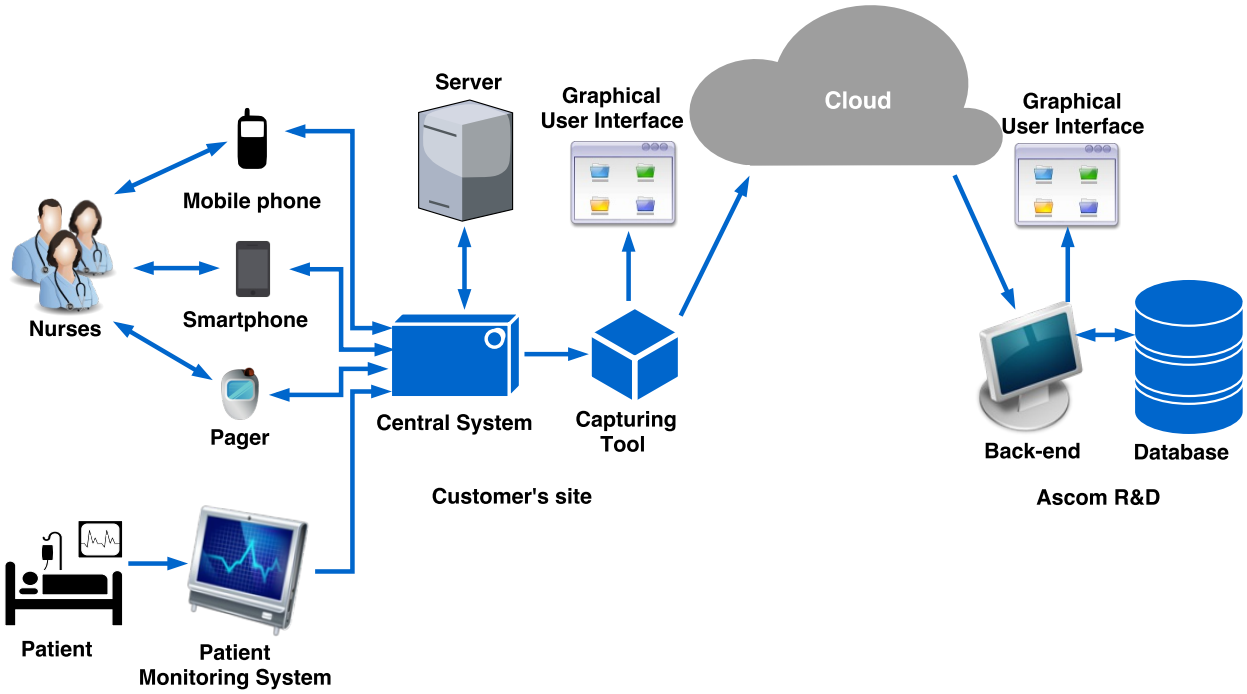
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Generic Log and Performance Data from Customer Installations

Collection, Transmission, and Processing of Unite Communications from Ascom's Unite System at Customer's Site

Master's thesis in Computer Systems and Networks

ALAA ALNUWEIRI

JEMIMA MASAMU

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2016

Master's thesis 2016

Generic Log and Performance Data From Customer Installations

Collection, Transmission, and Processing of Unite Communications from Ascom's Unite System at Customer's Site

ALAA ALNUWEIRI

JEMIMA MASAMU

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Computer Networks and Systems
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2016

Generic Log and Performance Data from Customer Installations
Collection, Transmission and Processing of Unite Communications from Ascom's Unite System at Customer's Site

ALAA ALNUWEIRI
JEMIMA MASAMU

Supervisor: Andreas Abel, Department of Computer Science and Engineering
Supervisor: Mats Andreasen, Ascom Wireless Solutions
Examiner: K V S Prasad, Department of Computer Science and Engineering

Master's Thesis 2016
Department of Computer Science and Engineering
Division of Networks and Systems
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of our solution to capture the Unite System communications. This solution captures information from the customer's site, then transfers the information via Internet for storage in a database located at Ascom R&D.

Typeset in LaTeX
Gothenburg, Sweden 2016

Generic Log and Performance Data from Customer Installations
Collection, Transmission and Processing of Unite Communications from Ascom's Unite System at Customer's Site
ALAA ALNUWEIRI
JEMIMA MASAMU
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Ascom Wireless Solutions has been developing on-site wireless communication solutions to customers all over the world. It does this by providing customers with solutions to support and optimize their mission-critical processes. The information generated from these solutions has been used by Ascom in developing their products. But, Ascom faces a challenge on how to gather this information as they use a method that is time-consuming, requires customer's involvement, and imposes security concerns to their network.

The main aim of this thesis is to get a better understanding of the Unite System which is being used differently by different customers. The understanding of Unite System involves capturing of large amounts of data (big data) generated from customer's site. The captured information assists in further system development such as database processing. Furthermore, Ascom can use it to perform analysis and troubleshoot different faults or errors identified in Ascom's systems.

In this project, we used Ascom's special tool known as a *Buslogger* which was configured and automated on an embedded PC to capture Unite communications from customer's site. To present Ascom with captured information, we tested both FTP applications and cloud storage services for transmission, and relational and NoSQL database solutions for storage. We later selected Tresorit cloud service for transferring Unite communications due to its compatibility with HIPAA standards and ability to support end-to-end Encryption. Once the transferred files reached Ascom R&D, we stored them in Oracle database. Among tested solutions, Oracle database is the only relational database that supports XML datatype and is easier to manage when compared to NoSQL.

Keywords: Big data, Unite (Unified IP and Telecommunication Environment), FTP, SQL, Encryption, Privacy, Cloud.

Acknowledgements

We are taking this chance to express our sincere gratitude to our supervisor Andreas Abel and examiner K.V.S. Prasad, of the Department of Computer Science and Engineering, Chalmers University of Technology, for their valuable supervision and guidance throughout the whole study.

Special appreciation is extended to our co-supervisor Mats Andreasen of Ascom Wireless Solutions, for his valuable assistance, encouragement, and guidance throughout the project.

Our sincere gratitude to Ascom Wireless Solutions, Gothenburg office for their facilitation in providing office space, data and a great team of professionals who assisted us at every chance they got.

Also, we are profoundly grateful to the Swedish Institute for awarding us full-time scholarships to pursue our master degrees.

Last but not least, we would like to thank our families and friends in Palestine, Tanzania, Syria, and Sweden for all the love and support.

Alaa Alnuweiri, Gothenburg, September 2016
Jemima Masamu, Gothenburg, September 2016

Contents

List of Figures

List of Tables

1
Introduction

Big data is being used in most of the IT sectors as a means of running analysis on large data sets through high-performance computing (HPC) systems. There are key enablers that have contributed to the growth of big data which include the increase of storage capacities, availability of data, and the increase of processing power. The big data storage is used to handle enormous amounts of data, keep scaling to keep up with growth, and provide the input/output operations per second (IOPS) necessary to deliver data to analytics tools [10]. Most of the big data practitioners e.g., Google or Amazon, run hyper-scale computing environments such as servers with direct-attached storage. Although big data storage has been implemented in several IT sectors, the challenge remains as to: *How do you capture, store, access and analyze enough data to gather those insights and justify the resources that have been committed to the task?*

Big data can be characterized by either volume, velocity (e.g., data streaming) or a variety in terms of structured or unstructured, meaning data with predefined data schema/model/structure or data that does not have a well-defined data model respectively. Due to advancements in technology, volume or size of data can range from Gigabytes to Terabytes, and even more. The velocity refers to the speed at which data is generated since different applications have different speed requirements when generating data. Additionally, variety refers to different formats in which data is generated or stored, where unstructured data is widely generated as opposed to structured data. Such data can be derived from sources such as enterprises, transactional systems, and social media [19].

1.1 Background

The modern company environment uses various communication technologies and devices to communicate and collaborate with various stakeholders. The communication infrastructure has changed from the traditional wired local area network to embrace the emerging wireless technologies, for example, *Wi-Fi*. Additionally, the devices used to connect to the network are not limited to company computers only but to more devices such as smartphones, tablets, and laptops.

Ascom Wireless Solutions develops on-site wireless communication solutions to customers all over the world, by providing them with solutions to support and optimize their mission-critical processes. Additionally, Ascom Wireless Solutions uses a

large amount of data (big data) generated from customer's site in order to develop their products. The collected information also aids in activities such as performing analysis or troubleshooting different faults or errors identified in Ascom's systems. However, there is a challenge on how to gather big data which is currently used at the customer system, and security concerns that may arise in the process. Additionally, Ascom has a vast geographical distribution with a large number of installed systems, which presents a challenge in understanding the use of their products and solutions since the installed systems are being used in many different ways.

1.2 Project Objectives

The main aim of this thesis is to get a better understanding of the Unite System since it is being used differently by different customers. This helped in obtaining information that assists in further system development such as database processing and analysis. Understanding of Unite System involved investigating and evaluating different approaches for collecting large amounts of data (big data) using TCP/UDP IP-based communications from customer's site. The primary beneficiary in our case is Ascom, our industrial partner, who will use the solution to better understand patterns in which its customers use Unite Systems. In terms of academia, we contributed by introducing an information model that can be used by other researchers in order to duplicate or extend the work of this project. The model assists in obtaining information about different events, such as the times the alarm went off and messages sent and acted upon.

We achieved our goals by building an application and deploying it on the customer's site. The application kept track of the customer's activity in Unite System and log them. The log data was sent to a server located at Ascom in order to further analyze it. We studied different alternatives to implement and deploy the solution. We did this by having an embedded application installed on the embedded PC located at customer's site so as to solve the business problem with capturing big data as mentioned in the introduction.

We investigated different alternatives for transforming and storing the collected data, to facilitate analyzing it efficiently. Additionally, we identified metrics for measuring the system's performance.

When implementing the solution, we took into consideration the very important data security aspects. These include avoiding intrusion during data transmission, compliance with the customer's security regulations, and compliance with Health Insurance Portability and Accountability Act (HIPAA). HIPAA outlines the security standards to protect the created, received, maintained, and electronically transmitted data in the health domain. This implies that the transmitted data being encrypted and accessed by authorized persons only.

Achieving the project goal provided Ascom with the required information for further studies and researches in order to improve the working environment in hospitals.

That includes, but are not limited to, reducing nurse stress level, giving more nurse's time for the patient, and improving the work quality by reducing the risk for error. Moreover, it allows further studies by Ascom to improve the Unite System functionality by using the information model to identify bottlenecks and missed messages/responses and to detect errors.

1.3 Report Structure

The main structure of the report starts from Chapter 1 which introduces the background problem then details on general objectives of this thesis work. Chapter 2 gives an introduction to Unite System including its logical structure and protocols, and Unite Connectivity Manager. Furthermore, Chapter 3 surveys related works for secure file transfer protocols and database solutions. Chapter 4 highlights the architecture and design of our system. The specifications are given in Chapter 5 and include use case diagrams and their descriptions. Chapter 6 describes the methodologies used in capturing, transferring, and storing Unite communications, and the Proof of Concept. We finalize with the discussion of our findings in Chapter 7, concluding our findings and suggesting what to be done in the future in Chapter 8.

2

The Unite System

Ascom Unite System is the communication platform that links Ascom messaging system with mission-critical work processes and tasks. This system is designed to operate with high-performance, flexible and reliable communication. The Unite System communication is based on sending and receiving information in the form of XML-based messages. In addition, Ascom Unite system has a centralized web-based Graphical User Interface (GUI) for updating and configuring devices. The Unite System can be built by either of the two central modules: *Unite Connectivity Manager* (Unite CM) which is a Linux embedded server platform or *Unite Communication Server* (Unite CS). A Windows' server known as Unite Application Manager (Unite AM) provides an administration interface which facilitates configuration, management, and daily operations to customers. Unite AM works in combination with Unite CM and other Unite modules e.g., Mobile Monitoring Gateway (MMG), telecare IP System Manager (NISM2), and Unite Connect for Nurse Call module. Also, the Unite System has a platform which consists of following three logical structures as shown in Figure 2.1:

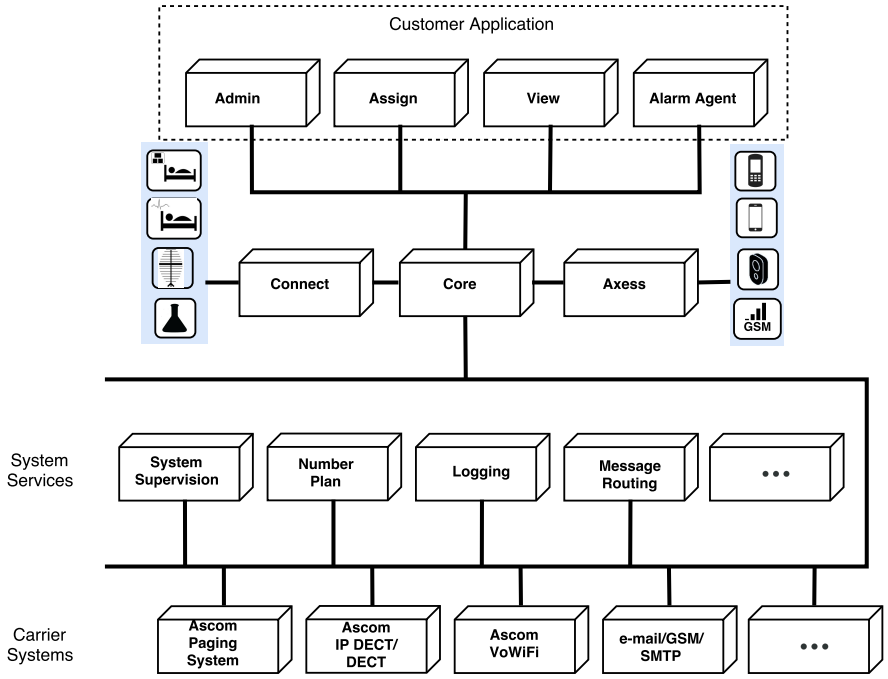


Figure 2.1: Logical structure of Unite System

- *Customer application* - These are applications running on the customer's site on Ascom Unite AM in a Windows server environment. Unite AM provides administration interface, capacity enhancing functionality, and throughput for the system. An example of these applications includes Ascom Unite Admin for the management of users and devices, Ascom Unite Assign for staff scheduling and assignment, Ascom Unite View for overviewing incoming events, and Ascom Unite Alarm Agent for sending out and coordinating emergency alerts.
- *System services* - Example of service handled in the system is message routing which also involves message diversion. Other services are system supervision in which proper handling of messages is ensured by Unite CM, activity logging for storing different activities e.g., messaging and alarms in the Unite System, and address resolving.
- *Carrier systems* - Unite CM is used as an interface to gain access to the carrier systems. It handles data bearer dependent communication with wireless handsets, input/output hardware and other systems. There is a range of different carrier systems such as Ascom paging system, Internet Protocol - Digital Enhanced Cordless Telecommunications (IP-DECT), Wireless Fidelity (WiFi), and Global System for Mobile communication (3G/GSM).

2.1 Unite Protocol

The Unite System is built up by modules which interact as one system using Unite protocol. The Unite protocol is an encrypted proprietary protocol that is built in the following three layers on top of TCP/IP:

(i) Unite Transport Protocol (UTP): Is a low-level transport layer protocol that enables a secure encrypted transmission channel between Unite components. It can be used as a transport protocol for message delivery in the Unite System.

(ii) Unite Service Delivery (USD): Is a high transport layer that handles the addressing and delivery of messages. This layer is located between Unite Transfer Protocol (UTP) and Unite Service Protocol (USP). Also, it can be seen as a mobility layer since it contains the location of the user which is not always static. Additionally, USD provides mobility to users and services and helps to route messages to the correct destination.

