A Software Architecture for a Real-Time Big Data System : A Case Study of a Spectrum-Sensing Enabled Whitespace Database

By

Litsietsi Montsi

Faculty of Engineering and Built Environment
University of the Witwatersrand Johannesburg

A Masters Dissertation

Supervisor: Prof. Barry Dwolatzky
Co-supervisor: Dr. Luzango Mfupe (CSIR)

September 10, 2018

DECLARATION

I declare that this research is my own work. It is being submitted for the Degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other university.

Parts of this work have been accepted and presented at:

IEEE Global Wireless Summit (GWS) Conference, Cape Town, South Africa, October 15 - 18 2017.

No work has been published or presented anywhere else.

_____

10th September 2018

ACKNOWLEDGMENTS

I would firstly like to thank God for whom everything is possible; also, my family and loved ones, for their unwavering support throughout my life. I would like to give my sincere thank you to my supervisor at the University of the Witwatersrand, Professor Barry Dwolatzky, for being patient and for guiding me throughout the entire course of this Research. Professor Barry's research and technical leadership in Big Data research was very crucial in helping me complete this research. I would also like to give thanks to my supervisor at the Council for Scientific and Industrial Research (CSIR), Doctor Luzango Mfupe, for his support and guidance and technical leadership in the area of television whitespaces.

I would also like to thank my employer, the Council of Scientific and Industrial Research (CSIR) for not only funding my studies but for also providing me with a conducive environment for learning and growing as a professional. I would like to give a special thank you to my manager, Doctor Keith Ferguson, for giving me the time and support I needed to complete this research. I would also like to thank the Chief Scientist at the CSIR who was the project leader/client; Professor Fisseha Mekuria. He gave me access to the systems' data and made this research possible. A warm thank you to the entire CSIR and to the University of the Witwatersrand for cultivating learning and providing a platform for growth.

ABSTRACT

Due to the ever-growing need to process vast amounts of data in real-time, more and more tools which serve different needs in the real-time Big Data processing pipeline have sprung out. However, holistic industry-accepted frameworks that address all real-time Big Data processing requirements across the entire pipeline have not yet been developed. More so, the area of dynamic spectrum access has more and more devices connecting to previously unavailable radio frequency spectrum. This vastly growing number of devices need real-time orchestration on how they access this newly made available spectrum. The development of a real-time Big Data system in the realm of dynamic spectrum access as required by the Council for Scientific and Industrial Research served as a case study for this research.

This research provides a step in reaching an industry-wide accepted software reference architecture for real-time big data systems which will be followed in the development of a real-time Big Data system. This is done by uncovering the most important quality/architectural requirements of real-time Big Data systems which such a reference architecture is to address. It is shown that all major software reference architectures (Java Enterprise Edition, AutoSar, Microsoft.Net, and others) were developed with emphasis placed on addressing a set of specific prioritised requirements. Hence this research uses this principle to propose a method to help in the development of software architectures and software reference architectures of real-time Big Data systems.

In this research, a case study is used to make inference on the general population of real-time Big Data systems about the method proposed in this research. A mathematical ranking method is employed to prioritise software architecture requirements of a case study system and the results are compared with literature to increase the accuracy of the inference. Then architecture design and experiments were carried-out and presented to the Council for Scientific and Industrial Research as the client for acceptance, which would serve as validation. This was further validated by comparing the results of the case study to work done by other researchers. Having uncovered the most important quality attributes for

real-time Big Data systems, the software architecture design process for such systems is simplified and fertile ground has been laid for the development of software reference architectures for real-time Big Data systems.

CONTENTS

GLOSSARY

Acronym	Meaning
ACM	Association for Computing Machinery
ADS-B	Automatic Dependent Surveillance-Board
AHP	Analytic Hierarchy Process
AUTOSAR	AUTomotive Open System Architecture
CHPC	Centre for High Performance Computing
CSIR	Council for Scientific and Industrial Research
DADU	Data Analysis and Decision Unit
DPU	Data Processing Unit
EJB	Enterprise Java Beans
Gbit	Gigabit
HDFS	Hadoop Distributed File System
HTTP	Hypertext transfer protocol
IBM	International Business Machines
ICASA	Independent Communications Authority of South Africa
ICT	Information Communication Technology
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of things
IP-address	Internet Protocol Address
ISO	International Standardization Organization
JAD	Joint Application Development
Java EE	Java Enterprise Edition
JSON	JavaScript Object Notation

KML	Keyhole Mark-up Language
MCDA	Multi-Criteria Decision Aid
MQTT	Message Queuing Telemetry Transport
NIST	National Institute of Standards and Technology
OFCOM	Office of Communications
OR	Operations Research
OSI	Open Systems Interconnection
PAWS	Protocol to Access White-Space
PROMETHEE	Preference Ranking Organisation Method for Enrichment Evaluation
RDLab	Laboratory Research and Development
RSDU	Remote Sensing Data acquisition Unit
SABC	South African Broadcasting Corporation
SBT	Smart Transportation Building
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SSEWDB	Spectrum Sensing Enabled Whitespace Database
TV	Television
TVWS Database	Television Whitespace Database
UHF	Ultra-High Frequency
UML	Unified Modelling Language
URL	Uniform Resource Locator
VHF	Very High Frequency
WISP	Whitespace Internet Service Provider

WSD	Whitespace Device
WSDB	Whitespace Database

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The term "Big Data software architecture" refers in this context to the infrastructure put in place to handle huge volumes of data with varying formats and sources that might be arriving at a system with very high speeds. Before delving into Big-Data, it is worthwhile explaining what software architecture is, as it is the central focal point of this research.

The concept of "software architecture" places emphasis on satisfying quality requirements and serving as a basis of design through the selection of architectural elements, their interaction and the constraints on those requirements (Perry and Wolf, 1992). Solms (2013) went a step further and described software architecture as the infrastructure within which application components are deployed and executed. What this implies in relation to modern software development is that software applications are no longer created from scratch, but instead are developed from application components that are deployed within an already existing infrastructure called the software architecture. Solms (2013) corroborated this notion in much more detail and based his discussion on the definition of software reference architectures provided by Lloyd and Galambos (1999) which stated that *software reference architectures are domain specific architectural templates which aim to address architectural concerns of a particular class of problems*.

From the definitions above, one can develop a conceptual model of a typical software architecture development lifecycle. Figure 1 below illustrates the conceptual model of a software architecture development life cycle.

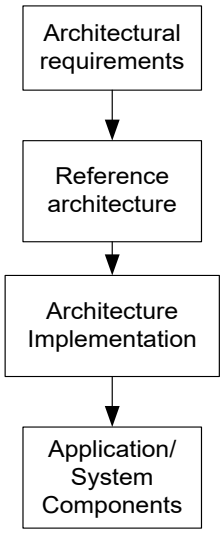


Figure 1: Conceptual model of a software architecture development life cycle

These definitions are central to this research, so it is worth looking at an example of what they illustrate. To clarify these definitions, an online banking system and online services system are used. Table 1 shows the software architecture process of an online banking system and online services system from requirements to implementation.

Table 1: Software architecture development lifecycle examples

Architectural requirements	**Software reference architecture**	**Implementing framework/Architecture Implementation**	**System**
Scalability, security, performance	Java Enterprise Edition	JBoss application server	Implemented banking system
Integrability	Services Oriented Architecture (SOA)	Service Oriented Architecture Implementation Framework (SOAIF)	Implemented online services system

Scalability, security, and performance are the architectural requirements of enterprise systems such as online banking system. The Java Enterprise Edition (Java EE) reference architecture was developed to address architectural requirements with emphasis on scalability, security, and performance (Solms 2013). JBoss application server is an example of an implementation framework of java EE. Enterprise systems whose main architectural

requirement was integrability are implemented following the Service Oriented Architecture (SOA) reference architecture (Solms 2013). The implementing framework for SOA is the Services Oriented Architecture Implementing Framework (SOAIF) (Fiorano 2017).

The following subsections give more details to the concepts just introduced. They also provide more justification to the software architecture development life cycle model derived from the definitions above. This research follows the software architecture development life cycle model derived from the definitions of what software architecture is hence emphasis is placed on explaining it.

1.1. BACKGROUND

1.1.1. Introduction

In this section, quality attributes such as latency, security and many more, which are the main drivers behind any software architecture development as well as the development of a software reference architecture, will be explored. It will also be shown through literature that across different areas of Information Communication Technology (ICT), there are different reference architectures that address certain quality attributes that are shared between different problems (systems) within a class of problems. This is important in that, it justifies the generalisation which is made from a case study, of quality attributes being the same for real-time Big Data systems. Software reference architectures being templates for developing solutions are also studied by examining different areas of ICT and the reference architectures they use. It will be shown that for Big Data, there are no reference architectures.

1.1.2. Software Quality Requirements

1.1.2.1. Introduction

Quality attributes are the general groupings of non-functional requirements of a software system that describe its behavioural characteristics (Wiegers 2003). Non-functional requirements can be categorized into the following quality attributes: performance, integrity, security, interoperability, safety, testability, and usability (Wiegers 2003). Blaine

and Cleland-Huang (2008) however, pointed out that this list can be longer. In building an ontology for software product quality attributes, Kayedet. Al. (2009) described software quality attributes and gave a list like Wiegers (2003) but with several additions.

1.1.2.2. Challenges of quality attributes

Blaine and Cleland-Huang (2008) stated that quality attributes, as opposed to functional requirements, have serious problems in implementation. One of the most important challenges is that, quality attributes exhibit trade-offs that one must take careful consideration of when making a choice between competing quality attributes. The authors made an example of stakeholders wanting a secure system which is, at the same time, easily accessible to users. This is clearly going to result in one of the quality attributes being an opportunity cost for the other.

Another challenge which Blaine and Cleland-Huang (2008) stated was that quality attributes are harder to measure as opposed to functional requirements. Functional requirements tend to be binary in nature. That is, a system either has the desired functionality or it does not. Quality attributes, on the other hand, are achieved at a certain magnitude along a continuous scale of measurement. For example, one cannot simply say a system has been secured. They must express to what magnitude has this been accomplished. This poses a challenge in the specification, tracking and implementation of these quality attributes.

1.1.3. Software Reference Architectures Across ICT

The automotive industry is the first area of focus in ICT whose reference architecture, called AUTomotive Open System Architecture (AUTOSAR), is presented. Automobile manufacturers, suppliers and tool developers formed the AUTOSAR consortium to help manage electronic complexity, costs, quality, and re-usability issues in the automobile industry (D Kum et. al 2008). It is therefore evident that the quality attributes which AUTOSA addresses are re-usability, performance, accuracy and availability under quality. Reduced cost and ease of development also fit in this category.

Another domain where reference architectures are employed is computer networks. At the highest-level granularity, the International Standardization Organization (ISO) defined a reference architecture called the Open Systems Interconnection (OSI) Model (Buschmann et al. 1996). The OSI model describes seven layers of the network being:

- application
- presentation
- session
- transport
- network
- data link
- physical layers

The architectural pattern used by the OSI model is the layered architectural pattern.

Enterprise applications have a few reference architectures depending on which non-functional requirements they were designed to meet. If an enterprise wants their systems to meet requirements of high performance, integrability, with special emphasis on reliability, scalability and security, then the Java Enterprise Edition (Java-EE) is the reference architecture of choice (Solms 2012). Java-EE uses a layered architectural pattern. The layers in the context of java-EE are the access layer, which is the web container, the process execution layer which is the Enterprise Java Beans (EJB) container, and the integration layer which is the persistence context.

If one is concerned with providing an integration infrastructure which also makes modifiability easy, then Service Oriented Architecture (SOA) is the desired reference architecture (Solms, 2012). SOA is based on the microkernel software architectural pattern. Solms (2012) pointed out that, this pattern has to have a service bus that routes services and requests to providers and consumers respectively. Such a component in SOA is the Enterprise Service Bus (ESB). A catalogue of service providers is kept in the service

registry. Service providers and consumers tap into the ESB through adapters and services are executed in the process execution engine.

