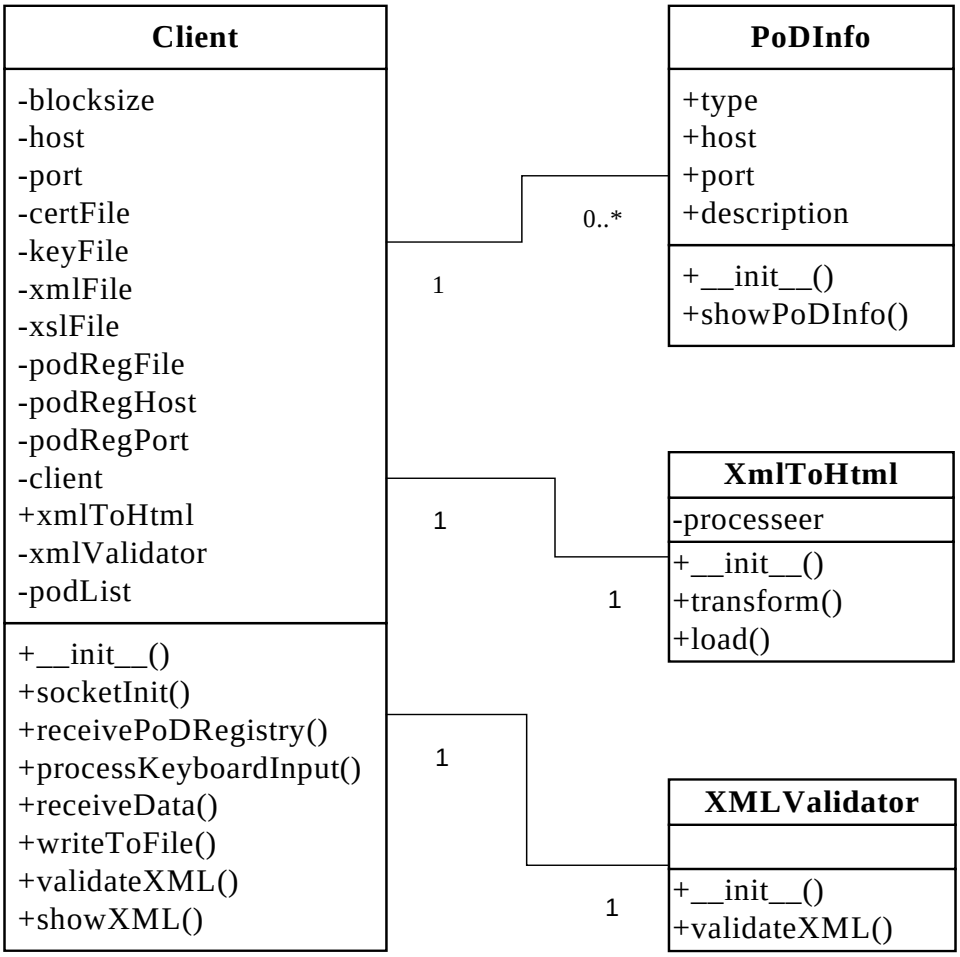




5 Data Analysing Function



6 Example VFClient



Appendix B: Sequence Diagram

This diagram shows the interaction between the client and system components, and some classes have been excluded for the simplicity of drawing.