(iii) Unite Service Protocol (USP): This is an application layer that uses transport protocol for message delivery and defines the exchange of data between applications in the Unite System e.g., Unite services and their contents. Example of services defined by USP include:

- Sending messages to users and event communication e.g., interactive messages, paging or input events.
- System internal communications e.g., error handling, data updates or system control.
- Supervising communications and service registration.

2.2 Unite Connectivity Manager

Unite CM is used for delivering messages, data, and critical alarms to mobile staff members regardless of the type of device used (e.g., Ascom smartphones and IP-DECT handsets) and as an interface to different carrier systems such as VoWiFi and paging system. As shown in Figure 2.2, Unite CM connects different information systems e.g., Patient Monitoring System, by improving workflow through delivering of messages or events to staff. Also, the Unite CM supervises the Unite system so as to ensure safety during message handling. A good example being the delivery of the status log to the fault handler application in the Unite CM in the event of faults or when an equipment/module is lost in the system. In addition, the Unite CM can store collected logs consisting of different activities in the Unite System for troubleshooting or future analysis.

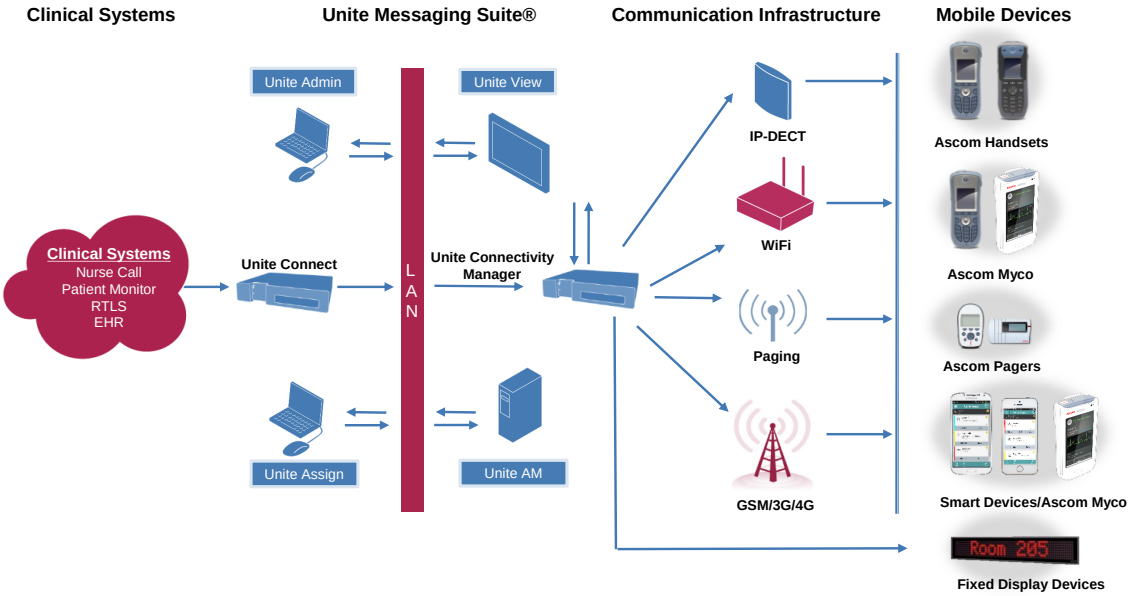


Figure 2.2: Unite System Illustration © Ascom

Furthermore, Unite CM offers other services such as message routing through the use of number plan tables and centralized management of devices used in the system i.e. configuration and software upgrading. In message routing, Unite Name Server (UNS) is used to contain the number plan tables which are used for translating unique call IDs used by users and application in the system to their destination addresses. The number plan assists in forwarding or diverting a message to a user/system or to another user/system respectively when the origin user is absent or out of range.

3
Related Work

3.1 Big Data Transfer Protocols

Se-young *et al.* [9] presented a characterization study of big data transfer protocols on a long-haul network. In this research the big data transfer protocols were categorized into *TCP-based protocols* which utilize the existing TCP congestion avoidance algorithms and *UDP-based protocols* that depend on the UDP congestion avoidance algorithms. In addition, three open-source protocols; *GridFTP*, *FDT*, and *UDT* were used in the analysis of big data transfer. GridFTP provides enhancements such as data striping, TCP socket buffer optimization, parallel data transfer and extensions to File Transfer Protocols (FTP). Fast Data Transfer (FDT) uses TCP as its transport protocol and provides multiple I/O threads and platform independent implementation. UDP-based Data Transfer (UDT) provides TCP-friendly congestion control on top of UDP and uses De-creasing Increases Additive Increase Multiplicative Decrease (DAIMD) and rate control to achieve high throughput over long-haul networks.

In the experiments performed, round trip time (RTT) was used to investigate the effect of congestion caused by the protocols building up queues in routers along the path, which increased as the data flow increased. It was later found that using data flow, GridFTP had the fastest data transfer, UDT had an implementation issue which limited its performance and FDT had limited performance due to the use of small buffer size.

3.2 Big Data Security in Globus

Secure methods are of equal importance as high-performance methods when the data size grows. An example of these data sets includes medical, personal, financial, government and intellectual property data. In the research by Kyle Chard *et al.* [3], *Globus* was used for the customers to access, move, and share large amounts of data remotely. Globus capabilities were split into *Globus Nexus service* which manages user identities and *Globus transfer service* that manages transferring, synchronization and sharing of tasks. In ensuring security for the user identities, Globus Nexus stores salted passwords for comparison when authenticating, SSH public keys and X509 Certificates for single-sign-on. Furthermore, in transferring data between logical endpoints, Globus uses GridFTP protocol in which data encryption is supported based on secure sockets layer (SSL) connections.

Although Globus managed to provide high-performance, reliability, secure data transfer, synchronization, and sharing of data compared to its competitors such as GreenButton and WarpDrive, there were challenges during its use. Some of these challenges include difficulty in accessing metadata from the files due to storage in different formats and inefficient use of file-based data because the data required for analysis does not always match the model used to store it [3].

3.3 Cryptographic Cloud Storage

Most of the cloud-based services such as Dropbox, Tresorit, Google Drive and One Drive use either *server-side encryption* or *end-to-end encryption* when sharing or storing files. So, cloud service providers are faced with the challenge to ensure integrity, confidentiality, and control over stored data. Most of the cloud services e.g., Dropbox and Google Drive, use server-side encryption in which the encryption/decryption of the stored files is offered by the cloud providers. In this type of encryption, stored data remain secure in the cloud (at rest), but users have no control over how the encryption is done or who has access to the decryption keys. Since the stored files get encrypted and decrypted at the cloud, the files remain insecure in transit if (SSL/TLS) has not been used by the cloud service in creating a secure channel.

The challenges in server-side encryption have resulted in the use of end-to-end encryption in which data is encrypted before transmitted to a server and then decrypted by the receiving party. An example of cloud-based services that use end-to-end encryption is Tresorit and SpiderOak. Additionally, when using end-to-end encryption, no one can access the stored data except for the owner and his/her authorized users only [22].

Although end-to-end encryption has provided a lot of benefits, there are disadvantages associated with the use of it such as non-recoverability of the lost credentials (username/password). This is caused because the service providers have no access to the encryption keys or password. Another challenge is the failure in detecting spam and phishing files over encrypted files. This can occur because the third party which is the cloud server can not perform spam filter on the stored files due to lack of access to the clear text [8].

3.4 Relational and NoSQL Database

Due to the scalability issues in the traditional database that degrade performance as the data size increases, new systems have been designed to provide good horizontal scalability (ability to distribute both data and load of simple operations over many servers) for the simple read/write database operations. Rick Cattell [2] examined a number of Structured Query Language (SQL) and Not Only SQL (NoSQL) data stores. The NoSQL systems use the CAP theorem which states that a system can have two out of the following three properties: Consistency, Availability,

and Partition-tolerance. The SQL systems provide horizontal scalability without abandoning SQL and ACID (Atomicity, Consistency, Isolation, and Durability) of transactions. The data stores were then categorized into:

(i) *Key-value stores* for storing values and indices to find those values, e.g., Riak and Scalaris.
(ii) *Document stores* for storing documents ("pointerless objects"), e.g., SimpleDB and MongoDB.
(iii) *Extensible record stores* for storing extensible records that can be partitioned horizontally or vertically across different nodes, e.g., HyperTable and Cassandra.
(iv) *Relational database* for storing tuples, e.g., MySQL Cluster and VoltDB.

When concluding, Cattell argued that document stores have the ability to query collections (a grouping of documents) based on multiple attributes value constraints, unlike key-value stores. In addition, some of the systems use RAM for storage which proved to have very poor performance when overflow occurs compared to those designed for disk storage.

4

System Architecture and Design

4.1 System Architecture

In describing the architecture of our system, we will use both physical view and information flow. The physical view, also known as deployment view, shall depict the topology of the software components on the physical layer with the connections between those components while information flow will show how raw information is being processed when transferred from customer's site to Ascom R&D as explained below.

4.1.1 Physical View

The general overview of this thesis work shall involve collection of Unite communications (log data) from Unite CM at the customer's site using Ascom's capturing tool which is a special R&D tool. The collected log files can be from different Ascom systems such as patient monitoring system, paging system or Ascom's mobile

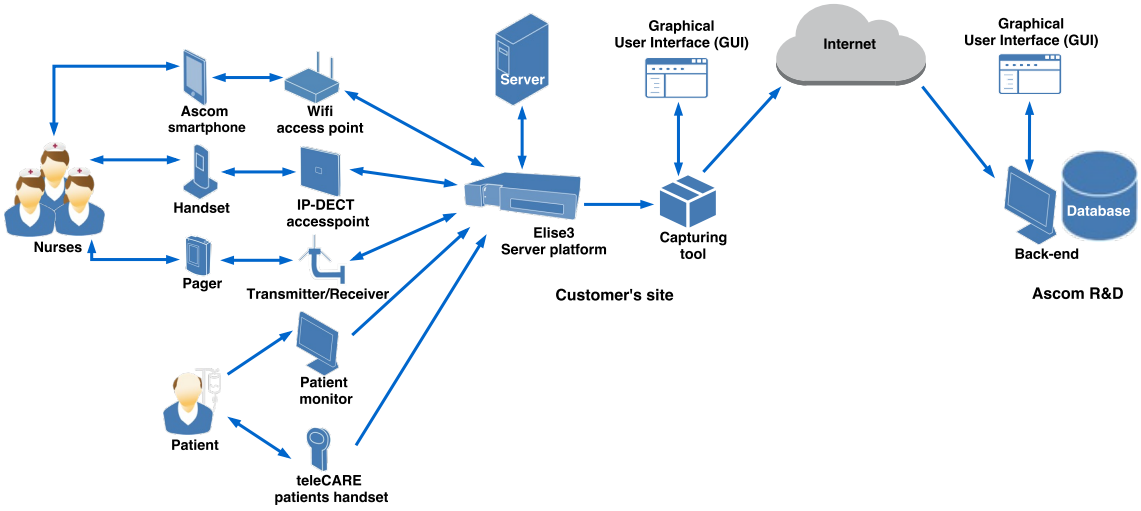


Figure 4.1: Physical view

devices which are intended to alert the nurses of the events that require their immediate attention e.g., potential patient problems. After being collected, the log files will then be transferred from customer's site to Ascom R&D over the Internet as shown in Figure 4.1. The internet can either be a cloud storage service in which the uploaded files are stored in the remote servers for other users to access them or an

FTP application where files are uploaded to an FTP server and can be downloaded through an FTP client.

The Graphical User Interfaces (GUI) at the customer's and Ascom's side allow users to interact with different applications e.g., Buslogger, where they can perform different operations such as viewing the log files, store them locally or process them in the database. Lastly, when the log files reach Ascom R&D, they can either be saved locally or being processed in the database where different types of communication information i.e. paging or alarms, are identified. The log files can also be used to perform further analysis in which different evaluations will be performed e.g., nurse's stress level or travel time between patients.

4.1.2 Information Flow

Figure 4.2 shows how collected log data is processed from the customer's site to Ascom R&D. The log data is first processed by removing or hiding the patient's personal information in order to keep its track without revealing the patient's identities. It is further encrypted before it is sent to Ascom's R&D. The collected log file can then be saved locally at the customer's site. Afterward, the log files shall be compressed and encrypted; then, their integrity is checked before they are sent to Ascom. When sending, the encrypted log file shall either be physically delivered to Ascom R&D or transferred over a secure transmission channel either through a cloud service or FTP application. Once received, the encrypted log file shall have their integrity checked, decrypted, decompressed then saved locally at Ascom R&D. Lastly, the log files will be pre-processed i.e. saved in an appropriate format before stored in the database.

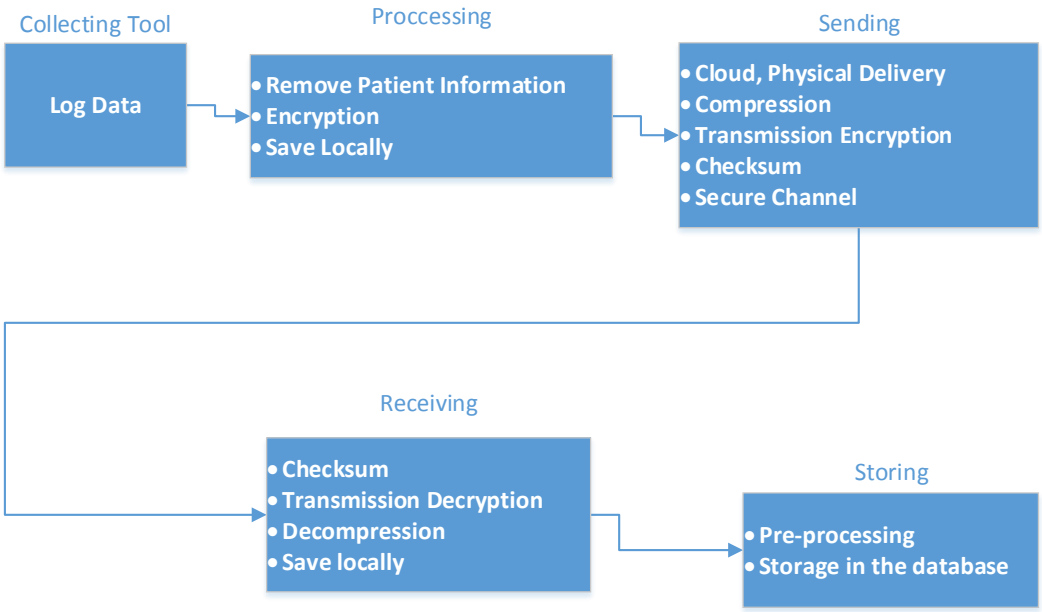


Figure 4.2: Information flow

4.2 System Design

In designing our system, we will consider using solutions or applications that support the use of either client-server architecture or an embedded PC when capturing, transferring, and processing of Unite communications as detailed below.

4.2.1 Client-Server Architecture

Client-Server architecture is a network architecture in which each computer or process on the network is either a client or a server. This architecture is based on both hardware and software components that are designed to communicate across the network [7]. Often servers and clients communicate on separate hardware but may also reside on the same system. Additionally, the client-server architecture brings out a logical perspective of distributed cooperative processing where a client sends requests while a server handles and processes them. A client also known as *front-end application* can be a single-user workstation while a server also known as *back-end application* can be one or more multi-user processors with high capacity of shared memory and ability to support multiple and simultaneous clients requests as shown in Figure 4.3.