1.1.4. Software Architectures in Big data

Big Data is, for the most part, an enterprise-based phenomenon. At the time of writing, there is no industry-wide accepted software reference architecture for Big Data. Most companies that were consulted through a preliminary study stated that they only make tailored solutions to the problems of the customer. Industry expert Nigel Freddy asserted that a generic solution is not possible (personal communication, October 28 2014). To the best of the author's knowledge, only two organisations, namely, International Business Machines (IBM) and Oracle have their own software reference architectures for Big Data. However, these are proprietary reference architectures tailored for solutions that are only provided by the two companies. Additionally, there was no mention of which patterns these were based on and which quality attributes were identified as most common in the problem class.

Although there is no industry wide accepted reference architecture, there has been work done around this area of Big Data. Doshi et al (2013) described an approach of blending SQL and NoSQL Database Management technologies to address different classes of Big Data problems encountered by enterprises. The authors coined this approach as "reference architectures for Big Data challenges". However, it does not conform to the definition of what a reference architecture is and is merely an approach. This means it does not provide a holistic architectural template on how to solve Big Data problems but focuses only on data management and suggests which technologies to use.

A reference architecture for Big Data solutions has also been presented (Geerdink, 2013). The suggested architecture outlined a template of how to solve Big Data problems using three layers. These three layers were *business layer, application layer and technology layer*. Geerdink discussed that there is reference architecture and solutions reference architecture. He stated that solutions architectures are templates that addressed how a solution would be constructed from a process point of view and a technology point of view.

Geerdink (2013) presented a Big Data process flow from data collection to analytics and presented the corresponding technology for each stage of the process flow in the technology layer. This differs from the research presented in this dissertation in that; the research will focus more on the generic architectural requirements of real time Big Data systems and how they are addressed by the software architecture and focus on lower levels of granularity in the technology layer.

In an online white paper, Oracle presented a reference architecture for Big Data and analytics. Oracle Information Architecture (2012) used a layered architectural pattern in presenting this work. The Oracle reference architecture seemed to address generic Big Data architectural requirements but was very brief and only showed the logical view of the reference architecture. More focus was put into mapping Oracle technologies into this proposed architecture. Industry Standardisation bodies have begun carrying out work similar to the one proposed in this dissertation. The closest work to this research is the draft by National Institute of Standards and Technology (NIST) released in 2014 (NIST 2014). In this draft, NIST (2014) pointed out that, although there are differing opinions on important issues with regards to Big Data, there are some fundamental common questions that need to be addressed by the prospective Big Data architecture which is a superset of real time Big Data architecture.

1.1.5. Software architecture for Real-time Big Data Systems

There are three characteristics of Big Data. These are velocity, volume and variety, the so-called three Vs (Hurwitz et al. 2013). This dissertation describes research on Big Data systems with specific focus on the aspect of velocity. This is where the real-time processing of data that is confronting a software system at high speeds is a critical requirement. The data may be the stimulus for an action which may also need to be carried out in real-time. This is especially problematic in the realm of sensor networks, where the monitoring and actuation of large sensor networks becomes increasingly hard due to the vast number of sensors sending requests to the system at a high frequency.

From previous sections, it has been pointed out that reference architectures for Big Data are still under development. This is also the case with the subset of big data being real-time

Big Data systems. Since there is no software reference architecture for real-time Big Data systems, there is no standard approach or template for developing software architectures for real-time Big Data systems.

1.2. PROBLEM STATEMENT

Discussions in the previous sections have shown that there is no reference architecture of Big Data systems and real-time Big Data systems. This makes it inherently difficult to develop software systems that aim to solve real-time Big Data problems. Furthermore, it has been shown that prioritising which quality attributes to address is the first step towards developing a software architecture or a reference architecture. This has also been shown to be a non-trivial exercise. The research problem this thesis aims to solve is that of reducing the complexity of architecting systems whose requirements are that, they should assimilate and process Big Data in real time. By uncovering the most important quality attributes in real-time Big Data systems, the research would have laid the foundation towards the development of a reference architecture as well as providing guidance to architects in developing software architectures for Big Data systems.

1.3. RESEARCH QUESTION

- What are the most important quality attributes for real-time Big Data systems?

1.3.1. Sub-questions

- What are quality-attributes of a typical real-time Big Data system?
- How can ranked quality requirements be used to develop a software architecture for real-time Big Data systems?

1.4. OBJECTIVES

- Study and investigate existing and emerging ranking methods for software architecture quality attributes.
- Design a software architecture for a real-time Big Data system.

- Implement and evaluate a real-time Big Data system test bed.
- Propose an approach for selecting the most important quality attributes for real-time Big Data systems.

1.5. ORGANISATION OF THE THESIS

In Chapter 2 (LITERATURE REVIEW) ranking methodologies are reviewed. In answering the research question of what the most important quality attributes of real-time Big Data systems are, a ranking methodology must be employed. In this chapter, various ranking methodologies are examined to put the process of ranking into perspective and to justify the choice of the ranking methodology used in the research. To fully answer the research question, the case study needed context of other real-time Big Data systems. Part two of the literature review is an examination of other real-time Big Data system architectures reported in literature.

In Chapter 3 (METHODOLOGY AND RESEARCH QUESTIONS), the research question is stated, and details are provided on how it is to be answered. The ranking methodology is described in great detail. The software architecture design and how it will be presented is also detailed in this chapter. How the research case study relates to other real-time Big Data systems and steps towards answering the research question are explained in this chapter.

In Chapter 4 (RESULTS), the ranking methodology is applied to the requirements of the case study and the result is a ranked list of quality attributes which answers the research question within the context of the case study. The ranked quality attributes are then used to design a software architecture. The resultant architecture is implemented and tested against the quality attributes with the highest priority.

In Chapter 5 (DISCUSSION), the most important quality attributes of real-time Big Data systems are discussed from the results given in the case study and from the one specified in literature to fully answer the research question. How these ranked requirements helped in the design of the software architecture is also discussed. The implemented simulation and

the results of whether the most important real-time Big Data attributes were easily achieved through ranking are then discussed.

In Chapter 6 (CONCLUSION), a summary of the research study is presented. This is done by examining the research questions and determining from the findings, whether or not they were addressed. Objectives are also discussed in this chapter, and it is determined whether or not they were achieved. Future work is then suggested.

2. LITERATURE REVIEW

INTRODUCTION

The literature review covers three main topics which this research hinges on. The first topic covers scientific methods used to choose between competing attributes. When answering the question of which quality attributes are most important in real-time Big Data systems, the method for ranking these and comparison with what was found to be important by other researchers, are critical points to be addressed. The second topic is the work already done in developing real-time Big Data systems. Under this topic, real-time Big Data software architectures and the architectural attributes they address will be discussed. The last topic introduces the case study.

The first part of the literature review, which is a review of scientific ranking methods, presents a review of some of the most common ranking techniques. The chosen ranking technique is briefly discussed first, then others follow. A ranking technique is presented, its features discussed, and its advantages and disadvantages as compared to the ranking method which was ultimately chosen for this research. Common deficiencies between the methods are then discussed in comparison to the method chosen for the study.

In reviewing related work within the realm of real-time Big Data systems, the work done on this research is fitted into a larger perspective and a complete picture of the research is drawn. Quality attributes which other research reveals as important are uncovered. For this part of the literature review, only peer-reviewed published work not older than three years at the time of writing is considered. This is because Big Data and especially real-time Big Data has evolved and so has the technology. Only the current trends and design principles are more impactful.

Presentation of work done by other people in designing real-time Big Data systems is presented on a system basis. Published results/designs of real-time Big Data systems are reviewed one at a time in a total sample of seven. These are ones which the authors felt were most relevant and also kept the sample more diverse. Each system will be presented in four parts. These parts are:

1. Rationale
2. System description
3. System architecture
4. Quality attributes

The rationale describes the motive behind the creation of that particular, real-time Big Data system as expressed by the authors in that particular published material. The system description and architecture will be presented as originally presented by the authors. The quality attributes addressed by a particular system will be taken as explicitly stated on the literature, or extracted, if they were implicitly presented in the published material.

2.1. RANKING METHODOLOGIES

Multi-Criteria Decision Aid (MCDA) methodologies are widely used in the field of Operations Research (OR). These MCDA methodologies are classified into two groups, namely, multi-criteria utility function and outranking methods (Zhaoxu& Min 2010). Chen (2006) showed several ways of classifying MCDA methods; one example of this was classification using approaches in obtaining decision values. Chen (2006) and Triantaphyllou & Mann (1995) discuss the structure of a MCDA problem as a matrix consisting of N alternatives and Z criteria, where N is the total number of alternative decisions and Z is the total number of criteria.

2.1.1. Multi-Criteria Decision Making Based on PROMETHEE Method

Preference Ranking Organisation Method for Enrichment Evaluation (PROMETHEE) is a MCDA method which falls within the outranking methodologies (Zhaoxu and Min 2010). For one to use this method there should be decisions which have to be taken and several options per decision. There are also a number of criteria involved in making a decision to choose an alternative over other alternatives. This method came about as a result of a huge effort required on the decision maker to elicit preferential parameters in the construction

of a preference model from the original PROMETHEE. This method, however, tries to infer preferential parameters from the information given (Zhaoxu & Min 2010).

Alternative decision options are listed, and a pair-wise comparison is made. In each pair, outranking must be performed, where the decision maker has to provide the preference threshold and the indifference threshold. The preference threshold and indifference threshold will be used by a preference function. The preference function tries to model the difference in preference of each pair of alternatives from the decision makers' point of view. This is done per pair for each available criterion listed. After the preference function is done per pair in each criterion, weights of each criterion are inferred using linear programming.

This method aids decision making from multiple alternatives in multi-criteria situations as stated. Although it infers the weights of the criteria, it still relies heavily on the decision maker's opinion (Zhaoxu & Min 2010). This might be a problem in most software architecture design processes where either the decision maker has limited technical knowledge to make sound architectural decisions or the impact a decision has on the overall architecture cannot be estimated.

2.1.2. The Analytic Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) is a MCDA method that uses a hierarchy structure that spans multiple levels in aiding decision making (Triantaphyllou & Mann 1995). These levels are objectives, criteria, sub-criteria and the alternative choices that must be ranked. Under each individual decision criterion, there is a pairwise comparison of each alternative in order to deduce the relative importance of each design decision. This results in a decision matrix where N alternatives are paired together with Z criteria resulting in an N by Z matrix of performance values (Triantaphyllou & Mann 1995).

The alternative weight values which result from the pairwise comparison which resulted in the creation of the N by Z matrix have a value range from one to nine. One represents equal importance to two activities, in this case, requirements. Nine represents absolute importance, where one requirement has the highest overall importance over another. The

pairwise comparison is done for each sub-criterion. Then each sub-criterion has a value or weight, which adds up to the value or weight of each criterion. This now forms a hierarchy-like structure, and all values are evaluated along the hierarchy.

Assigning each criterion and sub-criterion weights is still the role of the decision maker, hence even in this method, the decision makers' opinion is still relied on. Heavy reliance on the decision makers' opinion has been described as a problem even in the PROMETHEE method as some decision makers may have limited knowledge on the software architecture process to make sound architectural decisions. The impact of choosing each alternative over another on the software architecture processes is also not taken into consideration.

2.1.3. Common deficiency

MCDA methods rely heavily on decision makers' point of view or preference to influence the ranking outcome. This has several disadvantages in software architecture design and implementation. Stakeholders or decision makers may have limited knowledge on software architecture and software development in general, which may result in an imperfect architecture design and implementation of the system. Even if these methods are used by the experts themselves, besides requiring a high cognitive effort, they are results oriented, not process-oriented. This, therefore, makes them blind to the impact of implementing certain requirements before others in the software architecture development process.

Most MCDA methods use pairwise comparisons to obtain relative importance of alternatives; Triantaphyllou & Mann (1995) did point out that the challenge with this is quantifying the verbally articulated choices selected by the decision maker.

Another important common deficiency amongst the reviewed methods is the consideration of relative performance during the computation of the values of importance that will influence the ranking process. This is a problem in that, for systems that do not have all the requirements already listed, the entire process of ranking would have to be done each time additional requirements are elicited, whereas for the method of ranking requirements based on the impact they have on the architectural process, additional requirements would have their importance or weight computed independently from other methods and then added to the sorted list of requirements.

2.1.4. Facilitating Software Architecting by Ranking Requirements based on their Impact on the Architecture Process

'Facilitating Software Architecting by Ranking Requirements based on their Impact on the Architecture Process' by Galster & Eberlein (2011) ranks requirements on the impact they would have on the architecture process. This method requires that each requirement be listed in an atomic form. Each requirement is given quality attributes that measure the degree to which that requirement would affect the architecture design process. These attributes are:

1. Effort
2. Risk
3. Complexity
4. Early design
5. Dependencies
6. Volatility
7. Hardware constraints
8. Aspects addressed
9. Importance
10. Familiarity

Values are assigned to these attributes for each requirement. Then the impact each requirement would have on the architecture process based on the value of these attributes is calculated (Galster & Eberlein 2011).

The advantage of this method is that it considers several qualities that have an important bearing on how a requirement affects the entire architecture design process. This therefore makes this method less prone to human error that may come from methods that just depend largely on stakeholder's perception of what architectural requirements are most important. This method focuses on both getting a quality system developed by addressing

the most important requirements of stakeholders and easing the process of architecting software by concentrating on the impact which implementing requirements in a certain order would have on the architecture process. Hence, this method was picked over the others presented. A more detailed description of this method is presented in the methodology section.

2.2. REAL-TIME BIG DATA SYSTEMS

In this part of the dissertation, the work done by other researchers when it comes to ranking requirements and developing real-time Big Data systems is reviewed. A comparison between these systems and the case study will be made in the discussion chapter of the dissertation. In the choosing the literature, the Institute of Electrical and Electronics Engineers (IEEE) and Association for Computing Machinery (ACM) libraries were selected. Only four of the most relevant papers were chosen to try and balance time spent on literature review and time spent to carry out the actual case study. Before arriving the final four papers, the initial sample size was seven, but all these papers could not meet the selection criterial explained in the following paragraph.

In general, a lot of papers were examined and based on the tittle and abstract, only seven paper were shortlisted or examined in detail. Of the seven shortlisted papers, three were further rejected or disqualified. That is, as mentioned in the introduction of this chapter, an actual real-time big data system must be presented in the paper and architectural requirements which the system addressed must be derivable from the paper. The three real-time big data systems that did not meet this selection criteria are presented first (section 2.2.1) and then the selected four real-time big data systems follow in the subsequent sections (2.2.2 to 2.2.5).

2.2.1. Disqualified real-time big data systems work

Surekha, Swamy & Venkatramaphanikumar (2016) presented work on Real-time Streaming, Data Storage and processing using storm and Analytics with Hive. What this work focuses on is how these technologies can be implemented to achieve the streaming, storage and processing or real-time big data. It does not provide context into the actual implementation of a real-world real-time big data system with real architectural requirements to address. Hence it is this precise fact that disqualifies it from being considered as part of the systems to be reviewed in this dissertation, as no architectural requirements can be explicit or implicitly derived.

Trinks & Felden (2017) presented work on real-time analytics as a state of the art. That is, the authors presented real-time analytics as a study field by first presenting analytics and all its different facets. Analytics was broken down and explained, classified and all its different associated terminologies defined. This work was an educational presentation and gave context into what real time analytics is, but it did not suit the purpose of this dissertation, which was deriving architectural attributes from real-time big data systems implementations. Because of this reason, the work presented by Trinks & Felden (2017) was also disqualified.

Patil & Seshadri (2014) presented work on big data security and privacy issues in health care. This work highlighted the state-of-the-art privacy issues in big data as applied to the health care industry. Real time big data systems in the health industry are introduced and privacy and security concerns like patient data theft are highlighted. This work already has a specific architectural requirement, being security which it focuses on. An actual real-time big data system that serves this purpose and its design process is never actually presented. No architectural requirements can be derived to be ranked from this work as no actual system is presented.

2.2.2. Real-time Smart transportation system

Paul et al. (2016) discussed a real-time system to enable smart transportation. The motivation which the authors had in developing such a system was the increasing demand of real-time traffic information in urban areas which continues to grow because of

urbanization. The authors argued that, the increased population density in urban areas caused by urbanization also causes traffic density which in-turn increases the likelihood of accidents and traffic jams as people try to get to their destinations. There was, therefore, a need for the authorities and commuters to have real-time access to traffic information. This would enable the authorities to better manage traffic and commuters to better navigate to their destinations (Paul et. al. 2016).

The smart transportation systems would require the use of millions of devices using the Internet of Things (IoT)technology (Paul et. al. 2016). The large number of devices would be vehicular sensors and road sensors. Besides having large volumes of devices, another important aspect of the smart transportation system was the transportation infrastructure which was represented by a graph data structure. Big cities with thousands of roads populate this Big Data structure to form what the authors called a big graph (Paul et. al. 2016). The vast number of devices feeding data into the system also generated big graphs which in-turn posed a processing problem as the result from the system was expected in near real-time conditions.

The intelligent transport system uses a layered architectural pattern. The system has mainly two blocks connected together through the Internet. Figure 2 shows the system architecture of the intelligent transportation system. These two main blocks are the Smart Transportation Building (STB) which house the main system where all the collection, processing and storage of the data happens, and the sensor network that lies on top of the IoT infrastructure.

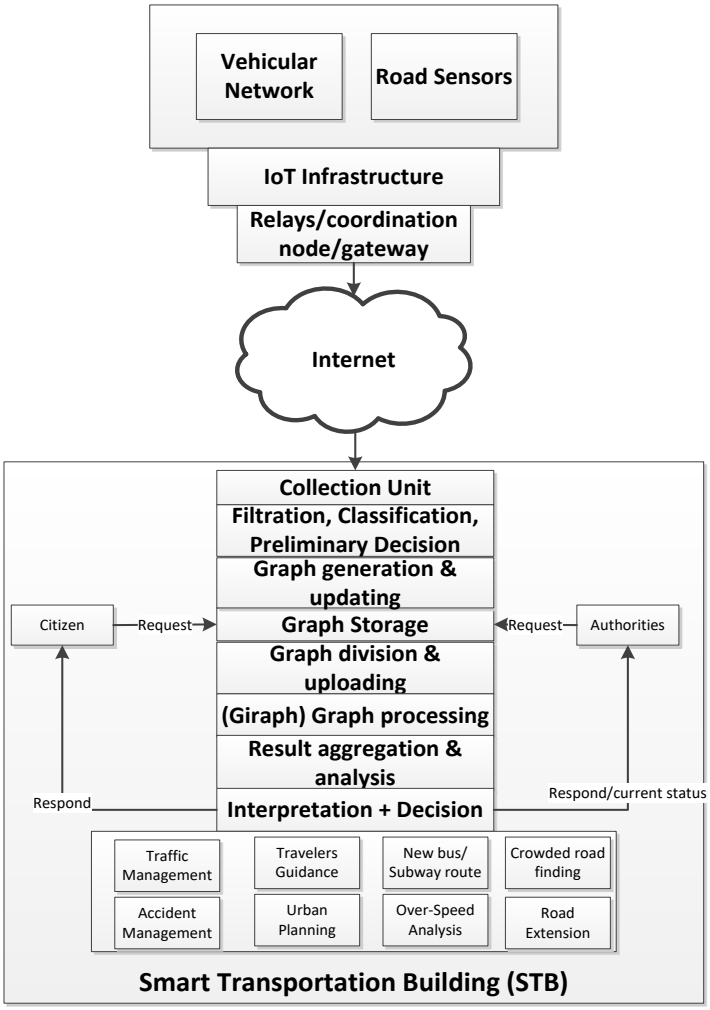


Figure 2: Architecture of the real-time smart transportation system (Paul et. al. 2016)

The sensor network is comprised mainly of vehicular network sensors and road sensors. These are the vast number of data generating agents of the system. Relays, coordinator nodes or gateways are used to collect data from various sensors of the system and transport it through the internet to the main nerve centre housed in STB. Once the data is at the STB, it passes through several stages as depicted in figure 2, at several layers of the system, where a graph is created or updated with new information, stored, processed, analysed and results are interpolated and decision making happens. The two main users of the system are citizens and authorities.

The system had to capture fast coming data from its vast sensor networks and gateways were used to collect and transport data to the STB. At the STB, the collection unit also used fast capturing hardware to capture the fast, incoming, relayed data from the sensor networks. The graph was stored using the Hadoop Distributed File System (HDFS) (Paul et. al. 2016). The HDFS had a special tool called Giraph which would be used for fast processing of the graph. Huge graphs would be broken down into sub-graphs and processed in parallel using the tools mentioned above and other Hadoop eco-system tools such as Map-Reduce (Paul et. al. 2016).

Table 2: Quality attributes of the real-time smart transportation system

Architectural requirement	**Quality Attribute**	**Architectural Strategy**	**Instance of strategy**
Efficiently process large graphs quickly	Performance	Scaling-out	Parallel processing
Fast capture of data (response time/throughput)	Performance	Scaling-up	Specialized high-performance hardware

As can be seen by table 2 above, the only quality attribute reported by the authors in the smart transportation system is performance. This is not surprising as the smart transportation system is a real-time Big Data system. Performance being the only reported quality attribute does not mean there were no other quality attributes which were dealt with, but the most important one to the authors in their system seems to be performance. The architectural requirements which would be addressed by ensuring system performance were fast processing of large graphs and the fast capture of data. The overall architectural requirement of the system seems to have been satisfied by making the system have high performance.

2.2.3. Real-time processing of streaming data system

Batyuk and Voityshyn (2016) explain the need to handle large amounts of data in real time to take advantage of the insights that could be derived from such data. The data in question was generated by social media, had to be accumulated and then analytical models built

from it. The authors furthermore pointed out that the challenge to handle Big Data in real time meant that the software architecture design approaches needed to be revised. The authors designed a process to stream real-time data from social networks using Apache storm based on an approached called "Lambda Architecture" (Marz & Hening 2014).

There are generally two main components involved when streaming real-time Big Data; the algorithms that achieve the functional goal and the software infrastructure that the entire system runs on. The authors concentrated on the software infrastructure. There were already architectural requirements which were explicitly stated by the authors (see table 3 on page 20) and an Apache storm-based architecture was developed to address these.

At the highest level of granularity, the system followed a Lambda architecture which is based on a layered architectural pattern. The layers of the Lambda architecture are usually the stream layer and the batch layer. Figure 3 depicts an architectural concept of the system.

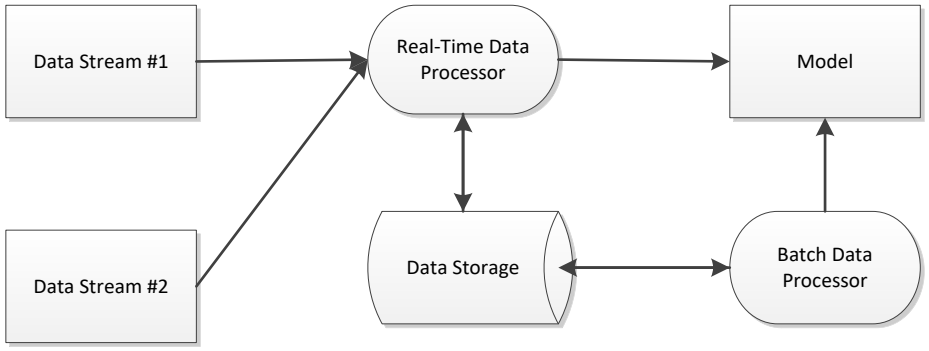


Figure 3: architectural concept of the real-time processing of streaming data system (Batyuk and Voityshyn 2016)

The real-time data processor was the main focus of the authors. This was implemented using Apache storm. The real-time data processor also followed a layered architectural pattern. The layers are the formatting, aggregation, analytics and storing layers. How these layers are structured, and their connections are depicted below in figure 4.