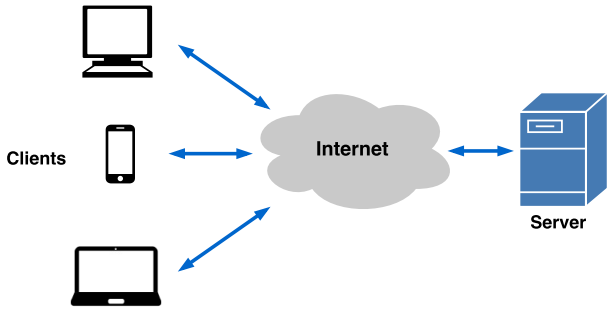


Figure 4.3: Client-Server architecture

If properly implemented, the client-server architecture can provide advantages such as improvement in data sharing, shared resources regardless of hardware platforms, and data processing capability despite the location. Another advantage is the provision of better security since servers have control access to ensure only authorized clients can access it [16]. However, there is a disadvantage associated with the use of client-server architecture such as criticality to failure for the centralized servers which is caused by overloading them with frequent simultaneous requests from the clients.

4.2.2 Embedded PC

Embedded PC is a dedicated computer-based system that can either be part of a larger system, independent or part of a heterogeneous system. Most of the modern embedded control systems have high demanding functions such as real-time machine sensors, visions and motion controls as well as Graphical User Interface (GUI). This

complexity of the embedded systems and fast growth of the PC hardware and software support has led to the adoption of embedded PC technology [4].

Most of the embedded PCs e.g., Next Unit of Computing (NUC), are fully compatible with a standard PC and virtually all desktop and real-time PC Operating Systems (OS) e.g., Windows and Linux. So it is possible to use the general purpose Windows as an embedded OS. Also, these embedded PCs have portable size and ease of installation which make it easy to add devices and scale up rapidly. Furthermore, they consume a small amount compared to a full-sized PC which results in reducing the cost of operating them.

4.3 File Transfer Protocol

In the early days of computing, one had to learn complex sets of commands in order to use the Internet. File Transfer Protocol (FTP) was then invented in the early 1970s as a protocol that transfers files between an FTP host/server and an FTP client computer on the Internet [11]. FTP can either be used to download files from the World Wide Web, upload files to the FTP servers, transfer large files among two parties or distribute the latest versions of programs by software developers. Examples of FTP client programs available for transferring files are *Filezilla client*, *WinSCP*, and *SmartFTP*. For communication to take place between a server and a user, FTP must connect using two TCP ports: *command* and *data* port. Command port e.g., Port 21 or 990, is the main TCP port that is created upon a session for passing commands and replies. Data port is used to establish data connections for transferring files or directories between server and client, and once the transfer is complete, the connection is closed.

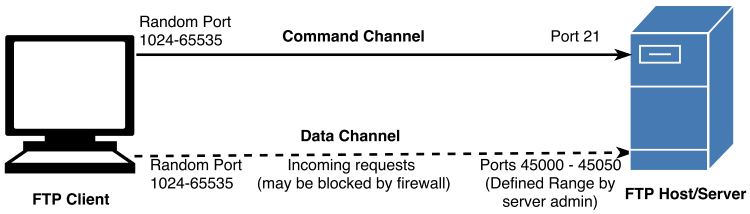


Figure 4.4: Passive FTP: Client sets both port connections to server

FTP transfers files between systems using one of the two active/passive connection modes: *binary* which transfers images, zip files or executable files in binary form and *ASCII* for transferring text and HTML. When the connection mode is passive, FTP client initiates a connection to the command or data port to the host server as in Figure 4.4. But in active mode, the FTP client initiates the connection by connecting to the server's command port (port 21) then opens a listening data port and sends the number of its own command port to the server, Figure 4.5.

Additionally, FTP connections are usually not encrypted, but some FTP servers may offer or require encrypted connection to secure the data when it is being transferred between systems. The types of encryption in FTP include implicit SSL,

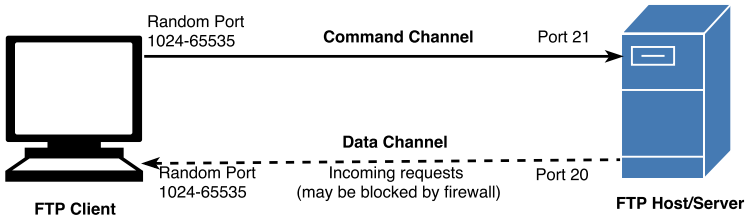


Figure 4.5: Active FTP: Client sets command port, Server sets data port

explicit SSL and Secure FTP (SFTP). In implicit SSL, secure communication is set up at the beginning of the connection where SSL supported clients are allowed access. When using explicit SSL encrypted connection, unencrypted FTP connection is established and can be upgraded to a secure connection when sensitive data is requested to be sent. In this encryption, both secure and non-secure clients are allowed access. Furthermore, when using SFTP encryption, secure shell connection is used for transferring data between computers and encrypted public key is required for authentication.

4.4 Cloud Storage Services

The cloud concept has roots dating back to the 1950s and 1960s, but it was not able to take off due to technological limitations such as internet data speed and computer hardware which could not support the amount of data to be sent and received [15]. Cloud Storage is a service in which data can be remotely maintained, managed, backed up, and made available to users from any location via the internet. Many of the cloud storage services upload files to the external servers which give users ease and convenience but can be costly. Most of these services are free up to a certain number of gigabytes (GB), but one can request additional storage for a monthly fee. All cloud services provide synchronization of folders and files, drag-and-drop accessing, and collaboration of users on documents. Examples of these cloud storage service providers are *Dropbox*, *Google Drive*, *Box*, and *Microsoft OneDrive* [5].

Furthermore, cloud storage has an architecture that is based on delivery of storage on demand in a multi-tenant and highly scalable way. This architecture consists of the front-end layer that exports an Application Programming Interface (API) to access the storage e.g., at the client system. Behind the front end is the middleware layer (storage logic) which implements features such as replication and data reduction e.g., at a web server. Lastly, there is a back-end layer that implements physical storage of data e.g., database as shown in Figure 4.6.

Moreover, cloud storage providers offer cloud encryption services that transform text or data to ciphertext by using encryption algorithms before being placed on a storage cloud. The encryption capability offered by providers must match the level of sensitivity of the data being hosted because it consumes more processor overhead and become expensive for the customers. To overcome that, providers have offered an alternative technique which includes *redacting* or *obfuscating* data where vendors

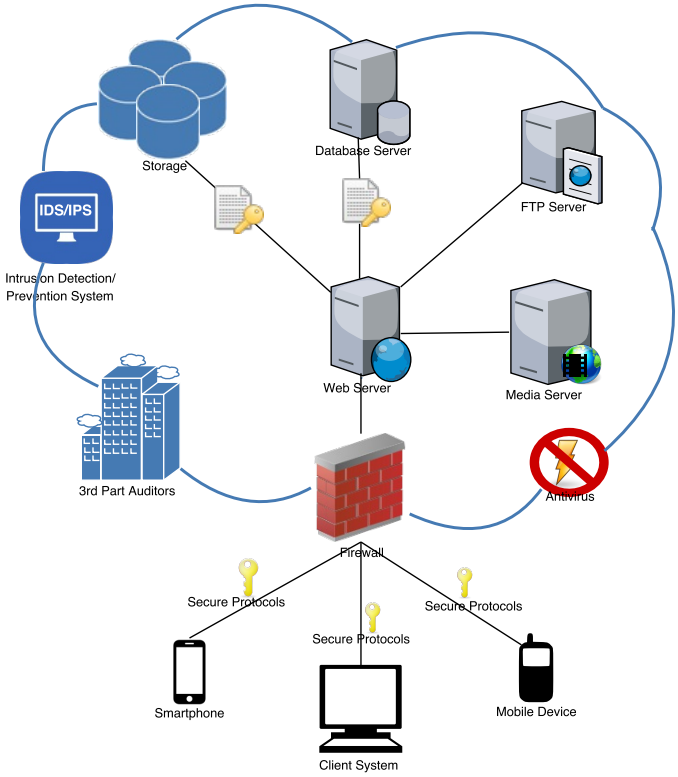


Figure 4.6: Cloud storage architecture

use their proprietary encryption algorithms on their data before sending to the cloud.

There are some advantages associated with the use of cloud storage services such as easy accessibility of the stored files despite the location and disaster recovery since it can be used as backup storage for the important files. However, there are also disadvantages of cloud storage such as the need of Internet connection in order to access the files. Another disadvantage is the limitation in bandwidth as some cloud storage services have a specific bandwidth allowance. In addition, there are concerns about the safety and privacy of importance data due to the possibility of intermixing private data with other organizations.

5
Specifications

In this chapter, we specify all the requirements necessary for implementing our system. This will involve the use of use case diagrams and their descriptions for both the customer's site and Ascom R&D as explained below:

Actor	Use case
Nurse	Message • Alert/Paging • Text message • Text message
	Alarm
	Call • Normal call • Assistance call • Emergency call
	Nurse login • Activity logging
Buslogger	Send subscription
	Retrieve subscription
	Store retrieved data • Encrypted format • Plaintext (limited information)
	Open saved log file • Real time view • Analyse view • Real time snapshot
	Select what to log
System administrator	Login • Perform updates • Performing configurations – Basic configurations – Advanced configurations

Table 5.1: Actors and use cases involved in Unite System

5.1 Use Case Diagrams

The use case diagrams will portray different types of users of the Unite System and the various ways that they interact with the Unite CM. The Unite CM contains different actors who interact with each other through use cases as shown in Table 5.1.

5.1.1 Nurse

The nurse interacts with other staff members using their carrier devices e.g. Ascom smartphones and IP-DECT handsets, as shown in Figure 5.1. The following operations can be performed by the nurse through the Unite CM at the customer's site:

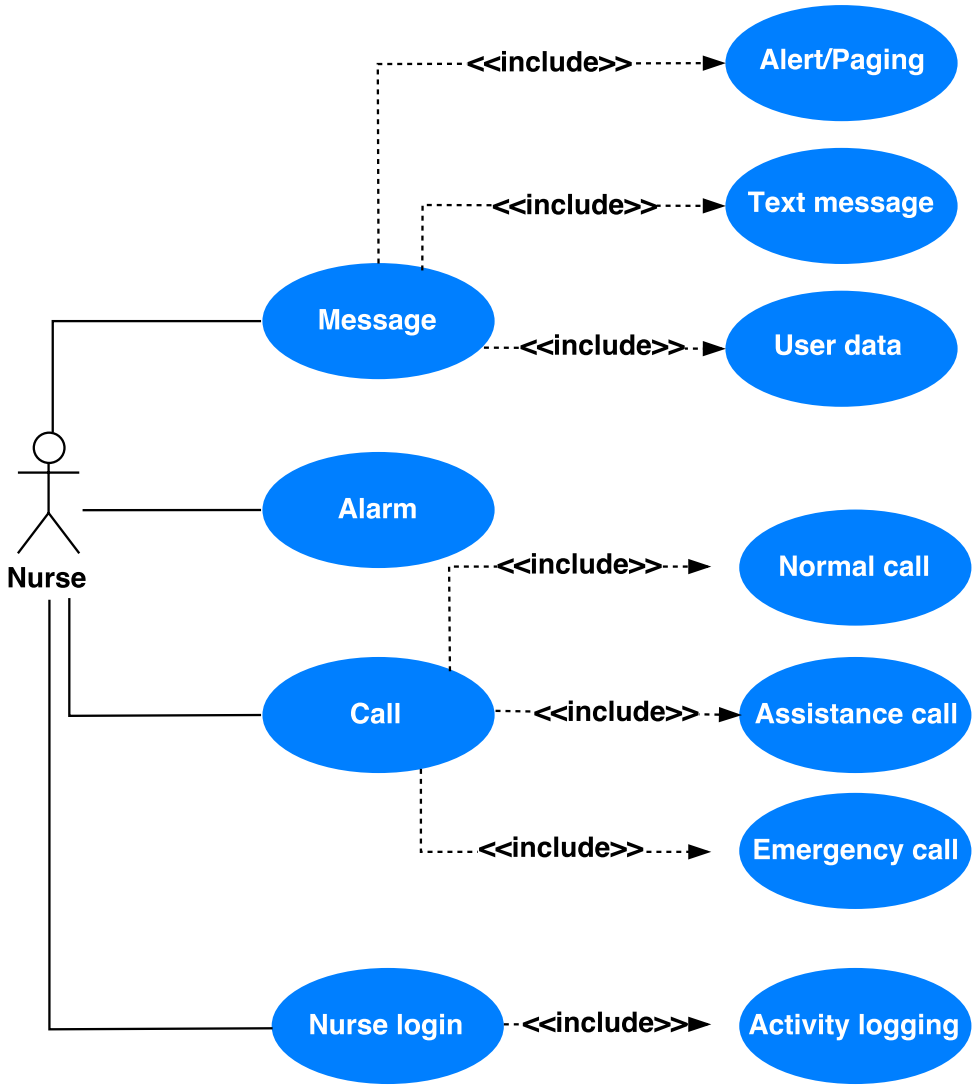


Figure 5.1: Nurse - Use case diagram

(i) Message - The Unite System allows the nurse to send or receive messages from other staff members through the Unite CM to the carrier systems. The message can be one of the following:

- *Paging* - A paging is a message sent and displayed immediately and it is usually displayed in the nurse's paging system.

- *Alert* - An alert is an indication notifying the nurse about something that has happened and it requests his/her action. It is sent by applications e.g., patient monitoring system.

- *Text message* - This is the message with the user (nurse) as a sender or can be a continuous dialog between two users e.g., Interactive Message (IM).

- *User data* - Information about the data sent from a handset e.g., VoWiFi or DECT handset, by the user (nurse) within the system. The most common fields in the user data block include handset address and input data.

(ii) Alarm - The Unite System should also allow the nurse to receive alarms generated by medical equipment through the Unite CM.

(iii) Call - The Unite System allows the nurse to make calls to other staff members using their unique call ID or receive calls from patients that are distributed by the nurse call system. The calls can be:

- *Normal call* - Example being when a patient presses a red button on patient handset (bedside handsets) requesting the nurse's presence.

- *Assistance call.*

- *Emergency call* - Example being when a patient has fallen or has not moved for a longer period and it's usually automatically triggered by a moving sensor or angle detection device.

(iv) Activity logging - Activities such as duty assignment e.g., work shifts for nurses with their respective assigned day of the week and time is mostly performed by the head nurse. For different activities to be logged, it requires the head nurse to log into the system. Additionally, the head nurse can also make changes to the existing shifts.

5.1.2 Buslogger

The Buslogger is a Windows-based troubleshooting tool developed by Ascom to collect and store Unite communications from the Unite CM at the customer's site. The Unite System also allows the Buslogger to send a subscription to the Unite CM in order to access the Unite communication. As shown in Figure 2, for the Buslogger to retrieve collected logs from the Unite CM, it has to send the subscription in order to get the specified logs. Additionally, after logs are retrieved from the Unite CM, they can be stored by the Buslogger either in encrypted format, where the messages are presented in XML format, or in plaintext where only limited information is saved. Once stored, the log files can be opened in the Buslogger at which the display can be in real time view in which different fields of the message are displayed e.g., source address or status of the message. Also, the log files can either be opened in real-time snapshot where the XML format of the messages is shown or analyze view in which the analysis of different parts of the block message is displayed e.g., USD sender or receiver.