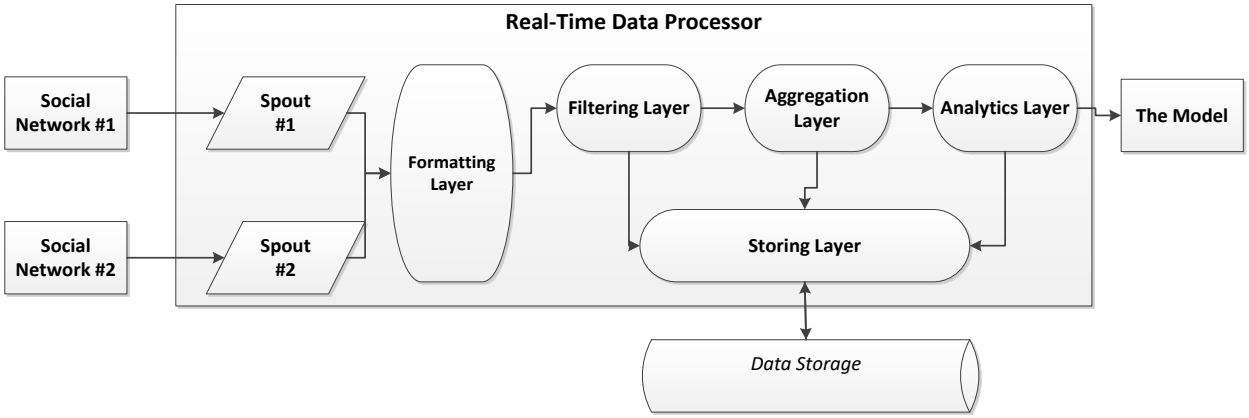


Figure 4: Architectural layers of real-time processing of streaming data system (Batyuk and Voityshyn 2016)

There were several architectural strategies and tactics employed to ensure the architectural requirements listed in table 3. Some of the associated quality attributes of the architectural requirements were addressed as follows; choosing the Apache storm framework as a real-time processing platform ensured high performance quality attribute was achieved. Furthermore, scaling out in the form of clustering and employing Apache Trident which was configured to have in-memory data streams for processing also ensured the most important quality attribute of performance being realized. Having the spouts from Apache Storm adapt data from different sources, and then converting it into a unified format ensured interoperability, which was the second most important quality attribute. A database and an internal queuing mechanism with store and forward capability ensured reliability (Batyuk and Voityshyn 2016).

The architectural requirements together with their associated quality attributes which were ranked were explicitly stated by Batyuk and Voityshyn (2016). The quality attributes of a real-time processing of stream data from social media based on Apache Storm topology are depicted in table 3.

Table 3: Quality attribute of real-time processing of streaming data system

Architectural requirement	**Quality Attribute**	**Architectural Strategy**	**Instance of strategy**
High throughput	Performance (rank 1)	Scaling-out	Parallel processing
Low latency	Performance (rank 1)		
Support for heterogeneous data sources	Interoperability (rank 2)	Data conversion	
Support for heterogeneous protocols and data formats	Interoperability (rank 2)	Adapters	Spouts
Fault tolerance (tuple processing)	Reliability (rank 3)	Queueing	Internal messaging system
Fault tolerance (model temporarily unavailable	Reliability (rank 3)	Persisting data	Using internal Storage
Architecture evolution	Maintainability (rank 4)		
Provide information about failures	Maintainability (rank 4)		Logging (For repair)
Protect communication points	Security (rank 5)		

From table 3 above, it is evident that performance as the most important quality attribute for the system, was in the form of real-time processing. The data which the system was processing was from numerous different social networks; hence interoperability was the second most important quality attribute. Lastly, because the intent of making this system, which was to gather, process and analyse data from social networks in a timely manner, reliability, maintainability and security followed respectively as the least important quality attributes.

2.2.4. Processing and analytics of big data streams with Yahoo!S4

Xhafaet et al(2015) describes a system of processing big data streams using yahoo!S4. The need for such a system arose from the realisation that batch processing of Big Data, using the Hadoop ecosystem, does not work for real-time systems. The real-time systems which need real-time processing as mentioned by Xhafa et al (2015) were:

- IoT –based system
- Monitoring online workspaces and Massive online open courses
- Global flight monitoring systems
- transaction/credit card fraud detection
- security thread detection prediction
- network fault prediction from sensor data

This therefore led to the authors building a system on an experimental basis using Yahoo!S4 to showcase online stream processing as opposed to the batch processing of traditional Big Data systems (Xhafa et al. 2015).

The system built by Xhafa et al (2015) came from the realm of global flight monitoring systems. Information about aircraft movement around the world from flight-tracking services would be collected and fed to the system in real-time for processing. This would make quick decision-making possible through real-time or near real-time processing of Big Data streams.

The tracking services that would be used here are the radar system and the transponders on-board most aircraft. The transponders would feed information to a worldwide network of radio receivers that use Automatic Dependent Surveillance-Board (ADS-B) technology (Xhafa et al. 2015). This flight dataset is periodically published online.

The typical flight dataset contains between four thousand to six thousand records. Each record contained flight specific information like flight name, longitude, latitude and many others. The result for each record computed in real-time, would be departure airport, destination airport, flight status, landing track optimum and many more (Xhafa et al. 2015).

At the highest level of granularity, the architecture presented by Xhafaet. Al (2015) follows a layering architectural pattern where the server cluster presented in figure 5 is separated by layers as each layer has different concerns. From figure 5, a Laboratory Research and Development (RDLab) cluster as developed by the authors can be seen where on the first layer, login and cluster control servers were set up, then on the second layer, compute nodes were set up and finally, the third final layer show the disk arrays for storage.

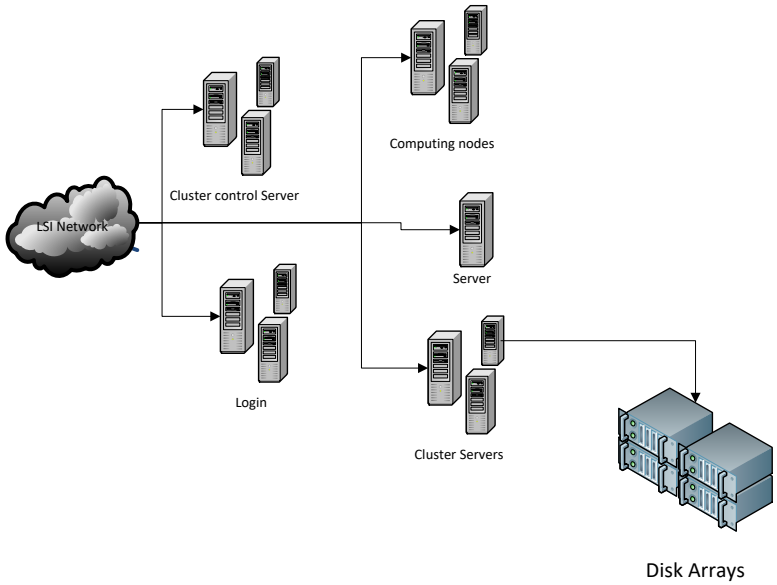


Figure 5: System Architecture of Processing and analytics of big data streams with Yahoo!S4 (Xhafa et al. 2015)

While developing this architecture, certain architectural strategies and tactics were followed in ensuring high performance is achieved. One of these were scaling out, where over 160 servers comprised system through clustering. Scaling up was achieved through increasing the power of resources. The memory was increased to 3 Terabytes, disk space to 130 Terabytes and the network throughput to 10 Gigabit (Gbit) speeds. Other quality attributes like fault tolerance, scalability and extensibility were ensured through picking Yahoo!S4as the execution environment.

Table 4: Quality attributes of Processing and analytics of big data streams with Yahoo!S4 system

Architectural requirement	Quality Attribute	Architectural Strategy	Instance of strategy
Quick response/low latency	Performance	Scaling-out and scaling up	Parallel processing, clustering, increasing resources
High throughput	Performance	Scaling-out and scaling up	Parallel processing, clustering, increasing resources
System should grow to handle large volumes of data	Scalability	Scaling out	Clustering
When a server in a cluster fails, it should be replaced by one on standby	Fault tolerance	Passive redundancy	Clustering
Hide complexity	Extensibility	Modular development	Use of simple API

Table 4 above shows the quality attributes of a system to process and analyse big data streams using Yahoo!S4 as extracted from a paper written by Xhafa et. Al (2015). Throughout the paper, the quality attribute which was emphasized was High Performance. Other quality attributes as shown in table 4 were also reported but not with as much emphasis as performance.

2.2.5. Real-time Big Data Analytical Architecture for Remote Sensing Application

In the current digital world, remote sensing results in massive amounts of data being generated (Rathore et al. 2015). Insights from this data need to be extracted, which means the effective collection and aggregation of this data is of paramount importance. There is however, a challenge when using conventional computing platforms in collecting, storing, extracting and processing these massive amounts of data. There is therefore a need for a software architecture that would handle the processing of both real-time and batch type data.

Rathore et al. (2015) proposed a real-time analytical Big Data architecture that would handle the collection, storing, processing and analysis of remote sensing satellite data. Satellites in orbit, taking images of the earth using sensors or specialized cameras would act as data collectors. Special techniques would then be applied to the satellite imagery collected by the remote satellites in order to produce maps, resource surveys and other related information.

The system architecture was designed using a layered architectural pattern. It was divided into three layers which were namely:

- remote sensing data acquisition unit (RSDU)
- data processing unit (DPU)
- data analysis and decision unit (DADU)

The remote sensing Big Data architecture is shown in figure 6 below.

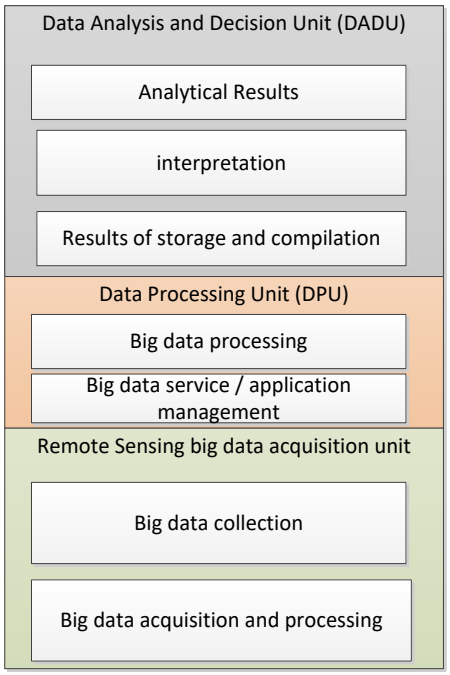


Figure 6: Remote sensing Big Data architecture (Rathore et al. 2015)

The RSDU unit is mainly used for acquiring data from various satellites orbiting the earth. The satellites pre-process the data and send it to ground stations. At ground stations, at the

DPU, the data is processed in two ways which are real-time and offline processing, also known as batch processing. The real-time data is filtered and sent to the load-balancing server which then divides the data among computing nodes. The offline data is sent to a data warehouse for the traditional approach of processing. The DADU is then responsible for storing, interpretation and analysis of results. Figure 7 depicts a flowchart of the remote sensing big data architecture. The architecture flowchart shown in figure 7 shows the steps of the operation of the system, with respect to its architectural responsibilities.

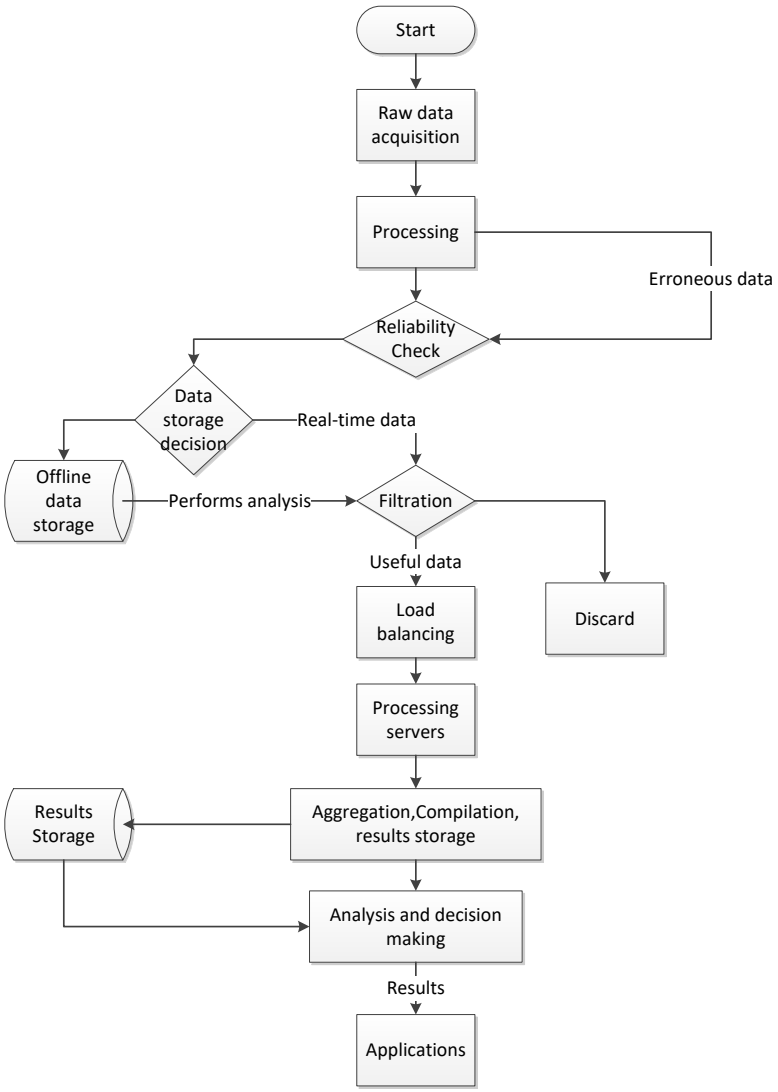


Figure 7: Architecture flowchart of remote sensing Big Data architecture (Rathore et al. 2015)

Table 5 shows the architectural requirements of the remote sensing big data software architecture.

Table 5: Quality attribute of remote sensing Big Data software architecture

Architectural requirement	Quality Attribute	Architectural strategy	Instance of strategy
Process high volumes of data	Scalability	Scaling out	Parallel processing
Correct data and remove distortions	Accuracy	Scaling out by time (Filtering)	Pre-processing
Real-time processing performance	Performance	Load balancing/ clustering	Clustering

2.3. CASE STUDY OF THE SPECTRUM SENSING ENABLED WHITESPACE DATABASE (SSEWDB)

The problem domain covered in the research presented in this dissertation is that of telecommunications and in particular, dynamic spectrum access. National Communications regulators all over the world are opening licensed but underutilised radio frequency spectrum in the terrestrial broadcast television (TV) bands. The Very High Frequency (-VHF) and Ultra High Frequency (-UHF) bands are designated for sharing usage basis between the incumbent user (licensed) and secondary user (unlicensed). These secondary users will be using this spectrum for provision of broadband internet connectivity to un-served and underserved communities particularly in rural areas. However, the secondary users are required by the regulators, to not cause any harmful interference to the licensed users. The techniques used to prevent interference are spectrum sensing, beacons and applications called Geo-Location Spectrum Databases (TV white space Databases).

The term, TV White Spaces (TVWS) refers to frequencies in the TV band spectrum which are not locally used by the licensed users (e.g. TV broadcasting services like South African Broadcasting Corporation(SABC)) (Baykaset et al. 2010).In describing what a TVWS

Database (WSDB) is, Murty et al (2012) said, it's a service that contains an updated database of base stations and white space devices (WSDs) [users] in active operation at a particular location and based on that predict the available white spaces (free frequencies or channels) using propagation modelling and by including digital terrain data (as the terrain influence the signal propagation) as well as TV-Tower specific parameters, such as antenna height, etc. Examples of TVWS Databases are, the one whose developments the author has been part of, the CSIR (2013) TVWS database in South Africa, Google US (2016), Microsoft (2015) and several other companies that also possess such Databases.

In Spectrum Sensing, sensors to scan the frequency band for frequencies that are being used are deployed (Weiss et al. 2010). Enabling a whitespace database with spectrum sensing is simply establishing a connection between the sensors that scan for free radio spectrum with the prediction engine or capability of the whitespace database (Harrison and Sahai 2014). There would be many sensors deployed and they would have to all connect to the system which would have to serve users querying for free spectrum in real time. The summarized and oversimplified picture that depicts a Spectrum Sensing Enabled Whitespace Database is depicted in Figure 8.

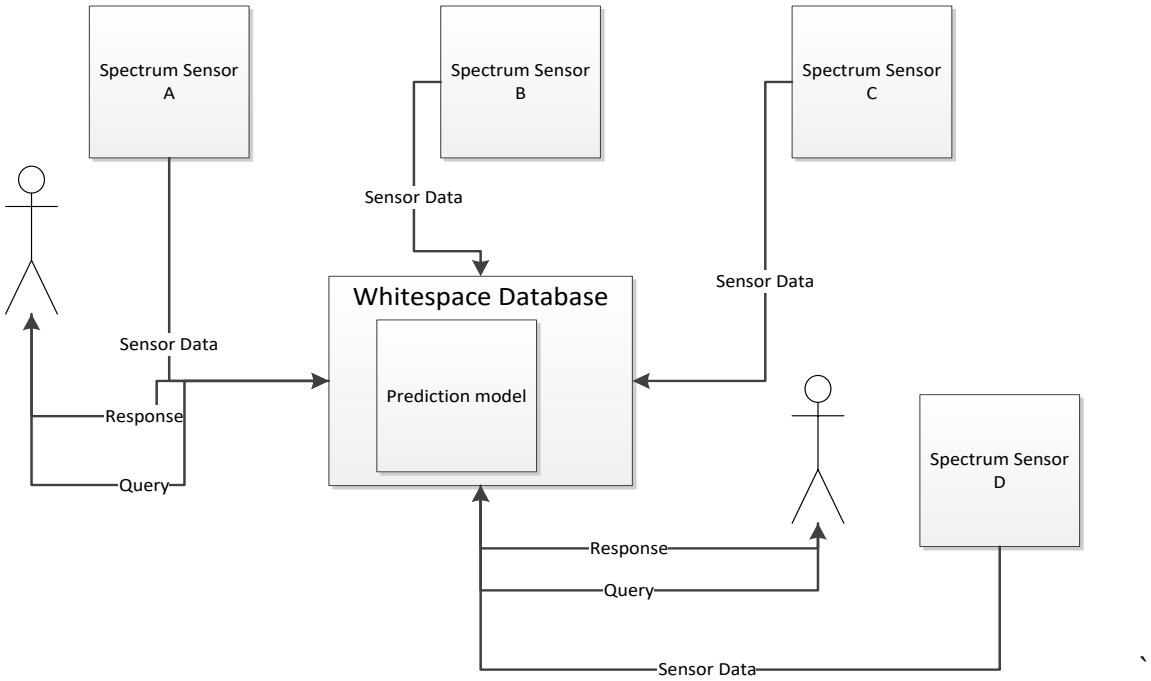


Figure 8: Spectrum Sensing Enabled Whitespace Database (SSEWDB)

When a user close to Sensor A queries the systems, his query triggers the prediction model as well as getting data from Spectrum Sensor A. The data from Spectrum Sensor A is then compared with the results from the prediction model and then a result is issued to the user. The same process applies for the user standing close to Spectrum Sensor D. The WSDB system must be able to deal with requests of users in real time and deal with numerous users doing requests and numerous sensors feeding it information to continuously enhance the performance of the WSDB system. This qualifies the spectrum enabled whitespace database as a real time Big Data system. Mfupe et al (2013), Lysko et al (2014), Masonta et al (2015), and Mfupe et al (2015) are some of the authors that further explain the concepts of TV whitespaces and TV whitespace databases.

2.4. CONCLUSION

In this chapter different ranking methods were reviewed. The advantages and disadvantages of these methods were discussed. It was shown that with regards to the process of architecting software systems, the method *'Facilitating Software Architecting by*

Ranking Requirements based on their Impact on the Architecture Process' by Galster & Eberlein (2011) is most suitable. Real-time Big Data systems developed by other researchers in peer-reviewed published material were also reviewed. A list of quality attributes which the software architecture of each system was addressing was reported in each system review.

3. RESEARCH METHODOLOGY

3.1. INTRODUCTION

This chapter outlines the steps that were taken in answering the research question explained in chapter one. Each decision that was made in each step of answering the research question is also justified in this chapter. The general research paradigm that was chosen for this research is design science. The design paradigm and steps taken in answering the research question are discussed and justified in the following sections.

3.2. DESIGN SCIENCE RESEARCH PARADIGM

3.2.1. Paradigm Description

In the field of Information System and Software Engineering, there are common research paradigms which include, but are not limited to, empirical research, exploratory research and constructive research (Wang 2011). Design Science research and constructive research are deemed the same and both can be classified as design-oriented frameworks (Piirainen & Gonzalez 2013). In empirical research, a phenomenon is experimented upon or observed with the aim of establishing a pattern that may lead to the realization of a certain nature of the phenomenon. Exploratory research employs investigative means to gain insight into a problem that has not been clearly defined (Wang 2011). Design Science on the other hand, is a pragmatic research paradigm that solves real world problems through the development of innovative artefacts.

3.2.2. Justification of design science research paradigm

Hevner et al. (2004) outlined some of the characteristics of problems that design science research addresses. It is these characteristics that were used in determining the suitability of employing design science in the study.

Design science research addresses problems with unstable requirements (De Silva et al. 2014). Generally, in the field of Big Data, requirements are not clear, especially in real-time

Big Data systems like the SSEWDB where this type of system is not common. The interactions among SSEWDB - system, the sensors, white spaces devices and users, in real-time is complex. One other characteristic that Hevner et al. (2004) outlined is that problems addressed by design science research have complex interactions among sub-components. The most important justification is that, unlike the scientific method, where the experiment validates the premise, in design science, theories are developed after development of the artefact. In which case developing any knowledge on the research question only came after the design took place. Problems that require creativity to be exercised in problem solving also fall under the umbrella of problems addressed by design science (De Silva et al. 2014). Even though ranking requirement is a straight forward mathematical procedure, the design of the software architecture for the SSEWDB which, to the best of the researcher`s knowledge, has not been done at the time of writing this report, will require a great level of innovation and creativity.

3.2.3. Outline of design science research

It has been mentioned earlier that in design science, deeper understanding of the problem will come as an effect of whatever artefact is made. Hevner et al. (2004) argued that by building and evaluating artefacts which are addressing identified business need, it is then that research will be made. The guidelines for carrying out design science research were also outlined by Hevner et al. (2004) as follows:

Design the artefact, which means that the goal of design science is to have a viable artefact as a product. This could be in a form of a method, construct or instantiation. In terms of this study, this was the ranked list, software architecture and system simulation implementation. When dealing with problem relevance, Hevner et al. (2004) stated that, the developed artefact must be addressing a particular problem. In this instance, the ranked list is answering the main research question; the software architecture is answering one of the research sub-questions in showing how ranked attributes can be used to develop an architecture, while the system itself is just further means of validating the work.

3.3. SOLUTION STEPS

This section details the steps taken in answering the research question. These steps also describe the flow of the entire research. These steps show the role of the case study, literature review and how they all fit together in answering the research question. Figure 9 summarises the steps taken to fulfil the research.