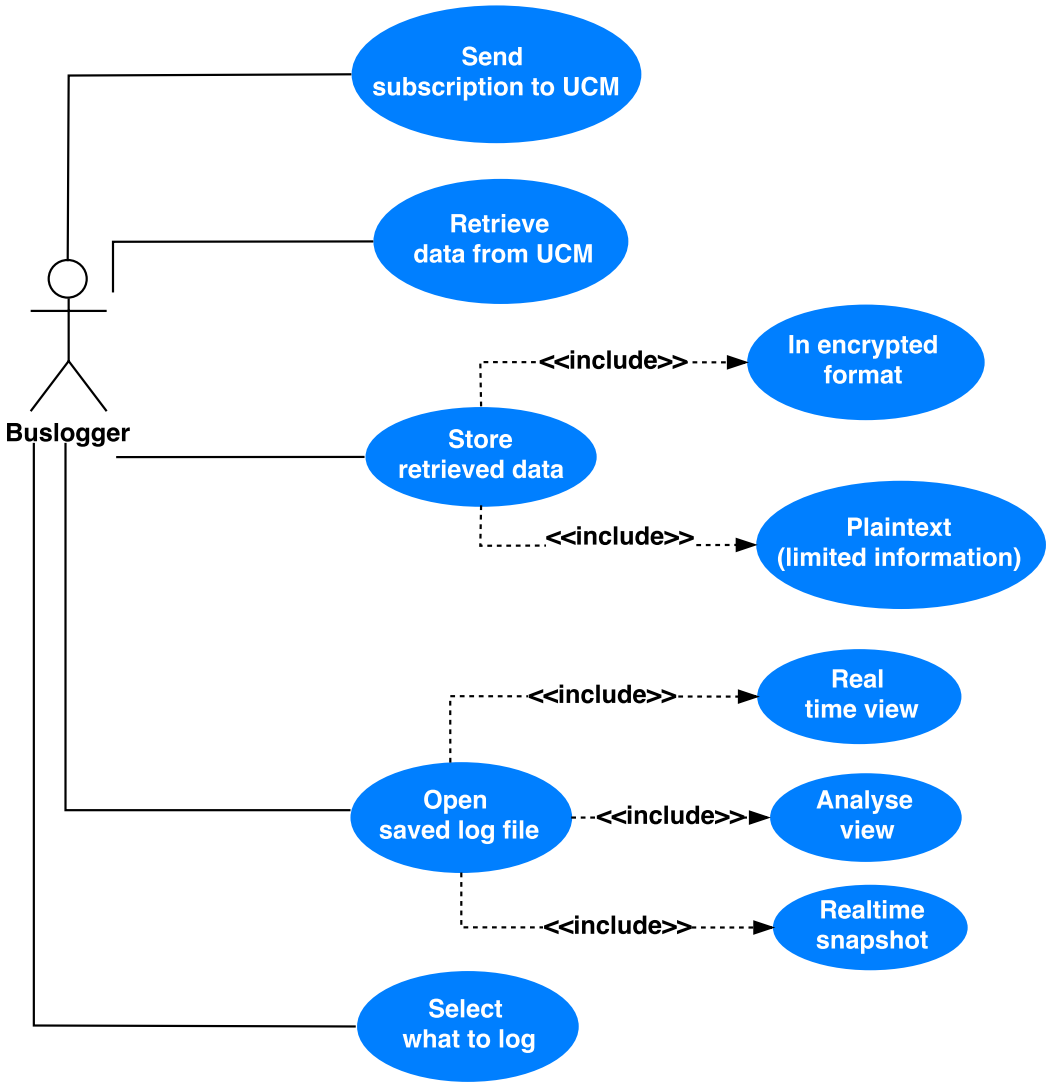


Figure 5.2: Buslogger - Use case diagram

5.1.3 System Administrator

The Unite System allows the system administrator to perform the following actions that require him/her to log into the system as described in Figure 5.3:

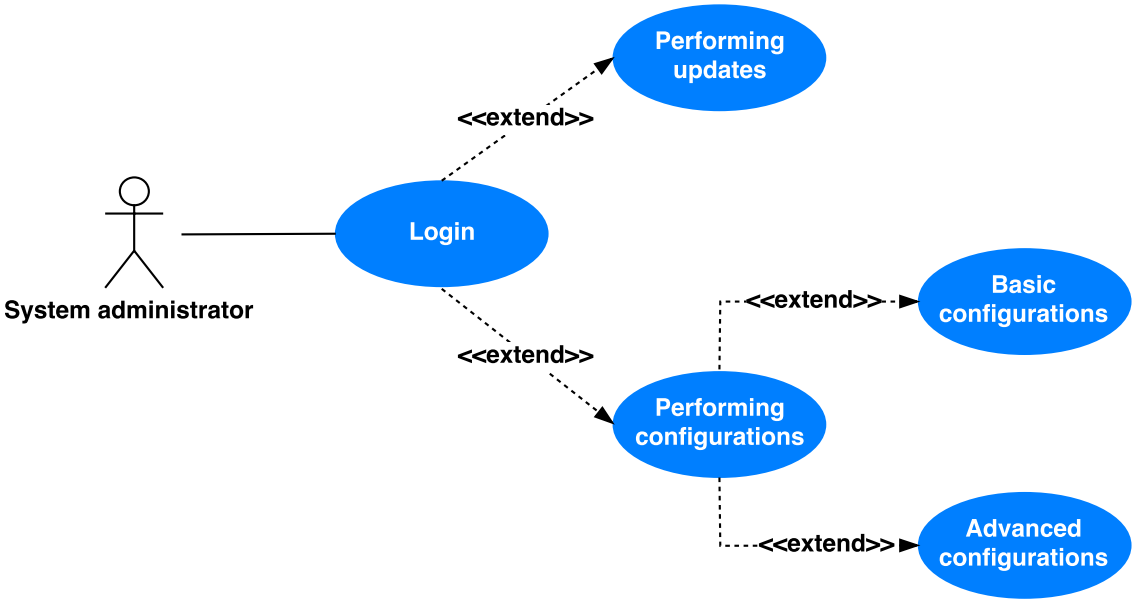


Figure 5.3: System administrator - Use case diagram

- Performing updates - This involves correction and improvement of existing functionalities in the system which also includes documentation of the additional information.

- Performing configurations - The system administrator performs a number of configurations in the Unite CM. This includes basic configurations such as adding users to enable messaging between the added users through their handsets and group management for broadcasting and multicasting. There are also advanced configurations e.g., event handling, remote management configurations or server setting.

5.2 Requirements

The requirements for a system are the descriptions of what the system is supposed to do and services it should provide. The following are the functional and non-functional requirements of our system at both customer's site and Ascom R&D.

5.2.1 Functional Requirements

R1.1	It shall be possible to install an application for collection of log data.
R1.2	The installation shall be according to normal Windows installation packages.

R1.3	If the installed application has to be removed, it should be possible to uninstall it without leaving traces (the application shall be as nonintrusive as possible).
R1.4	No certain skills shall be required to the customers when installing/uninstalling the application.
R2.1	When collecting Unite communications, it shall be possible to limit the size of log data that is to be retrieved from customer's site using a selected tool.
R2.2	Prior to collecting Unite communication from customers, it shall be possible to initiates logging activity through sending a subscription to the Unite CM using selected tools.
R2.3	After logging, it shall be possible to stop logging through retrieving log data from Unite CM using a selected tool.
R2.3.1	Also, it shall be possible to check the progress of the retrieved logs (in presence of errors, restart activity logging).
R2.4	After log files have been retrieved, it shall be possible to save them locally.
R2.4.1	Patients' personal information shall be replaced/removed e.g., through hashing algorithm.
R2.4.2	The log data retrieved from customer's site shall be saved in encrypted format with its contents in XML format.
R2.4.3	It shall be possible to save retrieved log data as log files (text format) which have limited information about the collected logs.
R3.1	It shall be possible to select a tool that will be used for transferring log data to Ascom R&D.
R3.1.1	The selected transfer application or service shall be able to secure the captured log files during transmission.
R3.2	The selected transfer tool shall be able to be installed either on an embedded PC or on the client's server.
R3.3	After installation of transfer tool, it should be possible to transfer collected log files to Ascom R&D.
R3.3.1	When the transfer is complete, the status of transfer and integrity of log files shall be checked.
R4.1	After successful transmission, it shall be possible to download them.
R4.2	After log files have been made available at Ascom R&D, it shall be possible to perform decryption of the files which can be through the use of the Buslogger.
R4.2.1	After a successful decryption, the decrypted files shall be saved locally at Ascom R&D.
R4.3	Once log files have been made available for storage, select a storage solution i.e. relational or non relational database.
R4.4	Install the selected storage solution at Ascom R&D.
R4.5	After installation of storage solution, import decrypted files for storage in the database.
R4.6	After log files have been stored, present analysis of the stored files to describe the nature and relation of data to be analyzed.

5.2.2 Non-Functional Requirements

R1.1	The implemented solution should not affect the operations of other systems at the customer's site.
R1.2	The transfer application or storage solution selected shall not introduce complexities, vulnerabilities or delays to the existing systems.
R2.1	The selected applications or solutions shall be compatible with the existing operating system.
R3.1	The capturing, transferring and storage solutions shall be available when needed i.e. they should perform their operations in a timely manner.
R4.1	The implemented solution should be easy and fast to use.

6

Implementation

6.1 Data Capturing from Unite CM by Buslogger

In capturing Unite communication from Unite CM at customers side, we plan on either performing configurations of the Buslogger tool on an embedded Windows PC, e.g., Next Unit of Computing (NUC), at Ascom R&D or have the Buslogger installed on client's server. If we use an embedded PC, it shall be located at customer's site so that at run time the embedded application program (Buslogger) can capture all Unite communications. After being captured, the log files will be encrypted by the Buslogger then saved locally.

6.2 Transmission of Unite Communications

After capturing the Unite Communications, we need to send it to Ascom R&D. We will investigate three different ways to transfer the Unite Communication to Ascom Cloud, FTP, and Physical Delivery.

6.2.1 Online Cloud Services

Online cloud services are the services that offer storage of files in the cloud server. These services can be accessed by using the web service application programming interface (API) or applications that utilize API. The following are the online cloud services that we tested:

6.2.1.1 Globus

Globus moves data between two GridFTP servers or between a GridFTP server and a user's machine. It can use either web interface or command line interface (CLI) to transfer files. It also offers a feature called Globus Connect Personal which enables users to move files to or from a laptop or desktop computer or other endpoints. Endpoints are set up on the Globus system for transferring files to and from the Globally Accessible Data Environment (GLADE) disk storage system. GLADE file spaces are intended to be used as work areas for day-to-day tasks. These storage spaces are available by default except for project space as shown in Table 6.1.

File space	Limitation	Backup	Description
Home: /glade/u/home/username	10GB	Yes	User home directory
Scratch: /glade/scratch/username	10TB	No	Temporary computational space
Work: /glade/p/work/username	512GB	No	User work space
Project: /glade/p/project_code	N/A	No	Project space allocations (via allocation request)

Table 6.1: Globus globally accessible data environment (GLADE)

- **Storage Capacity & Cost:** ranging from 10s of GBs to 10s of TBs. It needs an institutional subscription; however, the cost is not mentioned on their website.
- **Maximum file size:** No limitation on file size.
- **Operating system:** Windows, Mac OS X, and Linux.
- **Security:** Data encryption during Globus file transfer (but sometimes encryption may fail to be supported by an endpoint).
- **Other features:** performance monitoring, retrying failed transfers, recovering from faults (whenever possible), and transmission status reporting.

Limitation: We were not able to move date between two personal computer or to store the data at GLADE disk storage system because it requires *Globus Plus* subscription, which in turn requires Ascom to have an institutional subscription with Globus. Additionally, we could not connect Globus Connect server to GridFTP Server, because it requires a GridFTP server running on a Unix platform [14], and that is out of the thesis scope.

6.2.1.2 Tresorit

Most of the cloud storage services use server-side encryption and users have no control over how the encryption is performed and who can access the decryption keys. This limitation led to the use of cloud services that employs client-side encryption in which users encrypt their files then upload them to the cloud. Tresorit is an online cloud service that stores and shares files and uses an end-to-end encryption (client-side encryption) to guarantee security of the uploaded file.

- **Storage Capacity & Cost:** It required a minimum of two users and cost €20 / user / month with 1,000GB of storage for each user.
- **Maximum file size:** file size can be up to 10GB.
- **Operating system:** Windows, Mac OS X, and Linux.
- **Security:** Tresorit encrypts files using AES-256 client-side encryption before uploading, and they remain encrypted till they reach the recipient. In au-

thenticating the user, it uses HMAC (SHA-512) to authenticate the password by comparing it with the stored salted password. Also, it supports two-step verification which is a login authentication feature that enables the addition of another layer of security to the account.

Tresorit had an acceptable upload speed up to 11Mb/s when we tried it on a 100Mbs Internet connection. The unique feature is the end-to-end encryption and decryption support.

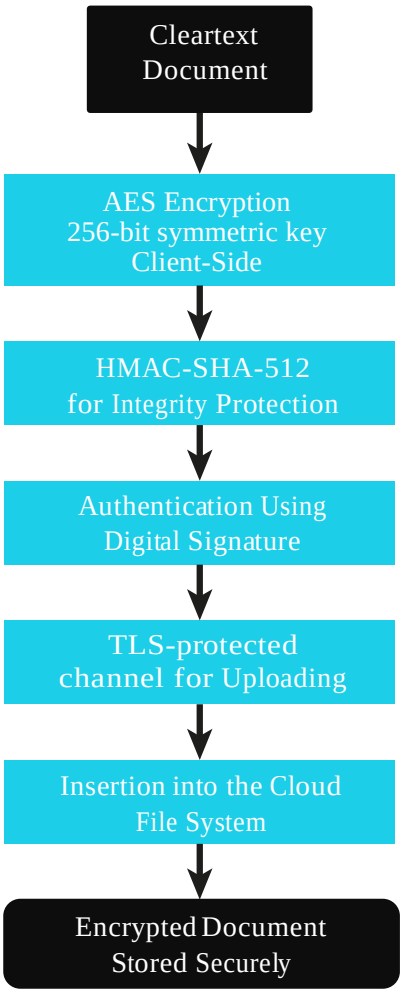


Figure 6.1: Tresorit security of file uploading.

Tresorit Secure Cloud Model: as shown in Figure 6.1, when a user uploads a file using the Tresorit client application, the file will be encrypted using AES encryption algorithm in CFB mode using a new 256-bit symmetric key chosen by the client application. The integrity of the file is guaranteed by using HMAC-SHA-512. After that, a TLS tunnel will be established to the cloud server for uploading the encrypted file. This adds an additional layer of protection to protect the file against eavesdropping and tampering during the upload process. Finally, a digital signature is used to authenticate each upload [22].

Tresorit End-to-end Encryption: Tresorit provides a secure cloud storage by using end-to-end encryption in which encryption and decryption process is done on the client-side [22]. In this type of encryption, no one can access the stored data except for the owner and authorized users only. Figure 6.2 shows the traditional cloud storage security in which encryption and decryption process is performed in the cloud while Tresorit end-to-end encryption shown in Figure 6.3 is performed at the client's side (Alice encrypt the file and Dave, Carol, and Bob decrypt it).

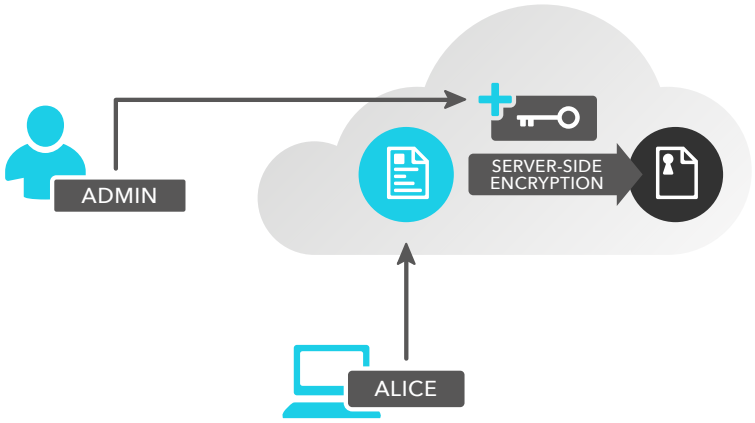


Figure 6.2: Traditional cloud storage security [22].