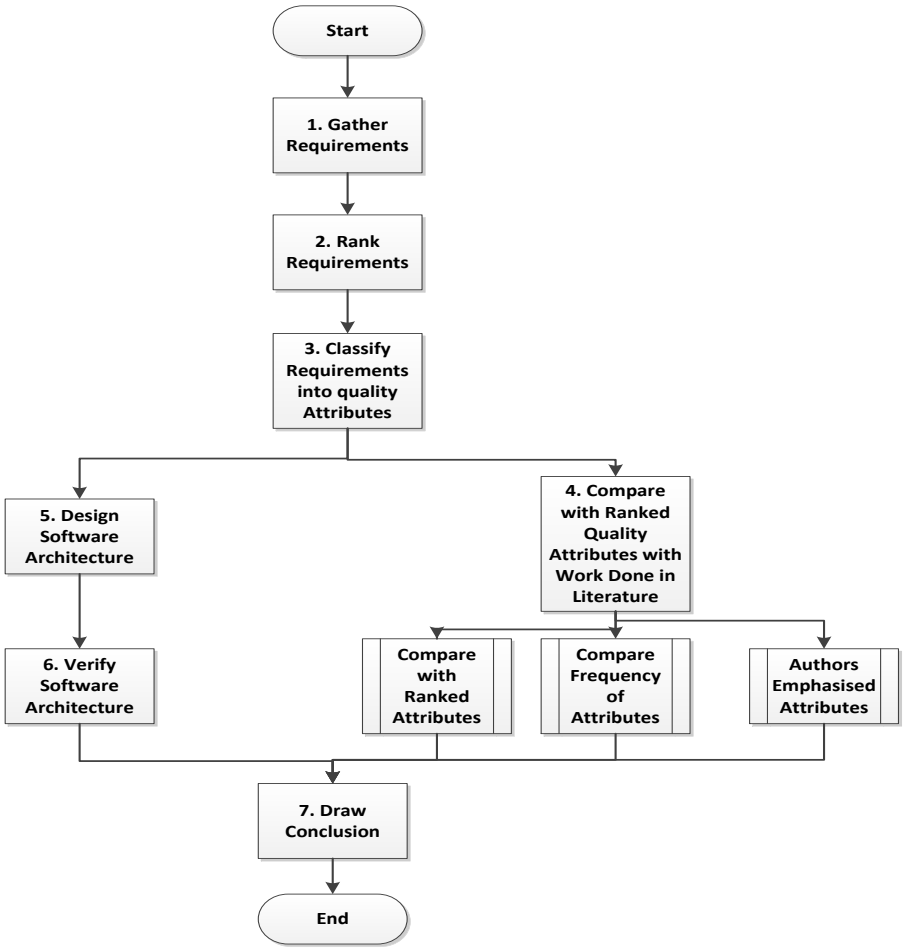


Figure 9: Research solution steps flowchart

3.3.1. Requirements Gathering

In figure 9 above, Step 1 titled *gather requirements* is the initial stage of the research. The CSIR provided a list of architectural requirements as input for the SSEWDB. These requirements are non-functional in nature and describe how the system is expected to behave rather than what it is expected to do. The list of architectural requirements is

expected to be atomic in nature, meaning that one requirement should specify one desired behaviour of the system, not multiple characteristics wrapped as one.

3.3.2. Ranking Requirements

The process of designing software architecture is not as straight forward as application design. For example, if an architect was told to design a system that is secure and easy to access at the same time. This in nature is a paradox. It therefore comes in handy to know the priority of architectural requirements. To achieve this, ranking requirements is key, and there are several scientific methods to choose from as shown in the literature, but the method chosen to achieve this task is, *'Facilitating Software Architecting by Ranking Requirements based on their Impact on the Architecture Process'* by Galster & Eberlein (2011).

This method addressed the problem that requirements techniques only focused on resource constraints and stakeholder priorities and did not take into consideration the effect which requirements have on the software engineering process. Galster & Eberlein (2011) pointed out that the order in which requirements are implemented affects the architecture process.

The method comprises of three steps, which are namely:

- Preparation
- Execution
- Presentation

These stages are directly mapped to the task that is being done on each stage respectively. That is, in preparation stage, the requirements are being prepared for processing by the method. The second stageis called execution. In this stage, the actual ranking of quality attributes is being performed using all the information supplied in the preparation stage. In the last stage, which is the presentation stage, the ranking index is calculated which will

then be used to present the outcome of the method to the stake holders (Galster & Eberlein 2011).

3.3.2.1. Preparation

In presentation stage, the data that is given as input to the method is made ready for the processes that it is about to undergo. That means the requirements are listed in an atomic form. Atomic requirements are those that address one aspect of the system. What the authors mean by that is that the quality attribute which a requirement is classified as must not be more than one. This process may occur right after the process of requirements engineering.

3.3.2.2. Execution

At this stage, each requirement is given attributes which would help foretell or deduce the impact of implementing that requirement on the entire architecture design process. These attributes were to be seen from an architect's point of view. Ten attributes were chosen to be applied and measured for each requirement. These were chosen using a goal question like metric, where the goal was describing the impact of implementing individual requirements on the architecture process. Questions which one would ask about the impact of implementing each requirement were summarised by the ten attributes which were uncovered, where each attribute emanated from a question which one would ask.

The requirements attributes are to be listed, and then given a short description and a value score. The ten attributes which are derived for this method and were to be applied in each requirement were namely:

1. Effort
2. Risk
3. Complexity
4. Early design

5. Dependencies
6. Volatility
7. Hardware constraints
8. Aspects addressed
9. Importance
10. Familiarity

To help rank requirements, each of these attributes was given a value score which ranged from one to five, in some cases, no or yes. For example, taking the familiarity quality attribute, it may be a requirement to implement a security feature using a specific encryption scheme, so a developer's familiarity with the encryption scheme in question may be one of the attributes of the requirement. Familiarity and all other requirement attributes had to be measured and given scores, so one being low and five being high was the score given. Table 6 below shows the requirements attributes (rn) and their associated possible values with short descriptions.

Table 6: Requirements attributes (Galster & Eberlein 2011)

Quality Attribute of Requirement	**Explanation**	**Value**
effort	Estimated cost of implementing *rn*	1 = low 2 = low-medium 3 = medium 4 = medium-high 5 = high
Risk	Expected risk of failing to implement *rn* and potential to introduce errors into the software	1 = low 2 = low-medium 3 = medium 4 = medium-high 5 = high

complexity	Estimated difficulty of implementing *rn*	1 = low 2 = low-medium 3 = medium 4 = medium-high 5 = high
early design	Existence of design information implied by *rn*	yes no
dependencies	Dependencies of *rn* with other requirements	Any combination of requirements in requirements set *R*
volatility	Likelihood of *rn* to change (this includes anticipated change, i.e., "changeability", or un-anticipated change, i.e., "flexibility")	1 = low 2 = low-medium 3 = medium 4 = medium-high 5 = high
hardware constraints	Existence of hardware constraints imposed by *rn*	yes no
aspects addressed	System aspects addressed by *rn*	internal communication external communication user interface business logic
importance	Importance of *rn*as perceived by stakeholders	1 = low 2 = low-medium 3 = medium 4 = medium-high 5 = high
familiarity	Developers' knowledge and experience related to *rn*	1 = low 2 = low-medium 3 = medium 4 = medium-high 5 = high

3.3.2.3. Presentation

The third and final stage of the process is called presentation. In this stage, a ranking index for each requirement is computed using all the attribute values derived from the execution stage. The higher the computed ranking index for each requirement, the more important that requirement will be. Calculating ranking index is a four-stage process. The four stages of calculating ranking index are:

- Defining impeding forces for each requirement.
- Defining enabling factors for each requirement.
- Calculating the impact of both the impeding and the enabling forces for each requirement on its ranking index.
- Calculating the ranking index for each requirement.

For both impeding and enabling forces, only attributes with ordinal values are used. Out of the ten attributes, attributes with ordinal values are: effort, risk, familiarity, importance, volatility, and complexity. The remaining four attributes for a given requirement will be used in a later stage of post- processing.

3.3.2.4. Impeding Forces and Enabling Factors

The impeding forces were defined as attributes of a requirement which make it difficult to implement that requirement. Table 7 below shows potential impeding forces that any requirement might have.

Table 7: Potential Impeding forces (Galster & Eberlein 2011)

Attribute	Criterion
effort	Impeding force if attribute value has the highest value among (*effort*, *risk*, *complexity*, *volatility*, *Importance*). If more than one attribute shares the highest value, more than one impeding force exists.
risk	

complexity	
volatility	
importance	
familiarity	Impeding force if value of *familiarity* is "low".

Enabling forces on the other hand, are attributes which support the implementation of a given requirement. Table 8below now shows in contrast, the potential values of the attributes which would make them enabling factors.

Table 8: Potential enabling forces (Galster & Eberlein 2011)

Attribute	**Criterion**
effort	Enabling factor if attribute value has a value of "low". If more than one attribute has a value of "low", more than one enabling factor exists.
risk	
complexity	
volatility	
importance	
familiarity	Enabling force if value of familiarity is "high".

3.3.2.5. Impact of Impeding Forces and Enabling Factors on Ranking Index

Impact is defined as the direct and inverse proportionality between both impeding forces and enabling forces with ranking index of a requirement respectively. Let $<I_n>$ denote impeding forces and $<E_n>$ denote enabling factors. The impact that $< I_n>$ has on the ranking index $<\rho_n>$ is denoted as $<II_n>$. The impact that $<E_n>$ has on $<\rho_n>$ is also denoted as $<EI_n>$. Then equation 1 below denotes the impact of $<II_n>$ on $<\rho_n>$.

$$II_n = isIF(risk) \times (-1) \times valueOf(risk) + isIF(volatility) \times (-1) \times valueOf(volatility) + isIF(importance) \times valueOf(importance) + isIF(familiarity) \times (-5) \times valueOf(familiarity) \quad (1)$$

Equation 1 means that if all attributes are impeding forces, then they are all summed up to obtain the impact on the ranking index for that particular requirement n. If volatility and risk are impeding forces, the values of those particular attributes are negated. This is mainly because both risk and volatility have a negative impact on ρ_n. Effort and complexity are not considered when calculating II_n as they require additional information to have influence on the order in which requirements are implemented. In the Equation 1 above, if an attribute is an impeding force, then the value thereof, is one, else zero. That is, isIF(<attribute>) = 1 if <attribute> is an impeding force, else 0 (Galster & Eberlein 2011).

When calculating EI_n on ρ_n, attributes with the value of "low" are the only ones considered with the only exception being familiarity with the value of "high". The values of attributes when calculating EI_n, are the same, unlike the values of attributes when calculating II_n. This therefore implies that EI_n, by multiplying the number of E_n by the highest value of which an attribute can attain is five. A high EI_n, is always preferred for a requirement to have a good positive impact on ρ_n. Equation 2below denotes the calculation of EI_n.

$$EI_n = 5 \times |E_n| \quad (2)$$

After the calculation of both the II_n and EI_n, the ranking index ρ_n must be calculated. Ranking index, ρ_n is calculate by weighing II_n and EI_n against each other, that is weighing impeding forces against enabling factors ($II_n + EI_n$). The value of the requirement attribute importance is now used at this stage of calculation of ρ_n. Importance is applied to ($II_n + EI_n$) by means of multiplication in different ways depending on the value of ($II_n + EI_n$). Equation 3 below shows the complete calculation of ρ_n.

$$\rho_n = \begin{cases} importance_n \times (II\text{n} + EI_n)\ iff\ (II\text{n} + EI_n) > 0 \\ importance_n\ iff\ (II\text{n} + EI_n) = 0 \\ (1/importance_n) \times (II\text{n} + EI_n)\ iff\ (II\text{n} + EI_n) < 0 \end{cases} \quad (3)$$

From the above equation, ρ_n can be calculated for every requirement n. After the calculation, a priority list can be obtained, showing the most important requirements. Hence, the implementation order when architecting a software system. There are other requirements attributes which were not used in the calculation of ρ_n, these are early design, hardware constraints, dependencies, and addressed aspects. The use of these requirements is to further create sub-groups out of the priority list of ranked software architecture requirements (Galster & Eberlein 2011).

After the list of ranked software architecture requirements is produced, then each requirement shall have a corresponding quality attribute which it falls under. This, in effect, is step 3 of the research approach shown in figure 9.

3.3.3. Comparison of ranked quality attributes with literature

Subsequently, a list of ranked quality attributes is produced; it is not enough to answer the research question. In answering the research question, of what the most important quality attributes to real-time big data systems are, one must compare the ranked list of quality attributes with what has been reported by other architects in literature. This will corroborate or dispute the findings made in the dissertation. In other words, revealing whether or not the resultant list of ranked quality attributes from the research is consistent with other work reported in literature gives validation of the dissertation.

Comparison is achieved in three ways:

- The first comparison is when the literature reports have already explicitly ranked quality attributes. This is straight forward comparison of the two ranked quality attribute lists.

- The second one is that of comparing the ranked quality attribute list from the study with the frequency of the occurrence of the quality attribute. That is quality in an ideal situation, quality attributes higher up the resultant ranked list would be expected to have a higher frequency in occurrence throughout the reported work in the literature.
- The third comparison is using the perceived emphasis which the author of this research had when reviewing the literature material. That is, the author's perception on which quality attribute was emphasis being place on by the authors of the reported literature material and comparing that with the ranked list in this research.

3.3.4. Software Architecture Design

When the architecture requirements have been ranked and mapped to their quality attributes, a software architecture is designed as denoted in step 5 of the research approach framework. This is done to further validate the work and answer the research sub-question of how a software architecture could be specified using ranked quality attributes. Such that, using the ranked quality attributes, a software architecture for the SSEWB is designed and the users, being the CSIR, had to approve. This would mean the users deem it to address the requirements they had. Some quality attributes, like accuracy, denote some elements on functional requirements presented. So, Architecture description languages which focus exclusively on non-functional (quality) attributes would not fully present the architecture of the system. It is for this reason that the Krunchten4+1 architectural methodology was used. This is mainly because most architectural description languages do not encompass application design whereas the Krunchten 4+1 methodology does.

The scope of the software architecture design would be as presented by Solms (2012), consisting of four parts, namely:

- Basic concepts and constraints

- Architectural Components
- Access and integration channels
- Architectural Strategies

Basic concepts and constraints are software architecture specific concepts. An example of a basic concept and constraint is that in SOA, a service must be stateless (Solms 2012). Architectural components refer to a set of software components addressing technical concerns. Access and Integration channels refer to how software services are accessed and how software components communicate. Architectural strategies are mechanisms and tactics through which quality attributes are addressed.

3.3.5. Software Architecture Verification

Software architecture verification is performed in accordance with the requirements stated by IEEE 1012-2004(2004) standard. For the research to achieve this, an experiment of the system that was intended for the architecture had to be build. The most important quality attributes to the client are tested and they reveal the conformity of the architecture to the given quality requirements. That is, the most important quality attributes to the client were tested in an experiment, demoed to the client and a sign off document where the client verified the system conforming to their requirements was used as a means of verification.

3.4. CONCLUSION

In this chapter, the research paradigm and the steps in answering the research questions detailed in chapter one, were presented. The research paradigm and solution steps were also justified and explained in context of other alternatives. The solution steps followed in this dissertation were detailed in this chapter. Evidence from all the solution steps is collected and analysed. A conclusion is drawn reflecting on the research questions and answers uncovered during the research. Light is shed on new problems discovered during this research in the future work section.

4. RESULTS

4.1. INTRODUCTION

This chapter shows the results of the research. The results shall be presented in three parts. The first part is the ranking of the requirements of the SSEWDB. The second part is the software architecture design. The final part will be an experiment where the SSEWDB was deployed and tested. The main output of the research is the first part of the results, which is the ranking of the requirements of the SSEWDB. The architecture design and experiment answer research sub-questions as specified in section 1.2.2 and serve to corroborate the findings in the first section of the results, which is the ranked list. The validity of the ranked list is corroborated by the software architecture design and experiment through showing that the ranked list eases the software architecture design process, which is one of the main aims of this research.

4.2. RANKING REQUIREMENTS OF THE SSEWDB

In this section, requirements of the SSEWDB are ranked using a method called, 'Facilitating *Software Architecting by Ranking Requirements based on their Impact on the Architecture Process*' by Galster & Eberlein (2011). By ranking requirements of the SSEWDB, the most important quality attributes of the SSEWDB, being a real-time big data system, are uncovered and so will the implementation sequence in the architecture design.

4.2.1. Presentation

In this phase of the ranking method, the requirements that need to be ranked are listed by the stakeholders which are the CSIR. Researchers at the CSIR had already prepared a requirements document listing the functional and non-functional requirements of the SSEWDB. Table 9 shows the architectural requirements of the SSEWDB as provided by the CSIR. The least important requirement will have a rank of 1 while the most important requirement will have rank 5.

Table 9: Architectural requirements of the SSEWDB

Requirement Index	Requirement	Stakeholder rank
R_1	Handle x (10 000) concurrent users.	4
R_2	System must be up 99.5% of the time.	4
R_3	Allow not more that 3% failure rate of request due to architectural faults.	4
R_4	Maximum delay in fulfilling a request should not be more than 4 seconds.	5
R_5	Support low powered sensors	5
R_6	Accuracy at 5 decimal points	4
R_7	Provide security at all levels	3
R_8	Use only open source software	5
R_9	Use open standards to avoid vendor locking where tv-whitespace protocols are not specified.	4
R_{10}	Use 100% accuracy on sensed spectrum	5

4.2.2. Execution

The requirements listed above will now serve as input to the ranking method. Table 10 shows the requirements of the SSEWDB with their attributes and values. These attributes and values were decided upon by the software development team and the researcher during regular Joint Application Development (JAD) sessions. Although the researcher is the main architect, the team was brought in to remove subjectivity and more accurately capture how fulfilling each requirement affects the software development process based on the teams` previous experience. The details of the regular JAD sessions are beyond the scope of this research.

Table 10: The requirements of the SSEWDB with their attributes with values

Requirement index	Requirement qualities
R_1	Effort = "medium" Risk = "medium-high" Complexity = "medium" Early design = "No" Dependency = {} Volatility = "medium" Hardware constraints = "yes" Aspects addressed = {process execution engine, access and integration channels, persistence} Importance = "medium-high" Familiarity = "medium-high"
R_2	Effort = "medium" Risk = "high" Complexity = "low" Early design = "No" Dependency = "None" Volatility = "low" Hardware constraints = "yes" Aspects addressed = {deployments} Importance = "medium-high" Familiarity = "high"
R_3	Effort = "medium" Risk = "High" Complexity = "medium-high" Early design = "No" Dependency = {R_1 ,R_2 ,R_4} Volatility = "low" Hardware constraints = "yes" Aspects addressed = {deployment} Importance = "medium-high" Familiarity = "medium"

R_4	Effort = “medium-high” Risk = “High” Complexity = “low-medium” Early design = “no” Dependency = {} Volatility = “low” Hardware constraints = “yes” Aspects addressed = {processing, persistence, deployment} Importance = “high” Familiarity = “medium”
R_5	Effort = “low” Risk = “medium” Complexity = “low” Early design = “yes” Dependency = {} Volatility = “no” Hardware constraints = “yes” Aspects addressed = {deployment} Importance = “medium-high” Familiarity = “high”
R_6	Effort = “low” Risk = “high” Complexity = “low” Early design = “no” Dependency = {} Volatility = “medium-low” Hardware constraints = “no” Aspects addressed = {software implementation} Importance = “medium-high” Familiarity = “high”
R_7	Effort = “medium” Risk = “medium-low” Complexity = “medium-low”

	Early design = “no” Dependency = {} Volatility = “high” Hardware constraints = “no” Aspects addressed = {Software, hardware} Importance = “medium” Familiarity = “high”
R_8	Effort = “low” Risk = “low” Complexity = “low” Early design = “yes” Dependency = {} Volatility = “low” Hardware constraints = “no” Aspects addressed = {software} Importance = “high” Familiarity = “high”
R_9	Effort = “low” Risk = “high” Complexity = “medium-low” Early design = “yes” Dependency = {} Volatility = “low” Hardware constraints = “no” Aspects addressed = {software} Importance = “high” Familiarity = “medium-high”
R_{10}	Effort = “medium” Risk = “high” Complexity = “medium-high” Early design = “yes” Dependency = {} Volatility = “medium”

	Hardware constraints = “no” Aspects addressed = {deployment, software} Importance = “high” Familiarity = “medium-low”

4.2.3. Presentation

The presentation phase is where the outcome of the method is organised and presented to the stakeholders. The ranking index, ρ_n, will be calculated for each requirement, R_n. In calculating ρ_n, the impact of impeding forces, II_n, for R_n and the impact of Enabling factors, EI_n, for R_n will first have to be calculated. The result of sum of II_n and EI_n as shown in equation 3 will then be used to calculateρ_n.

Calculating the impact of impeding forces, II_n on ranking index, ρ_n for each requirement, R_n of the SSEWDB as explained in Equation 1.

Where *isIF*(<attribute>) is equal to 1 if the attribute value is the highest among all attribute values except familiarity which must be “low”. There can be more than one highest attribute value.

For R_1, II_1 = [((-1) x (4)) + (0) + (4)]

= -4+4

= 0

For R_2, II_2 = [((-1) x (5)) + 0 + 0 + 0]

= -5

For R_3, II_3 = [((-1) x (5)) + 0 + 0 + 0]

= -5

For R_4, II_4 = [((-1) x (5)) + 0 + 5 + 0]

= 0

For R_5, II_5 = [0+ 0 + 4 + 0]

= 4

For R_6, II_6 = [((-1) x (5)) + 0 + 0 + 0]

= -5

For R_7, II_7 = [0 + ((-1) x (5)) + 0 + 0]

= -5

For R_8, II_8 = [0+ 0 + 5+ 0]

= 5

For R_9, II_9 = [((-1) x (5)) + 0 + 5 + 0]

= 0

For R_{10}, II_{10} = [((-1) x (5)) + 0 + 5+ 0]

= 0

Calculating the impact of enabling factors, EI_n on ranking index, ρ_n for each requirement, R_n of the SSEWDB is done with Equation 2, where the parameters are such that;

For R_n, EI_n= [(*isIF*(effort) + *isIF*(*risk*)+*isIF*(*volatility*) +*isIF*(*importance*) +*isIF*(*familiarity*) +*isIF*(*complexity*))x 5]

Where *isIF*(<attribute>) is equal to 1 if all attribute values are "low" except familiarity which has to be "high".

For R_1, EI_1 = [(0 + 0 + 0 + 0 + 0 + 0) x 5]

= 0 x 5

= 0

For R_2, EI_2 = [(0 + 0 + 1 + 0 + 1 + 0) x 5]

= 2 x 5

= 10

For R_3, EI_3 = [(0 + 0 + 1 + 0 + 0 + 0) x 5]

= 1 x 5

= 5

For R_4, EI_4 = [(0 + 0 + 1 + 0 + 0 + 0) x 5]

= 1 x 5

= 5

For R_5, EI_5 = [(1 + 0 + 1 + 1 + 0 + 1) x 5]

= 4 x 5

= 20

For R_6, EI_6 = [(1 + 0 + 1 + 0 + 0 + 1) x 5]

= 3 x 5

= 15

For R_7, EI_7 = [(0 + 0 + 0 + 0 + 0 + 1) x 5]

= 1 x 5

= 5

For R_8, EI_8 = [(1 + 1 + 1 + 1+ 0 + 1) x 5]

= 5 x 5

= 25

For R_9, EI_9 = [(1 + 0 + 1 + 0 + 0 + 0) x 5]

= 2 x 5

= 10

For R_{10}, EI_{10} = [(0 + 0 + 0 + 0 + 0 + 0) x 5]

= 0 x 5

= 0

Calculating ranking in index, ρ_n for each requirement R_n of the SSEWDB is done using Equation 3 is done below.