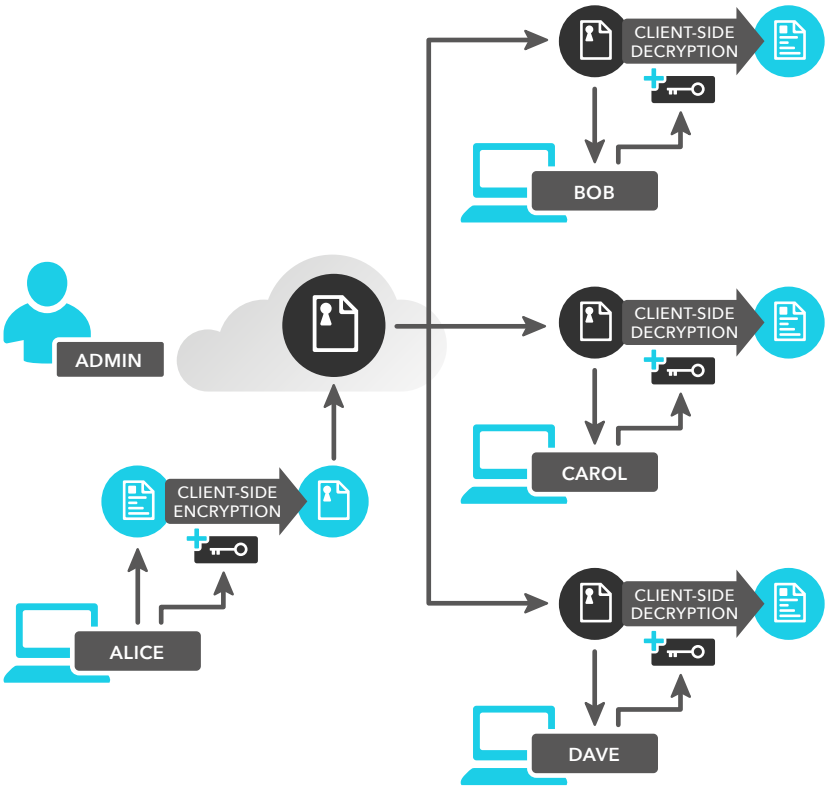


Figure 6.3: Tresorit end to end encryption and decryption [22].

The encryption algorithm is AES256 in Cipher Feedback mode (CFB). CFB is a

block cipher deterministic algorithm which works with fixed length block of bits. It needs an initial input block to operate called Initialization Vector (IV). Each file and each file version get a new random 128-bit IV to ensure a different encryption output even if the same data encrypted several times with the same key [6] [12] [21].

Client-side integrity protection: Message Authentication Code (MAC) is applied by Tresorit client to each file's content using a key known only by the owner and by the others who have access to the file, however, the key is not known by the server. A variation of MAC (keyed-Hashing for Message Authentication) HMAC-SHA512 is used in combination with a new random key for each file to ensure its integrity [21].

Zero-knowledge authentication: Users passwords are stored only as a salted hash format at the server. When a new user registers, a 160-bit cryptographic random salt is generated by the Tresorit client. The combination of the user password and the salt is iterated with the Password-Based Key Derivation Function 2 (PBKDF2) ten thousand times. PBKDF2 executes the input inside a pseudorandom function for a fixed number of times x to derive a key, and it is considered as one of the most used algorithm to manage users passwords [1]. The result of the iteration with the salt is sent to the server when the user clicks the sign-up button and stored there. When the user sign in, a challenge-response protocol is used as following: First, The server sends to the client, the stored salt, and a challenge request. Second, the client, using the received salt, calculates the derived key with PBKDF2. The HMAC of the salt and the challenge with the key calculated from user password are sent back as a response to the server. Third, The response is compared to the key stored in the database at the server [21].

Compliance with HIPAA: Tresorit security is compliant with the Health Insurance Portability and Accountability Act (HIPAA) standards because it runs in Microsoft Azure datacenters, which in turn meets the HIPAA standard [18] [21].

Advantage: When testing Tresorit, it was possible to share files between computers through uploading files to the shared folder that can be accessed by others.

Limitation: One limitation of using Tresorit is that it requires an account that is part of a business account of an organization. So, users can not use their personal account to register.

6.2.1.3 Google Drive

Google Drive is a file storage and synchronization service that offers storage of files in the cloud, sharing of files and allowing collaborative editing of documents and files between users. It assigns online storage space to its users that are accessed across different devices, e.g., smartphones and laptops.

- **Storage Capacity & Cost:** 15GB of storage for free but more storage needs monthly subscription as shown in Table 6.2.

Storage	Price
15GB	Free
100GB	$1.99 per month
1TB	$9.99 per month
10TB	$99.99 per month
20TB	$199.99 per month
30TB	$299.99 per month

Table 6.2: Google drive storage capacity & cost

- **Maximum file size:** file size can be up to 5TB.
- **Operating system:** Windows, Mac OS X.
- **Security:** It uses secure socket layer communication (HTTPS, SSL) by default to prevent man-in-the-middle attacks. It is not so secure since it does not provide end-to-end encryption also known as client-side encryption. That leaves user's information unsecured and could be accessed by Google or by unauthorized users.
- **Other features:** Require a Google client software to be running on the user's computer (at both customer's site and Ascom R&D)

Google Drive worked smoothly and continuously without any problem. It had an outstanding upload speed up to 50Mbs when we tried it on a 100Mbs Internet connection.

Advantage: It was possible to automatically upload new files with a high speed. Furthermore, it was easy to install and set it up to work on port 433 which is open on most firewalls.

Limitation: It performs encryption and decryption process at the cloud with the encryption key is owned by Google which imposes security threats to the stored confidential files when someone manages to gain access to the systems. Also, it does not support the recovery of deleted files.

6.2.1.4 Dropbox

Dropbox is a cloud storage service also known as online backup service that offers file sharing, cloud storage, file synchronization, and personal cloud. Dropbox has a client program (Dropbox desktop application) that enables users to drop any file in a designated folder and automatically upload it to Dropbox's cloud-based service (Dropbox server).

- **Storage Capacity & Cost:** It is free up to 2GB storage capacity, a Pro account with 1TB of storage space for €9.99 / month, or a Business account with 2TB of storage for €12 / user / month.
- **Maximum file size:** file size can be up to 20GB.

- **Operating system:** Windows, Mac OS X, and Linux.
- **Security:** Dropbox uses SSL/TLS to protect data during transit between Dropbox client and server (secure tunnel) and store data using AES-256 encryption in which Dropbox's own encryption keys are used. Additionally, it supports two-step verification which is a login authentication feature that enables addition of another layer of security to the account.
- **Other features:** Keep deleted files for 30 days.

Dropbox worked smoothly and continuously without any problem. It had a good upload speed up to 25Mbs when we tried it on a 100Mbs Internet connection.

Advantage: It was possible to automatically upload new files with good speed as well as keeping deleted files for 30 days. Also, it was easy to install, setup, and operate on both port 80 and 433 which are open on most firewalls.

Limitation: Same as Google Drive, Dropbox supports encryption and decryption of files at the cloud with the encryption key owned by Dropbox which impose security threats to the confidential files stored in the cloud.

6.2.2 FTP

FTP is a network protocol used to transfer files between clients and servers on a network. In this section, we will look into FTP solutions i.e. Filezilla and WinSCP. In testing both solutions, files were transferred between FTP clients (Filezilla client and WinSCP) and a Filezilla server as described below:

6.2.2.1 FileZilla

Filezilla a cross-platform FTP application that allows users to transfer files from the local computer to the remote computer. Filezilla is available as a client and server version with the following features:

- **Cost:** Open source.
- **Size:** Supports transfer of large files with no limitation.
- **Language:** C++.
- **Operating System:** Windows, Linux, BSD, and Mac OS X.
- **Security:** Supports FTP, FTP over SSL/TLS (FTPS and FTPES), and SSH File Transfer Protocol (SFTP).
- **Other features:** FileZilla supports resume when transferring files, allows remote file search on the remote server, and supports configurable transfer speed limit. Other features are support keep-alive to check connection status, IPv6, and performs compression with DEFLATE.

Limitation: One limitation of using Filezilla is it requires manual restarting if the automatic file uploading crashed during file transfer. Another limitation is that Filezilla does not support automatic file upload.

6.2.2.2 WinSCP

Window Secure Copy (WinSCP) is a free and open source SFTP client, FTP client, SCP client, and WebDAV client for Windows which are used to securely transfer files between a local and a remote computer [23]. WinSCP has the following features:

- **Cost:** Open source.
- **Size:** Supports transfer of large files with no limitation.
- **Language:** C++.
- **Operating System:** Windows, Linux, BSD, and Mac OS X.
- **Security:** It supports the following transfer protocols: FTP, FTP over SSL/TLS (FTPS), Secure Copy Protocol (SCP), Web Distributed Authoring and Versioning (WebDAV), and SSH File Transfer Protocol (SFTP). These protocols use either Secure Socket Shell (SSH) or TLS/SSL which provide a secure way to access a remote computer that results in guaranteeing the integrity of the data being transferred.
- **Other features:** WinSCP supports the use command-line interface when transferring files, automatic upload of new files in a specified folder, and resume when transferring files over FTP and SFTP.

When testing WinSCP to transfer log files between two computers, it was possible to establish FTP over TLS (FTPS) connection from WinSCP client to FileZilla Server using 4096 bit key.

Advantage: WinSCP supports automatic file uploading which worked successfully when tested with WinSCP watching a local directory on the local machine and uploaded any changes to the server.

Limitation: One limitation of using WinSCP is it requires manual restarting if the automatic file uploading crashed during file transfer.

6.2.3 Physical Delivery

Some customers do not allow any Internet connection to their system. To overcome that problem, we need to use the Embedded PC solution. In this solution, we will save the captured Unite Communications to the Embedded PC's hard drive. The customer will send back the Embedded PC later to Ascom R&D, which in turn will extract the saved captured data from the hard drive.

6.3 Storage of Communication Information

In this section, we describe and explain different solutions that were deployed in transforming and storing collected data. The collected log data can be saved locally to the hard disk, but for our solutions, we used database due to its ability to protect its contents through the use of access control on its users. A database can be defined as an organized information that can easily be accessed, managed, and updated by a computer program. In testing database solutions, we tested both relational database, e.g., Microsoft SQL Server 2014 and PostgreSQL, and NoSQL database, e.g., Neo4j that we might use to show some analysis of the collected log files.

6.3.1 Relational Database

6.3.1.1 Oracle Database

Oracle database is a collection of data that is treated as a unit. The database has both logical and physical structures that can be managed separately. The logical structures include data blocks, extends and segments while physical structures include datafiles, redo log files and control files. Oracle database has the following features:

- **Query language**: The supported query languages are SQL, PL/SQL, Java, and C.
- **Operating system**: It can run on either Windows, Linux, or Solaris SPARC.
- **Security**: Oracle database resource security is based on Access Control List (ACL) mechanism that restricts access to information based on privileges. This restriction helps in preventing unauthorized database access and unauthorized access to schema objects.

In testing this solution, we used Oracle Database 11g Release 2, Enterprise Edition to store information and Oracle SQL Developer which is an Integrated Development Environment for performing database tasks. In the end, we managed to get XML data into the database as XMLType datatype, but we had the following advantage and limitation:

Advantage: It provides native XML support by encompassing SQL and XML data models (XMLTYPE) in interoperable manner. This helped in processing the collected log data which is in XML format.

Limitation: When testing this solution, we found out that Oracle does not provide direct support for importing data from XML file. So, in order to get XML data into the Oracle database, one has to create an XML column in the created table. Create the XML tags for the XML column then load the values of the tags from the existing table. After that modify the contents of the XML column (by loading the existing XML data into that column).

6.3.1.2 Microsoft SQL Server 2014

SQL server is a relational database management system (RDBMS) from Microsoft that is designed for managing and storing information. Microsoft SQL Server supports the following features:

- **Query language**: Transact-SQL (T-SQL).
- **Size**: SQL Express 2014 has a database size limit of 10GB and requires 4.2GB of disk space.
- **Operating system**: It is supported on Windows only.
- **Security**
  Data encryption: It supports the use of symmetric key algorithms and keys such as DES, 3DES, RC2, RC4, 128-bit RC4, DESX, 128-bit AES, and 256-bit 256.
  Authentication: It supports Windows authentication mode: Using Windows user and group accounts to log in to SQL Server. Also, it supports mixed mode authentication through username and password pairs which are maintained within SQL Server.

When testing our solutions, we used Microsoft SQL Server 2014, Express Edition for storing imported log files. Also, we used SQL Server Management Studio as an integrated environment for accessing, configuring, managing, administering, and developing all components of SQL server. Additionally, It was possible to import the XML file into SQL database and store them in tables or as XML links but with limiting the XML tags in each message. However, there was pros and limitation associated with the use of SQL Server as detailed below.

Advantage: It supports Native XML feature which helped in preserving xml content of the log data.

Limitation: Microsoft SQL Server Express required large storage space as it supports one physical processor (Intel compatible), 1 GB memory, and 10 GB storage. Also, the log data collected was in XML format with a different number of tags, so SQL server 2014 did not support messages with a different number of XML tags.

6.3.1.3 PostgreSQL

PostgreSQL is an Object-Relational, fully featured and free to use Database Management System. It can be used in a client/server environment [13]. Its client program cannot access data directly even if they are running on the same computer as the server process. A network can be used to separate clients from the server, e.g., a client program can run on Windows while the database on UNIX as shown in Figure 6.4.

When using the PostgreSQL database to process log files, we ran both the client and the server on the same Windows computer. The PostgreSQL Version 9.5.3 was

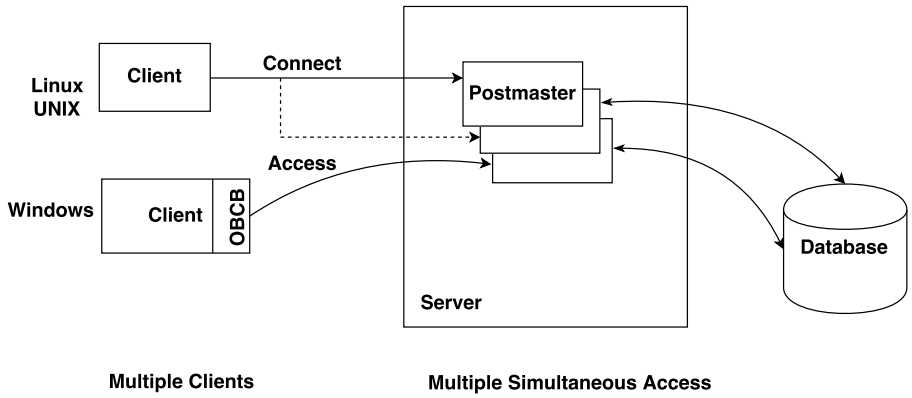


Figure 6.4: PostGreSQL Architecture

tested in which Command Line Interface (CLI) was used to create the database and run the queries. The following are the features of PostgreSQL:

- **Query language**: Structured Query Language (SQL).
- **Size**: Some general PostGreSQL limits are included in Table 6.3:

Limit	Value
Maximum database size	Unlimited
Maximum table size	32TB
Maximum row size	1.6TB
Maximum field size	1GB
Maximum rows per table	Unlimited
Maximum columns per table	250-1600 depending on column types
Maximum indexes per table	Unlimited

Table 6.3: PostGreSQL size limit

- **Operating system**: It runs on both Windows and on any UNIX-like platform including UNIX-like systems such as Linux, FreeBSD, and Mac OS X.
- **Security**: It supports data encryption and Kerberos V5 authentication.

Advantage: One advantage of using PostgreSQL is there is no associated licensing cost for the software which makes it more profitable for the business models with wide-scale deployment.

Limitation: Limited XML support - PostGreSQL does not provide comparison operators for XML data type, so we could not search and find rows by comparing an XML column against a specific search value.

6.3.2 Non Relational Database

6.3.2.1 Neo4j

Neo4j an open-source NoSQL and ACID-compliant (Atomicity, Consistency, Isolation, Durability) transactional database with native graph storage and processing [20]. It is an embedded and disk-based engine that stores data structured in graphs rather than tables. This database stores everything in the form of either an edge, node or attribute. The version of the Neo4j that was tested was 3.0, Community Edition and had the following features:

- **Query language**: Cypher Query Language.
- **Size**: Data size is limited by the address space of the primary keys for nodes, relationships, properties and relationship types.
- **Operating system**: It runs on both Windows and Linux.
- **Security**
  Authentication and authorization: We had to supply authentication credentials (username/password) when accessing Neo4j database.
  Encryption: Neo4j supports SSL encrypted communication over HTTPS. When the server starts, it automatically generates a self-signed certificate and a private key, or the client can provide their own key and certificate for the server to use.

It was possible to show the relationship between nodes and their properties through the comma separated values files (CSV files) that were imported into Neo4j database.

Advantage: It supported schema-free database and was easy to present the connected data.

Limitation: It was not possible to view all the nodes, relationship and properties from the created graph at the same time.

6.4 Proof Of Concept (POC)

In demonstrating the feasibility of the selected solutions, we used a combination of both embedded and client-server solutions. This assisted in the having a client program running on an embedded PC which resulted in minimizing customer's involvement and avoid affecting the functionality of other systems at the client's server. During the implementation the following applications or solutions were selected:

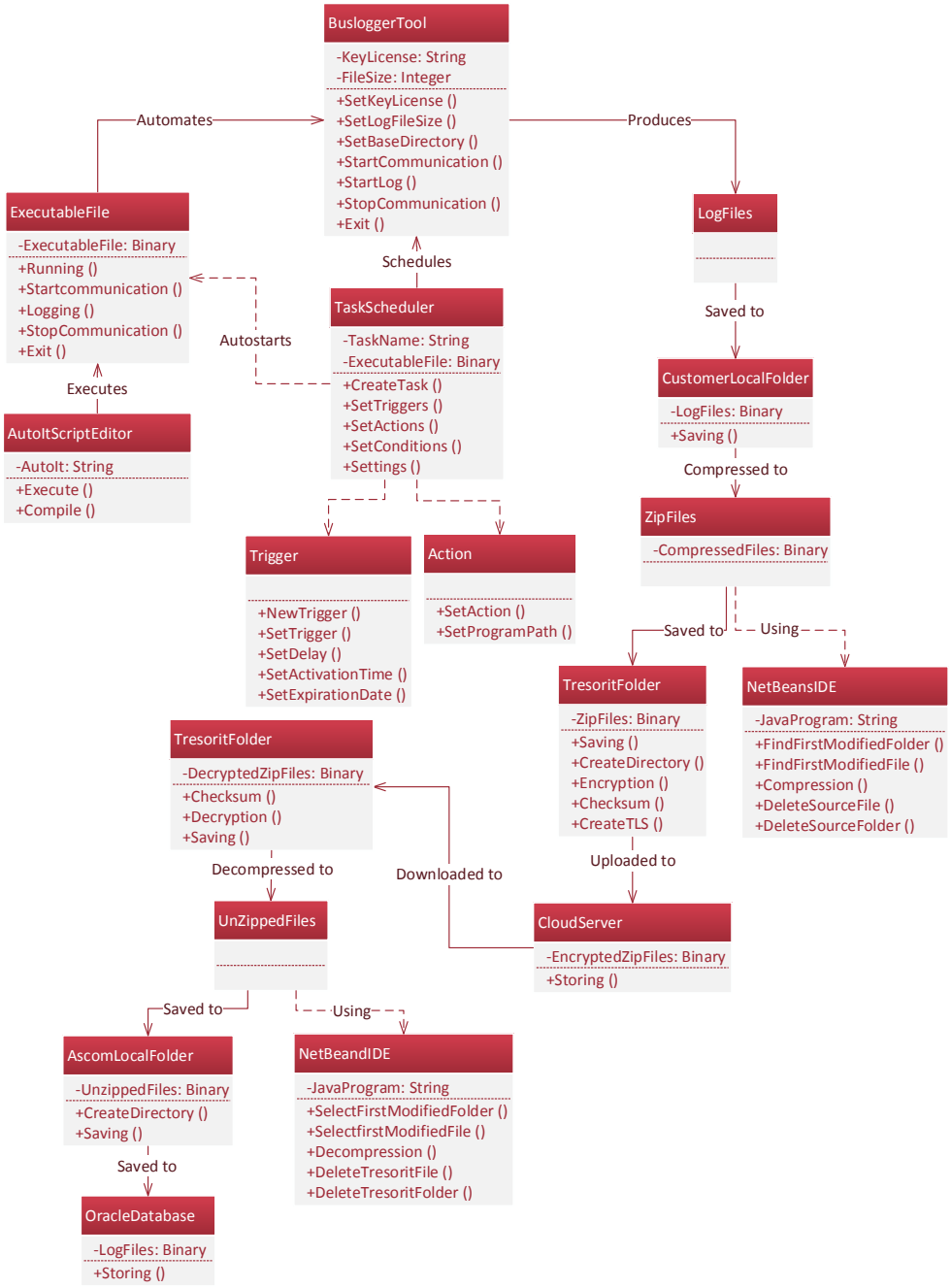


Figure 6.5: Class diagram showing implementation of proof of concept

File transfer service: In presenting Ascom with captured log files, we used a cloud-based storage service and not an FTP application since FTP imposes security concerns to Ascom. Among the tested cloud storage services, *Tresorit* proved to be the best solution for transferring Unite communications from customer's site to Ascom R&D. This is caused by its ability to use end-to-end encryption while Dropbox and Google Drive use server-side encryption. Also, Globus could not be used because we failed to meet its requirement for allocation of shared space.

Database solution: Among the tested database solutions i.e. relational and non relational, *Oracle database* proved to be the best storage solution because it supports the use of XML data types and searching within XML tags. Additionally, Microsoft SQL Server 2014 could not be used because it does not support messages (logs) with a different number of XML tags. However, Neo4j graph database could be used to show analysis for the collected log data.

Hardware devices: The following hardware devices were used during implementation:

- Embedded PC - Intel NUC (Next Unit of Computing) Broadwell Wifi - NUC5i3RYK .
- Hard Drive - Intel 540S Series 120GB m.2 - SATA SSD.
- RAM - Kingston 8GB SO – DIMM 1600MHz DDR3I CL11 1.35V.

Software platforms: The following software platforms were used during implementation:

- Operating System - Windows.
- Windows Task Scheduler.
- Automation language - AutoIt.
- Transfer service - Tresorit.
- Software development platform - NetBeans IDE 8.1.
- Database solution - Oracle Database Enterprise Edition 11g Release 2.

PC Configuration: We performed configurations on a normal PC due to delay in the delivery of the embedded PC. But, this did not affect our implementation since the same configurations would be performed on an embedded PC once it arrived. Windows was installed as an Operating System to run on the PC. The PC was then configured with a dedicated IP address, subnet mask, and the default gateway to facilitate connection to the network at the customer's site. The network information used was obtained from the customer. Afterward, the Buslogger was configured with the IP address of customer's Unite CM, license key, and size of log files to be captured. Later on, the selected cloud service i.e. *client-side Tresorit application* was installed to run on the PC.

6.4.1 PoC Description

When implementing the proof of concept, we created scripts for auto starting and stopping Buslogger, and java programs for compressing, decompressing, and storing

log files. In creating java programs, we used NetBeans IDE to run and compile them into executable files. Once the programs were created, we used Windows Task Scheduler to launch them for their specified time intervals. The implementation of proof of concept described in Figure 6.5 is as explained below:

Auto-start and control Buslogger with AutoIt: To eliminate the need for customer's involvement at the client's side, we automated the process of running the Buslogger, which involved starting the Buslogger, capturing Unite traffic, and saving the captured data. We used a freeware scripting language called AutoIt, which is designed to automate the Windows GUI. As described on AutoIt website: "It uses a combination of simulated keystrokes, mouse movement, and window/control manipulation in order to automate tasks in a way not possible or reliable with other languages (e.g. VBScript and SendKeys)" [17]. AutoIt compiles scripts into an executable file (.exe). "Windows Task Schedule" runs the executable file on Windows startup, which starts the Buslogger application. When Buslogger is running, AutoIt executable file calls the communication function of Buslogger which enable capturing of the Unite traffic. Afterward, it calls "Start log" function which saves the captured data to the local hard disk.

After running the Buslogger for a specified period, "Windows Task Schedule" starts another AutoIt executable file to stop Buslogger's communication function and then exits the application.

We disabled Windows "Sleep" and "Hibernate" functions to prevent interrupting the Buslogger. We also configured Auto-login feature for Windows, since Windows locks when Buslogger is running and login is required to run AutoIt script.

Transmission of log files: Once the log files were captured from the Unite CM, they were first saved into customer's local folder, then later compressed and uploaded to the cloud. After the upload, they were downloaded, decompressed then stored in the database at Ascom R&D. The operations involved at the customer's site as described in Figure 6.5 are explained below:

- ***Compression***: In order to increase transmission speed and save storage space on the cloud, the captured log files were compressed before being uploaded to the cloud. In compressing log files, we used a java program which compressed log files from the source directory i.e. customer's local folder to the destination directory that was created at Tresorit shared folder also known as *Tresor*.

  This java program compressed one log file after the other starting from the first modified one. It did this by selecting the first modified folder if there was more than one folder produced by Buslogger then selecting the first modified file if there was more than one file in a selected folder.

  In case there was only one folder created by the Buslogger, the program compressed all files with an exception to the latest modified file. When only one

file was left in that folder, the program checked if the Buslogger was still running in order to avoid *deadlock* i.e. more than one process using the same resource. If the Buslogger was still running, the program did not do anything until the Buslogger finished running or there were more files created. Moreover, to avoid congesting customer's hard drive, the compression java program deleted all the files with their folders from the source directory after zipping them.

- ***Uploading files to cloud***: After compressing log files, they were uploaded to the cloud using client-side Tresorit application. The client-side application chose a symmetric key to encrypts zipped files before uploading them to the cloud. After encryption, Tresorit ensured the integrity and authenticity of the encrypted zipped files by applying Message Authentication Code (MAC) to each content of a file using a random key that was only known by the client and other users who shared the files.

  Furthermore, before uploading the encrypted zipped files to the cloud, Tresorit establishes a *TLS channel* i.e. a secure channel between the client machine and the cloud so as to protect the data against tampering and eavesdropping while being uploaded [22]. Once The TLS channel was created, the encrypted files were uploaded to the remote directory at the cloud that was the same as the one created at the client-side.

When the encrypted zipped files were made available for download and storage at Ascom's side, the following operations were performed:

- ***Downloading files from cloud***: Before decrypting log files, Tresorit checks the integrity of the transferred files so as to detect any modifications that might have been done in transit. When the integrity checking was successful, the encrypted zipped files were decrypted using the key that was shared by the sender.

- ***Decompression***: Once the files were decrypted, we used a java program which decompressed the decrypted log files from the source directory i.e. Tresorit shared folder, to the destination directory which was Ascom local folder. The decompression java program started by selecting the first modified folder if there was more than one folder created by the Buslogger. Then, it decompressed all files from all the folders starting from the first modified to the least modified one.

  To avoid congesting Tresorit shared space, the decompression program deleted all the zipped files from the source directory after decompressing them. Also, all the created directories at Tresorit shared folder were deleted with an exception to the latest modified which might still be used by the Buslogger at the customer's site.

Storage of files in the database: After log files were decrypted then decompressed into Ascom's Local folder, they were stored into Oracle database for easy access. For storage, we used a java program which included SQL queries for communicating with the database.

The storage java program did this by first establishing a connection to the Oracle database. Once the connection was established, the program checked if the needed table existed in the database. If the table did not exist in the database, the program created it using SQL queries. The created table contained the following columns: the date when the file was inserted into the database, name of the log file, its directory, and the Binary Large OBject (BLOB) column for storing files. After the table was created, the program moved all the files with their needed fields from Ascom's local folder into the Oracle database. To avoid repetition of the same file in the database, the program checked if the name of the file and its directory existed in the database before moving it.

6.4.2 Testing PoC

The Proof of Concept was extremely successful with all the system's requirements met. Also, the created programs were able to run at both customer's and Ascom's side without affecting each other's functionalities. These programs could run and stop at their predefined time-interval and provided results in a reliable and timely manner. Additionally, the implemented solutions can also be used to log Unite communications despite the configuration and usage of Unite system.

7
Discussion

In this chapter, we will discuss our findings towards addressing the challenge facing Ascom in understanding their Unite System which is being used differently by different customers. We managed to address this challenge by first understanding how the Unite System works while using hospitals as a case study. This involved understanding how Unite communications were being logged, from what systems were this information obtained, and what tools were used in capturing it. As a result, we managed to obtain information that was captured from the customer's site and presenting it to Ascom R&D for storage in the database.

In doing so, we evaluated then tested different approaches for both capturing, transferring, and storing Unite communications. When capturing log data, we used the Buslogger which proved to be the best tool in capturing Unite communications from the Unite CM located at customer's site. It did not only manage to capture Unite communications from Unite CM but also ensuring security and privacy of the captured files through encrypting them before storing locally at the customer's site. Additionally, the Buslogger enabled us to view the contents of the log messages in XML format which helped us in determining the type and structure of messages being logged. Also, it gave us access to save the log files in different format and be able to configure their size.

Once log data was captured, we transferred the files using both FTP applications and cloud storage services. The FTP applications i.e. Filezilla and WinSCP, have successfully presented Ascom R&D with the collected log files but they imposed security concerns to Ascom's network. For that reason, we selected cloud storage services which managed to present Ascom with the captured log files with an exception to Globus in which we failed to meet its requirements for allocation of shared storage space.

Among the tested cloud storage services, we decided to use Tresorit since it supports the use of end-to-end encryption while the rest use server-side encryption. According to the related work [21], end-to-end encryption proves to provide security for the transferred data i.e. captured log files, both at rest (at the cloud) and in transit and most important it complies with HIPAA regulations.

When collected files were presented to Ascom R&D, we stored them in the database for improving data access to users, its security, and integrity. To accomplish this, we tested both relational and non relational database solutions in which Oracle

database proved to be the best solution for storing the collected log files. We also could not use Microsoft SQL Server 2014 to store the files because it did not support the format of the messages collected from Unite CM i.e. messages with different numbers of XML tags. Additionally, PostgreSQL failed to be used due to its inability to support searching within XML tags. However, we managed to test a non relational graph database called Neo4j which was found to be useful in performing further analysis for the collected log data although it requires pre-processing.

There are limitations associated with the implementation of this thesis work such as the transfer service used i.e. Tresorit does not keep a backup of the login credentials on the server. So, users have to remember their credentials or else they will not be able to access their stored files. Also, when capturing Unite communication at the customer's side, the Buslogger runs when after logging into the PC, so the user has to login for the Buslogger to operate successfully. Furthermore, our implementation was limited to only one client and one server with a maximum of 1 Terabyte of collected data.

8

Conclusion and Future Work

The understanding of Ascom's Unite System was done successfully and we were able to present Ascom with the captured information that could be used in further system development such as performing troubleshooting and analysis. The used approaches i.e. Tresorit and Oracle database, have their benefits and some limitations during implementation. However, most of these limitations can be addressed by the users at Ascom.

Although Oracle database provided enough storage for the captured log files, other NoSQL database solutions such as MongoDB and InfiniteGraph can also be used to store and analyze captured log data. These NoSQL database solutions support the storage of large files and virtually unlimited scalability. Also, another tool known as Wireshark's plugin, which is currently in the development phase at Ascom can also be used in capturing Unite communications from customer's site.

Even though we automated the process of running the Buslogger, Ascom can have the auto-start functionality built within it. Furthermore, Ascom can use captured logs to evaluate its customers' workload e.g. nurses stress levels as it was done by the previous thesis student.

9

Bibliographic Notes

Reference [1]	It shows how Field Programmable Gate Arrays (FPGA) technology was used to attack cryptographic applications using a password dictionary.
Reference [2]	It examines a number of SQL and NoSQL data stores that are designed to scale Online Transaction Processing (OLTP) application loads i.e data stores that support a large number of simple read/write operations per second, over many servers.
Reference [3]	It describes different efficient and secure ways of transferring, synchronizing, and sharing big data using an online cloud service "Globus".
Reference [4]	It introduces an embedded personal computing (PC) technology and how it allows desktop computer applications to adapt into a real-time control world.
Reference [5]	It emphasize on cloud storage providers, its architecture, security challenges facing it, and encryption methodologies offered by cloud storage services.
Reference [6]	It describes a software which encrypt and decrypt files and text in BlackBerry using cipher block encryption with Cipher Feedback (CFB) 8-bit mode.
Reference [7]	It introduces the client-server system, its architecture, and components both hardware and software components.
Reference [8]	It explains the challenges caused by the use of end-to-end encryption in email service.
Reference [9]	It describes the characterization study of big data transfer protocols on long-haul network by analyzing the performance and fairness of three open-source protocols: GridFTP, FDT, and UDT.
Reference [10]	It introduces big data, causes for its growth, and the requirements for its storage.
Reference [11]	It introduces FTP, its history, connections types, transfer modes, and types of encryption it supports.

Reference [12]	It describes four modes of operation for Data Encryption Standard (DES).
Reference [13]	It documents about PostGreSQL including its background, features, and advantages of using it.
Reference [14]	It is a technical manual for Globus Toolkit.
Reference [15]	It gives an overview of cloud storage, its background, how it works, its benefits, and disadvantages of using it.
Reference [16]	It introduces client-server computing, its purpose, characteristics of a client and a server, advantages, and disadvantages of using it.
Reference [17]	It documents about AutoIt including its features.
Reference [18]	It documents about Microsoft Azure including its security, privacy, transparency, and compliance.
Reference [19]	It introduces big data, its evolution (causes for its growth), characteristics, and sources of big data.
Reference [20]	It is a technical manual for Neo4j graph database.
Reference [21]	It documents about the security implementation in Tresorit.
Reference [22]	It explains why cloud is not trusted by its users then describe the security of an online cloud service "Tresorit" and why it is considered secure to use.
Reference [23]	It is a technical manual for WinSCP.

Bibliography

Books and Articles

[1] A. Abbas et al. "An efficient implementation of PBKDF2 with RIPEMD-160 on multiple FPGAs". In: *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. Dec. 2014, pp. 454–461. DOI: 10.1109/PADSW.2014.7097841.

[2] Rick Cattell. "Scalable SQL and NoSQL data stores". In: *ACM SIGMOD Record* 39.4 (2011), pp. 12–27.

[3] Kyle Chard, Steven Tuecke, and Ian Foster. "Efficient and secure transfer, synchronization, and sharing of big data". In: *Cloud Computing, IEEE* 1.3 (2014), pp. 46–55.

[4] Xin Feng, Steven A Velinsky, and Daehie Hong. "Integrating embedded PC and Internet technologies for real-time control and imaging". In: *IEEE/ASME transactions on mechatronics* 7.1 (2002), pp. 52–60.

[5] Okeke Stephen. "The Study of the Application of Data Encryption Techniques in Cloud Storage to Ensure Stored Data Integrity and Availability". In: *International Journal of Scientific and Research Publications* (2014), p. 259.

[6] M. Wangsadiredja and R. Munir. "Text and file encryption application for blackberry using cipher feedback 8-bit mode". In: *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*. July 2011, pp. 1–6. DOI: 10.1109/ICEEI.2011.6021761.

[7] Subhash Chandra Yadav and Sanjay Kumar Singh. *Architectures of Client/Server Systems*. New Delhi: New Age International (P) Ltd., Publishers, 2009, pp. 41–62.

[8] Jiangshan Yu, Vincent Cheval, and Mark Ryan. "Challenges with End-to-End Email Encryption". In: *Springer Reference* (2014).

[9] Se-young Yu, Nevil Brownlee, and Aniket Mahanti. "Characterizing performance and fairness of big data transfer protocols on long-haul networks". In: *Local Computer Networks (LCN), 2015 IEEE 40th Conference on*. IEEE. 2015, pp. 213–216.

Electronic Resources

[10] A Adshead. *Big Data storage: Defining Big Data and the type of storage it needs*. 2013. URL: http://www.computerweekly.com/podcast/Big-data-storage-Defining-big-data-and-the-type-of-storage-it-needs.

[11] C Chung. *An Introduction to FTP*. 2014. URL: http://www.2brightsparks.com/resources/articles/an-introduction-to-ftp.pdf.

[12] *Federal Information Processing Standards Publication 81. Announcing the Standard for DES MODES OF OPERATION*. 1980. URL: http://csrc.nist.gov/publications/fips/fips81/fips81.htm.

[13] The PostgreSQL Global Development Group. *PostGreSQL*. 1996-2016. URL: https://www.postgresql.org/about/.

[14] *GT 4.0 Component Fact Sheet: GridFTP*. URL: http://toolkit.globus.org/toolkit/docs/4.0/data/gridftp/gridftpfacts.html#s-gridftp-facts-testedplatforms.

[15] Kieu Le and Mark Singh Mark Singh. *Cloud Storage*. 2011. URL: http://public.csusm.edu/fangfang/Teaching/HTMmaterial/StudentProjectFall2011/Team3.pdf.

[16] ND Liyanage. *Client/Server Architecture*. 2013. URL: http://clientserverarch.blogspot.co.uk/2013/03/introduction-to-clientserver-computing.html.

[17] AutoIt Consulting Ltd. *AutoIt*. 2015. URL: https://www.autoitscript.com/site/autoit/.

[18] *Microsoft Azure Trust Center*. 2016. URL: https://azure.microsoft.com/en-us/support/trust-center/.

[19] D Sindol. *Big Data Basics–Part 1–Introduction to Big Data*. 2013. URL: https://www.mssqltips.com/sqlservertip/3132/big-data-basics--part-1--introduction-to-big-data/.

[20] *The Neo4j Manual v2.3.2 - Introduction*. 2016. URL: https://www.neo4j.com/docs/snapshot/introduction.html.

[21] *Tresorit Cloud Storage + End-to-end Encryption*. 2016. URL: https://tresorit.com/security/end-to-end-encryption.

[22] *Tresorit Security Whitepaper*. URL: https://tresorit.com/files/tresoritwhitepaper.pdf.

[23] *WinSCP Free SFTP, SCP and FTP client for Windows*. 2000-2016. URL: https://www.winscp.net/eng/docs/start.

A

Appendix 1

A.1 AutoIt: Script

A.1.1 Script for Starting Buslogger

```
#include <FileConstants.au3>
#include <MsgBoxConstants.au3>
#include <WinAPIFiles.au3>
#region --- Internal functions ---
Func _Au3RecordSetup()
Opt('WinWaitDelay',100)
Opt('WinDetectHiddenText',1)
Opt('MouseCoordMode',0)

EndFunc
; Function to check if Buslogger is the active window
Func _WinWaitActivate($title,$text,$timeout=0)
        WinWait($title,$text,$timeout)
        If Not WinActive($title,$text) Then WinActivate($title,$text)
        WinWaitActive($title,$text,$timeout)
EndFunc

_AU3RecordSetup()
#endregion --- Internal functions End ---
; The address of the Properties file that contain the address of Buslogger
Local Const $sFilePath = @ScriptDir & '\Launch.Properties'
; Open the Properties file to read
Local $hFileOpen = FileOpen($sFilePath, $FO_READ)
 If $hFileOpen = -1 Then  ; If the Properties file is not exist
        MsgBox($MB_SYSTEMMODAL, "", "An error occurred when reading the
 ↪  Launch.Properties file.")
        Exit
    EndIf
; Read the first line of the file
Local $sFileRead = FileReadLine($hFileOpen, 1)
FileClose($hFileOpen)

Run($sFileRead) ; run Buslogger
; Activate the Buslogger window
_WinWaitActivate("BusLogger","")
; Send keyboard keystorke to Buslogger
Send("{ALTDOWN}{ALTUP}fs{ALTDOWN}{ALTUP}os")
```

A.1.2 Script for Stopping Buslogger

```
#region --- Internal functions ---
Func _Au3RecordSetup()
Opt('WinWaitDelay',100)
Opt('WinDetectHiddenText',1)
Opt('MouseCoordMode',0)

EndFunc

Func _WinWaitActivate($title,$text,$timeout=0)
        WinWait($title,$text,$timeout)
        If Not WinActive($title,$text) Then
        WinActivate($title,$text)
        WinWaitActive($title,$text,$timeout)
EndFunc

_AU3RecordSetup()
#endregion --- Internal functions ---
_WinWaitActivate("BusLogger","")
Send("{ALTDOWN}{ALTUP}fs{ALTDOWN}{ALTUP}fx")
```

A.2 Compression Java Program

```
/*
 * This java program performs the following functions:
 * 1. Compresses log files from the local folder to Tresorit folder
 * 2. Deletes the source files and folders after compression at the
 *    local directory (given that the Buslogger is not running).
 * 3. Checking if the Buslogger is running, if not then compress the
 *    first file in the source directory then delete the folder. If
 *    the Buslogger is still running, then do nothing.
 */

package compression;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.FileReader;
import java.util.Properties;

public class Compression {

  public static void main(String[] args){

    while(true) {
      try{
        //Getting the current working directory
```

```
30          String workingDirectory = System.getProperty("user.dir");
31
32          //Loading java properties file
33          Properties propertiesFiles = new Properties();
34          FileReader reader = new FileReader(workingDirectory + "\\" +
35          "Customer.properties");
36          propertiesFiles.load(reader);
37
38          byte[] buffer = new byte[1024];
39
40          //Getting the destination directory
41          String destinationDirectory = propertiesFiles.getProperty
42          ("destinationpath");
43
44          //Getting the source directory
45          String dirPath = propertiesFiles.getProperty("sourcepath");
46          File dir = new File(dirPath);
47
48          //Get a list of subdirectories in that directory
49          File listDirectory[] = dir.listFiles();
50
51
52          /* Checking the first modified subdirectory
53           * If the list of subdirectories is more than one */
54          if (listDirectory.length > 1){
55            File FirstModifiedFolder = listDirectory[0];
56            for (int j = 0; j < listDirectory.length; j++){
57              if (FirstModifiedFolder.lastModified() >
58              listDirectory[j].lastModified()){
59                FirstModifiedFolder = listDirectory[j];
60              }
61            }
62
63            /* Creating a new subdirectory path and assign it
64             * to SubdirPath */
65            String SubdirPath = dirPath + FirstModifiedFolder.getName()
66            + "\\";
67            File SubDirectory = new File(SubdirPath);
68            String[] NumberofFiles = FirstModifiedFolder.list();
69
70            /* For loop for iterating the number of files to
71             * be compressed in the first modified folder */
72            for (int k = 0; k < NumberofFiles.length; k++){
73
74              //Getting array of files in the subdirectory
75              String[] empty = FirstModifiedFolder.list();
76
77              /* Checking if the number of files in the first
78               * modified folder is >= 1 */
79              if (empty.length > 0){
80                //Get all the first modified files from the subdirectory
81                File[] files = SubDirectory.listFiles();
82                File FirstModifiedFile = files[0];
83
84                /* Getting the name of the first modified file
85                 * without the path */
```

```
86              String filename = FirstModifiedFile.getName();
87
88              //Removing the extension from filename
89              String name = filename.substring(0,
90              filename.lastIndexOf('.'));
91
92              //The Zip extension
93              String FILE_EXTENSION = ".Zip";
94
95              /* Creating a directory in Tresorit shared folder with
96               * the same name as the source folder in local
97               * directory */
98              String New_destinationDirectory = destinationDirectory +
99              FirstModifiedFolder.getName() + "\\";
100             File newDirectory = new File(New_destinationDirectory);
101             newDirectory.mkdirs();
102
103             //Create a filepath for the zip file
104             String filepath = newDirectory + "\\" + name +
105             FILE_EXTENSION;
106
107             /* Getting the first modified file from the
108              * firstModiiedFolder */
109             for (int i = 0; i < files.length; i++){
110               if (FirstModifiedFile.lastModified() >
111               files[i].lastModified()){
112                 FirstModifiedFile = files[i];
113               }
114             }
115
116             FileOutputStream file_output = new FileOutputStream
117             (filepath);
118             ZipOutputStream zip_output = new ZipOutputStream
119             (file_output);
120
121             //Adding a file to a zip entry
122             ZipEntry zip_entry = new ZipEntry(filename);
123             zip_output.putNextEntry(zip_entry);
124
125             //Read the file from the given path
126             FileInputStream input = new FileInputStream
127            (FirstModifiedFile);
128
129             int len;
130             while ((len = input.read(buffer)) > 0){
131               zip_output.write(buffer, 0, len);
132             }
133
134             input.close();
135             zip_output.closeEntry();
136             zip_output.close();
137
138             //Delete the compressed file from source
139             FirstModifiedFile.delete();
140           }
141         }
```

```
142
143             //Checking if the folder is empty then delete it
144             String[] empty = FirstModifiedFolder.list();
145             if (empty.length == 0){
146               FirstModifiedFolder.delete();
147             }
148           }
149
150           else if (listDirectory.length == 1){
151             File FirstModifiedFolder = listDirectory[0];
152
153             //Create a new subdirectory path
154             String SubdirPath = dirPath + FirstModifiedFolder.getName() +
155             "\\";
156             File SubDirectory = new File(SubdirPath);
157             String[] NumberofFiles1 = FirstModifiedFolder.list();
158
159             /* Creating a folder at Tresorit that has the same name as
160              * the source folder at local folder and move its
161              * compressed files in it */
162             String New_destinationDirectory = destinationDirectory +
163             FirstModifiedFolder.getName() + "\\";
164             File newDirectory = new File(New_destinationDirectory);
165             newDirectory.mkdirs();
166
167             if (NumberofFiles1.length > 1){
168
169               for (int h = 1; h < NumberofFiles1.length; h++){
170                 //Get all the file from the subdirectory
171                 File[] files = SubDirectory.listFiles();
172
173                 File FirstModifiedFile = files[0];
174
175                 /* Getting the name of the first modified file
176                  * without the path */
177                 String filename = FirstModifiedFile.getName();
178
179                 //Removing the extension from filename
180                 String name1 = filename.substring(0,
181                 filename.lastIndexOf('.'));
182
183                 //The Zip extension
184                 String FILE_EXTENSION1 = ".Zip";
185
186                 //Create a filepath for the zip file
187                 String filepath = newDirectory + "\\" + name1 +
188                 FILE_EXTENSION1;
189
190                 //For loop for getting the first modified file
191                 for (int g = 1; g < files.length; g++){
192                   if (FirstModifiedFile.lastModified() >
193                   files[g].lastModified()){
194                     FirstModifiedFile = files[g];
195                   }
196                 }
197
```

```
198                 FileOutputStream file_output = new FileOutputStream
199                 (filepath);
200                 ZipOutputStream zip_output = new ZipOutputStream
201                 (file_output);
202
203                 //Adding a file to zip entry
204                 ZipEntry zip_entry = new ZipEntry(filename);
205                 zip_output.putNextEntry(zip_entry);
206
207                 //Read the file from the given path
208                 FileInputStream input = new FileInputStream
209                 (FirstModifiedFile);
210
211                 int len;
212                 while ((len = input.read(buffer)) > 0){
213                   zip_output.write(buffer, 0, len);
214                 }
215
216                 input.close();
217                 zip_output.closeEntry();
218                 zip_output.close();
219
220                 //Delete the compressed file from source
221                 FirstModifiedFile.delete();
222               }
223             }
224
225             else if (NumberofFiles1.length == 1){
226
227               String processIdInfo ="";
228               String line;
229
230               //Getting directory to task list
231               String Task_listing = propertiesFiles.getProperty
232               ("TASK_LIST");
233
234               //Get process name
235               String App_name = propertiesFiles.getProperty
236               ("Application_Name");
237
238               Process p = Runtime.getRuntime().exec(Task_listing);
239
240               BufferedReader input =  new BufferedReader(new
241               InputStreamReader(p.getInputStream()));
242
243               //Getting the id of the running process
244               while ((line = input.readLine()) != null) {
245                 processIdInfo+=line;
246               }
247
248               input.close();
249
250               //Checking if buslogger is running on windows
251               if(!processIdInfo.contains(App_name)){
252
253                 //Get all the file from the subdirectory
```

```
254                 File[] files = SubDirectory.listFiles();
255
256                 File New_FirstModifiedFile = files[0];
257
258                 /* Getting the name of the first modified file
259                  * without the path */
260                 String New_Filename =
261                 New_FirstModifiedFile.getName();
262
263                 //Removing the extension from filename
264                 String New_Name = New_Filename.substring(0,
265                 New_Filename.lastIndexOf('.'));
266
267                 //The Zip extension
268                 String FILE_EXTENSION2 = ".Zip";
269
270                 //Create a filepath for the zip file
271                 String filepath = newDirectory + "\\" + New_Name +
272                 FILE_EXTENSION2;
273
274                 FileOutputStream newfile_output = new
275                 FileOutputStream(filepath);
276                 ZipOutputStream Newzip_output = new
277                 ZipOutputStream(newfile_output);
278
279                 //Adding a file to zip entry
280                 ZipEntry zip_entry = new ZipEntry(New_Filename);
281
282                 Newzip_output.putNextEntry(zip_entry);
283
284                 //Read the file from the given path
285                 FileInputStream input1 = new FileInputStream
286                 (New_FirstModifiedFile);
287
288                 int len;
289                 while ((len = input1.read(buffer)) > 0){
290                   Newzip_output.write(buffer, 0, len);
291                 }
292
293                 input1.close();
294                 Newzip_output.closeEntry();
295                 Newzip_output.close();
296
297                 //Delete the compressed file from source
298                 New_FirstModifiedFile.delete();
299                 FirstModifiedFolder.delete();
300
301               }
302             }
303           }
304         }
305         catch(Exception e){
306         }
307       }
308     }
309 }
```

A.3 Decompression Java Program

```
/*
 * This java program performs the following functions:
 * 1. Check the first modified folder if there is more than one
 *    folder, if yes then decompresses all the log files from Tresorit
 *    first modified folder to Ascom's local folder.
 * 2. Then deletes the decompressed files from Tresorit folder and the
 *    source folder .
 * 3. If there is only one folder in the Tresorit directory,
 *    decompress all the zip files but do not delete the source folder.
 */
package decompression;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.io.FileReader;
import java.util.Properties;

public class Decompression {

  public static void main(String[] args) {
    while (true) {
      try {
        //Getting the current working directory
        String workingDirectory = System.getProperty("user.dir");

        //Loading java properties file
        Properties prop = new Properties();
        FileReader reader = new FileReader
        (workingDirectory + "\\" + "Ascom.properties");
        prop.load(reader);

        byte[] buffer = new byte[1024];

        //Setting the source directory
        String sourceDirectory = prop.getProperty("Sourcepath");
        File sourcedir = new File(sourceDirectory);

        //Get a list of subdirectories in that source path
        File listDir[] = sourcedir.listFiles();

        //Getting the destination directory
        String destDirectory = prop.getProperty("Destinationpath");

        /* Checking if there is unwanted file called "desktop.ini"
         * then delete it if present */
        String Unwantedpath = prop.getProperty("UnwantedPath");
```

```
51          if (new File(Unwantedpath).exists()) {
52            new File(Unwantedpath).delete();
53          }
54
55          /* Checking the first modified subdirectory if the list of
56           * subdirectories is more than one */
57          if (listDir.length > 1) {
58            File firstModifiedFolder = listDir[0];
59            for (int j = 0; j < listDir.length; j++) {
60              if ((firstModifiedFolder.lastModified() >
61              listDir[j].lastModified())) {
62                firstModifiedFolder = listDir[j];
63              }
64            }
65
66            /* Creating a new subdirectory path for the selected first
67             * modified folder at the source and assign it to Subdir */
68            String SubdirPath = sourceDirectory +
69            firstModifiedFolder.getName() + "\\";
70            File Subdir = new File(SubdirPath);
71
72            //Getting an array of files in that subdirectory
73            String[] NumberofFiles = firstModifiedFolder.list();
74
75            //Removing unwanted files
76            String Unwantedpath1 = prop.getProperty("UnwantedPath");
77            if (new File(Unwantedpath1).exists()) {
78              new File(Unwantedpath1).delete();
79            }
80
81            /* For loop for iterating the number of files to be
82             * compressed in the first modified folder */
83            for (int k = 0; k < NumberofFiles.length; k++) {
84              String[] empty = firstModifiedFolder.list();
85
86              /* Checking if the number of files in the first modified
87               * folder is greater than 0 */
88              if (empty.length > 0) {
89                //Get all the first modified file from the subdirectory
90                File[] files = Subdir.listFiles();
91                File firstModifiedFile = files[0];
92
93                /* Getting the name of the first modified file without
94                 * the path */
95                String filename = firstModifiedFile.getName();
96
97                /* Creating a directory that has the same name as the
98                 * source folder at local folder and move its compressed
99                 * files in it */
100               String newdestDirectory1 = destDirectory +
101               firstModifiedFolder.getName() + "\\";
102               File newDirectory1 = new File(newdestDirectory1);
103               newDirectory1.mkdirs();
104
105               /* Getting the first modified file from the
106                * firstModifiedFolder */
```

```
107                 for (int i = 0; i < files.length; i++) {
108                   if (firstModifiedFile.lastModified() > files
109                   [i].lastModified()) {
110                     firstModifiedFile = files[i];
111                   }
112                 }
113
114                 //Read the zipfile from the given path
115                 ZipInputStream zis = new ZipInputStream
116                 (new FileInputStream(firstModifiedFile));
117
118                 //Adding a file entry to zipEntry
119                 ZipEntry ze = zis.getNextEntry();
120
121                 //When the zipEntry is not empty (has a zip file)
122                 while (ze != null) {
123                   /* Get the name of the selected file before
124                    * compression e.g. 141202 094318.log */
125                   String fileName = ze.getName();
126
127                   //Create a destination path for the file
128                   String filePath = newDirectory1 + "//" + fileName;
129                   File newFile = new File(filePath);
130
131                   /* create all non exists folders to avoid
132                    * FileNotFoundException for compressed folder */
133                   new File(newFile.getParent()).mkdirs();
134
135                   FileOutputStream fos = new FileOutputStream(newFile);
136
137                   int len;
138                   while ((len = zis.read(buffer)) > 0) {
139                     fos.write(buffer, 0, len);
140                   }
141
142                   fos.close();
143                   ze = zis.getNextEntry();
144                 }
145
146                 zis.closeEntry();
147                 zis.close();
148
149                 //Delete the compressed file from source
150                 firstModifiedFile.delete();
151               }
152             }
153             //Checking if the folder is empty then delete it
154             String[] empty = firstModifiedFolder.list();
155             if (empty.length == 0) {
156               firstModifiedFolder.delete();
157             }
158           }
159           else if (listDir.length == 1) {
160             //Getting the first modified folder
161             File firstModifiedFolder = listDir[0];
162
```

```
163             //Create a new subdirectory path
164             String SubdirPath1 = sourceDirectory +
165             firstModifiedFolder.getName() + "\\";
166             File Subdir1 = new File(SubdirPath1);
167             String[] NumberofFiles1 = firstModifiedFolder.list();
168
169             /* Creating a folder at Tresorit that has the same name as
170              * the source folder at local folder and move its compressed
171              * files in it */
172             String newdestDirectory2 = destDirectory +
173             firstModifiedFolder.getName() + "\\";
174             File newDirectory2 = new File(newdestDirectory2);
175             newDirectory2.mkdirs();
176
177             //Checking if the number of files in the first folder is >=1
178             if (NumberofFiles1.length >= 1) {
179
180               //For all the number of files in the first folder
181               for (int h = 0; h < NumberofFiles1.length; h++) {
182                 File[] files = Subdir1.listFiles();
183
184                 File firstModifiedFile1 = files[0];
185
186                 /* Getting the name of the first modified file without
187                  * the path */
188                 String filename1 = firstModifiedFile1.getName();
189
190                 //For loop for getting the first modified file
191                 for (int g = 1; g < files.length; g++) {
192                   if (firstModifiedFile1.lastModified() >
193                   files[g].lastModified()) {
194                     firstModifiedFile1 = files[g];
195                   }
196                 }
197
198                 //Read the zipfile from the given path
199                 ZipInputStream zis = new ZipInputStream
200                 (new FileInputStream(firstModifiedFile1));
201
202                 //Adding a file entry to zipEntry
203                 ZipEntry ze = zis.getNextEntry();
204
205                 //When the zipEntry is not empty
206                 while (ze != null) {
207                   /* Get the name of the selected file before
208                    * compression e.g. 141202 094318.log */
209                   String fileName1 = ze.getName();
210
211                   //Create a destination path for the file
212                   String filePath1 = newDirectory2 + "//" + fileName1;
213                   File newFile1 = new File(filePath1);
214
215                   /* create all non exists folders to avoid
216                    * FileNotFoundException for compressed folder */
217                   new File(newFile1.getParent()).mkdirs();
218
```

```
                    FileOutputStream fos = new FileOutputStream(newFile1);

                    int len;
                    while ((len = zis.read(buffer)) > 0) {
                      fos.write(buffer, 0, len);
                    }

                    fos.close();
                    ze = zis.getNextEntry();
                  }

                  zis.closeEntry();
                  zis.close();

                  //Delete the compressed file from source
                  firstModifiedFile1.delete();
                }
              }
            }
          }
          catch (IOException e) {
          }
      }
    }
}
```

A.4 Storage Java Program

```
/*
 * This java program performs the following functions:
 * 1. Establish the connection with the Oracle database
 * 2. Creates a table in the Oracle database using SQL queries.
 * 3. Copy captured log files from Ascom's local folder to store them
 *    in a table created in Oracle database
 */

package processdatabase;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.InputStream;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Calendar;
import java.util.Properties;

```

```
26 public class ProcessDatabase {
27
28   public static Connection getConnection() throws Exception{
29     //Getting the current working directory
30     String workingDirectory = System.getProperty("user.dir");
31
32     //Loading java properties file
33     Properties propertiesFiles = new Properties();
34     FileReader reader = new FileReader(workingDirectory + "\\" +
35     "DatabaseInputs.properties");
36     propertiesFiles.load(reader);
37
38     /* Specifying the database URL with the following properties
39      * user, password, host, port number and SID */
40     String url = propertiesFiles.getProperty("CONNECT_URL");
41     String login_username = propertiesFiles.getProperty("USERNAME");
42     String login_password = propertiesFiles.getProperty("PASSWORD");
43
44     //Loading the driver class then establish the connection
45     Class.forName(propertiesFiles.getProperty("DRIVER_CLASS"));
46
47     Connection con = DriverManager.getConnection(url, login_username,
48     login_password);
49
50     return con;
51   }
52
53   public static void main(String[] args) {
54
55     while(true){
56       Connection con = null;
57
58       try{
59         //Getting the current working directory
60         String workingDirectory = System.getProperty("user.dir");
61
62         //Loading java properties file
63         Properties propertiesFiles = new Properties();
64         FileReader reader = new FileReader(workingDirectory + "\\" +
65         "DatabaseInputs.properties");
66         propertiesFiles.load(reader);
67
68         byte[] buffer = new byte[1024];
69
70         String FilePath = propertiesFiles.getProperty("SOURCE_PATH");
71
72         File dir = new File(FilePath);
73
74         //Get a list of subdirectories in that directory
75         File listDir[] = dir.listFiles();
76
77         for (int c = 0; c < listDir.length; c++){
78
79           /* Creating a new subdirectory path and assign it to
80            * SubdirPath */
81           String SubdirPath = FilePath + listDir[c].getName() + "\\";
```

```
82              File Subdir = new File(SubdirPath);
83              String[] NumberofFiles = listDir[c].list();
84
85              //Getting array of files in the subdirectory
86              String[] empty = listDir[c].list();
87
88              //Getting the subdirectory path as a string
89              String SubDirectory_Path1 = listDir[c].getAbsolutePath();
90
91              //Establish a connection
92              con = getConnection();
93
94              Statement st = con.createStatement();
95
96              //Check if the table exist if not create a table
97              String create1 = propertiesFiles.getProperty
98              ("CREATING_TABLE");
99              String table_naming = propertiesFiles.getProperty
100             ("TABLE_NAME");
101             DatabaseMetaData dbm = con.getMetaData();
102             ResultSet rs = dbm.getTables(null, null, table_naming, null);
103             if (rs.next()) {
104
105             }
106             else {
107               st.execute(create1);
108             }
109
110             if (empty.length > 0){
111
112               //Get all the first modified file from the subdirectory
113               File[] files = Subdir.listFiles();
114               File firstModifiedFile = files[0];
115
116               /* Getting the first modified file from the
117                * firstModifiedFolder */
118               for (int i = 0; i < files.length; i++){
119
120                 firstModifiedFile = files[i];
121
122                 //Converting a file directory
123                 String ModifiedFile_Path =
124                 firstModifiedFile.getAbsolutePath();
125
126                 //Getting the File_Name
127                 Path p = Paths.get(ModifiedFile_Path);
128                 String File_Name1 = p.getFileName().toString();
129
130                 InputStream inputStream = new FileInputStream(new
131                 File(ModifiedFile_Path));
132
133                 //Checking if the path of the log file exist
134                 String selecting = propertiesFiles.getProperty
135                 ("SELECT_STATEMENT");
136                 PreparedStatement psy = con.prepareStatement(selecting);
137                 psy.setString(1, ModifiedFile_Path);
```

```
138
139                 ResultSet rsy = psy.executeQuery();
140
141                 if (!rsy.next()) {
142
143                   //Getting the current time from Calender
144                   Calendar calendar = Calendar.getInstance();
145                   java.sql.Timestamp ourJavaDateObject = new
146                   java.sql.Timestamp(calendar.getTime().getTime());
147
148                   //Inserting values into a table
149                   String inserting_statement =
150                   propertiesFiles.getProperty("INSERT_STATEMENT");
151                   String inserting_values =
152                   propertiesFiles.getProperty("INSERT_VALUES");
153                   PreparedStatement ps = con.prepareStatement
154                   (inserting_statement + inserting_values);
155
156                   ps.setTimestamp(1, ourJavaDateObject);
157                   ps.setString(2, SubDirectory_Path1);
158                   ps.setString(3, File_Name1);
159                   ps.setString(4, ModifiedFile_Path);
160                   ps.setBlob(5, inputStream);
161                   ps.executeUpdate();
162                 }
163               }
164             }
165           }
166         }
167         catch(Exception e){
168         }
169       }
170     }
171 }
```