For R_1, ρ_1 = (0 + 0)

= 0

For R_2, ρ_2 = [(-5 + 10) x 4]

= 5 x 4

= 20

For R_3, ρ_3 = (-5 + 5)

= 0

For R_4, ρ_4 = [(0 + 5) x 5]

= 5 x 5

= 25

For R_5, ρ_5 = [(4 + 20) x 4]

= 24 x 4

= 96

For R_6, ρ_6 = [(-5 + 15) x 4]

= 10 x 4

= 40

For R_7, ρ_7= [(-5 + 5)]

= 0

For R_8, ρ_8= [(5 + 25) x 4]

= 30 x 4

= 120

For R_9, ρ_9= [(0 + 10) x 4]

= 10 x 4

= 40

For R_{10}, ρ_{10}= [(0 + 0)]

= 0

From the above calculations, one can derive the ranked list of requirements. The ranked list of requirements of the SSEWDB based on ranking index, ρ_n for each requirement, R_n of the SSEWDB is given in table 11.

Table 11: The ranked list of requirements of the SSEWDB based on ranking index, ρn

Requirement Index	**Requirement**	**Rank Index ρ_n**
R_8	Use only open source software	120
R_5	Support low powered sensors	96
R_6	Accuracy at 5 decimal points	40
R_9	Use open standards to avoid vendor locking where tv-whitespace protocols are not specified.	40
R_4	Maximum delay in fulfilling a request should not be more than 4 seconds.	25

R_2	System must be up 99.5% of the time.	20
R_1	Handle x (10 000) concurrent users.	0
R_3	Allow not more that 3% failure rate of request due to architectural faults.	0
R_7	Provide security at all levels	0
R_{10}	Use 100% accuracy on sensed spectrum	0

The list in table 11 is subject to four constraints listed by Galster & Eberlein (2011). These constraints are namely:

- Early design
- Dependencies
- Hardware constraints
- Addressed aspects

Requirement R_{10} has a constraint of being a requisite requirement that is imposed on the software architecture design. Thus even-though it has a calculated rank index lower than most requirements on the list, it inherits a greater significance because of this constraint. After considering all these constraints, the ranked list of requirements of the SSEWDB changes and yields final results of ranked requirements of the SSEWDB shown in table 12.

Table 12: final results of ranked requirements with quality attributes of the SSEWDB

Rank	**Requirement Index**	**Requirement**	**Quality Attributes**	**Rank Index ρ_n**
1	R_{10}	Use 100% accuracy on sensed spectrum	Accuracy	0
2	R_8	Use only open source software	Cost	120
3	R_5	Support low powered sensors	Interoperability	96
4	R_6	Accuracy at 5 decimal points	Accuracy	40

5	R_9	Use open standards to avoid vendor locking where tv-whitespace protocols are not specified.	Interoperability	40
6	R_4	Maximum delay in fulfilling a request should not be more than 4 seconds.	Performance	25
7	R_2	System must be up 99.5% of the time.	Availability	20
8	R_1	Handle x (10 000) concurrent users.	Scalability	0
9	R_3	Allow not more that 3% failure rate of request due to architectural faults.	Reliability	0
10	R_7	Provide security at all levels	Security	0

4.3. SOFTWARE ARCHITECTURE DESIGN

4.3.1. Introduction

The software architecture section describes the process involved in designing a software architecture of the SSEWDB using the ranked requirements list. The Software Architecture design will be conveyed in two phases, namely:

- Early design decisions
- Architecture presentation

The early design decisions phase or section shows all software architecture design decisions that were taken through the scope of all architectural responsibilities as laid out by Solms (2012). This section illustrates how the ranked list requirements/quality attributes were addressed and the rationale behind every design decision. Architecture Presentation describes the architecture of the SSEWDB using an architecture Description language known as the Krunchten 4+1 Architectural views.

Before the software architecture design process is presented, it is important to have a high-level overview of the description of the SSEWDB. The SSEWDB is introduced together with its main purpose in section 2.4. Figure 10 shows the basic process description of the SSEWDB, in terms of its inputs, processing and output components.

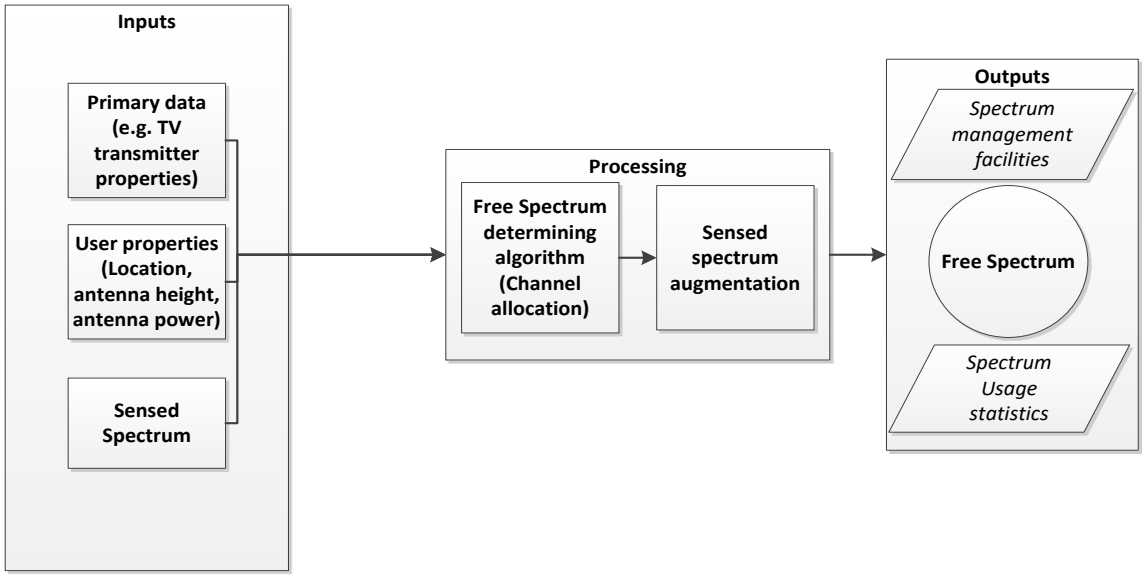


Figure 10:Basic process description of the SSEWDB

4.3.2. Early design decisions

In this section, design decisions are made and justified with respect to the ranked list in table 12. The requirements for the SSEWDB are addressed in descending order, from the most important to the least important, using the rank in table 12. The scope of architecture will be as stated by Solms (2012). The scope of architecture as stated in 2.3.4 covers mainly:

- Basic concepts and constraints
- Architectural Components
- Access and integration channels
- Architectural Strategies

It is important to note that while showcasing the design decisions that, how many decisions were there and exploring all other possibilities of design decision are beyond the scope of this research. The main point is to show that a ranked list enables making trade-offs easier.

Basic Concepts and Constraints

Ensure that architecture has low device compatibility.

Architectural Components

Table 13 shows the hosting platform choice matrix. Table 13 and all other tables in this section will show competing design options, the choice that was selected and the rationale behind picking one option over the other.

Table 13: Hosting platform choice matrix

Options	Choice	Reason
Centre for High Performance Computing (CHPC)	Yes	R_1, R_2, R_3 and R_4. • Ensure Maximum Performance through High Performance

		Computing. • Ensure high scalability and maximum availability through the use of a cloud Platform as a Service.
Local Hosting (Normal Servers at CSIR premises)	No	Will not satisfy maximum performance. Will not ensure maxim availability and will not scale as cloud hosting at CHPC.
Adding message queue to hosting environment to ensure R_1 and R_3 are achieved. (to ensure reliability)	No	Message queues will cause delays which would compromise R_4 which is higher in the ranked list. Scaling up using the CHPC platform is the preferred option.

Table 14 shows the deployment from sensors to SSEWDB choice matrix. Table 14 shows the technology type, or the integration pattern chosen between the SSEWDB and the spectrum sensors.

Table 14: Deployment from sensors to SSEWDB choice matrix

Options	**Choice**	**Reason**
Point-point	No	Will not support low powered sensors.
Gateway deployment	yes	R_5, ensuring interoperability with low powered sensors through using a gateway as a proxy.

Table 15 shows the SSEWDB execution environment choice matrix. This shows the options available for selecting the execution environment. The reason behind the execution environment ultimately chosen is given.

Table 15: SSEWDB execution environment choice matrix

Options	Choice	Reason
Apache Storm	No	There will be no unbounded data streams as spectrum is sensed and aggregated over time.
Java EE	Yes	R_{10} and R_6, spectrum is sensed over-time and aggregated to ensure maximum accuracy. Transactional Environment.

Table 16 shows the SSEWDB persistence technology choice matrix. Table 16 shows the type of technology used for storage of data of the SSEWDB. These were relational or the new NoSQL DBMS. The type of DBMS chosen, and the reason are also given.

Table 16: SSEWDB persistence technology choice matrix

Options	Choice	Reason
Relational	Yes	Data is structured and has relationships. Environment is transactional.
NoSQL	No	Data is semi-structured or Unstructured.

Access and Integration channels

Requirements R_9 and R_5 are considered when deciding on access and integration channels.

Table 17 shows application level protocols between sensors and SSEWDB:

Table 17: Application level protocols between sensors and SSEWDB

Options	Choice	Reason
Message Queuing Telemetry Transport (MQTT) (ISO/IEC JTC1 Information technology. 2016)	Yes	R_5, ensuring interoperability with low powered sensors as MQTT is; Light weight protocol, optimised for high latency so will perform better in rural parts of Africa with slow network connections,
Hypertext transfer protocol (HTTP) (Fielding et al 1999)	no	Heavier than MQTT and in rural parts of Africa with bad connections, will not perform as good as MQTT

Table 18 shows messaging protocols between whitespace devices and SSEWDB. The message protocols entail the type of message schema and type of message exchange protocol the SSEWDB will use. Below, this was mainly influenced by already existing standards.

Table 18: Messaging protocols between whitespace devices and SSEWDB

Options	Choice	Reason
Protocol to Access White-Space (PAWS) (IETF 2015) (over HTTP)	Yes	R_9 Ensure interoperability as PAWS is an accepted white space standard that is open.
XML and other forms of messaging (over HTTP)	No	Will not support interoperability as they are not defined standards in the white space community.

Table 19 shows messaging protocols between sensors and SSEWDB. As in above, the message protocols entail the type of message schema and type of message exchange protocol the SSEWDB will use. Below, this was mainly influenced by already existing standards.

Table 19: Messaging protocols between sensors and SSEWDB

Options	Choice	Reason
IEEE 802.22.3 (IEEE-STANDARDS ASSOCIATION 2017).	Yes	R_9 Ensure interoperability as this is a draft standard that will be accepted white space standard that is open.
IEEE 1900.6 (IEEE Standards Association 2016)	yes	R_9 Ensure interoperability as this is a draft standard that will be accepted white space standard that is open.

Architectural Strategies

Table 20 shows the Architectural Strategies employed on SSEWDB to ensure that it satisfies quality attributes specified in table 12.

Table 20: Architectural Strategies employed on SSEWDB

Problem	Quality attribute to address	Strategy	Implementation
To compute the parameters for primary data is processor intensive and causes delay	Performance (R_4)	Scale across time	Pre-process the data and store results
Ensure quick response time.	Performance (R_4)	Scale resource up	Centre for High Performance Computing (CHPC) Thread, connection and Object pooling at

			execution environment. Indexing the database layer.
Ensuring security at all levels	Security (R_7)	Authentication, encryption	Authentication when SSEWDB services are requested. Encryption at the device and sensor levels to the SSEWDB.

4.3.3. Architecture Presentation

This subsection depicts the architecture of the SSEWDB. The Architectural Description Language (ADL) used here is the Krunchten 4+1 architectural views. This ADL expresses software architecture in five views, which are the logical view, process view, physical view, development view and use case/scenario view. What each view represents or depicts will be explained in the brief introduction of the respective view, then diagrams will follow.

4.3.3.1. Logic view

This view is used to show application functionality. In this view the functionality of the SSEWDB will be shown and how the SSEWDB fulfils that functionality will be the main concern of the view. Figure 11 below shows the UML state diagram SSEWDB and shows how the SSEWDB fulfils its main functionality, which is to calculate and provide users with free spectrum, considering spectrum sensing in the calculation.

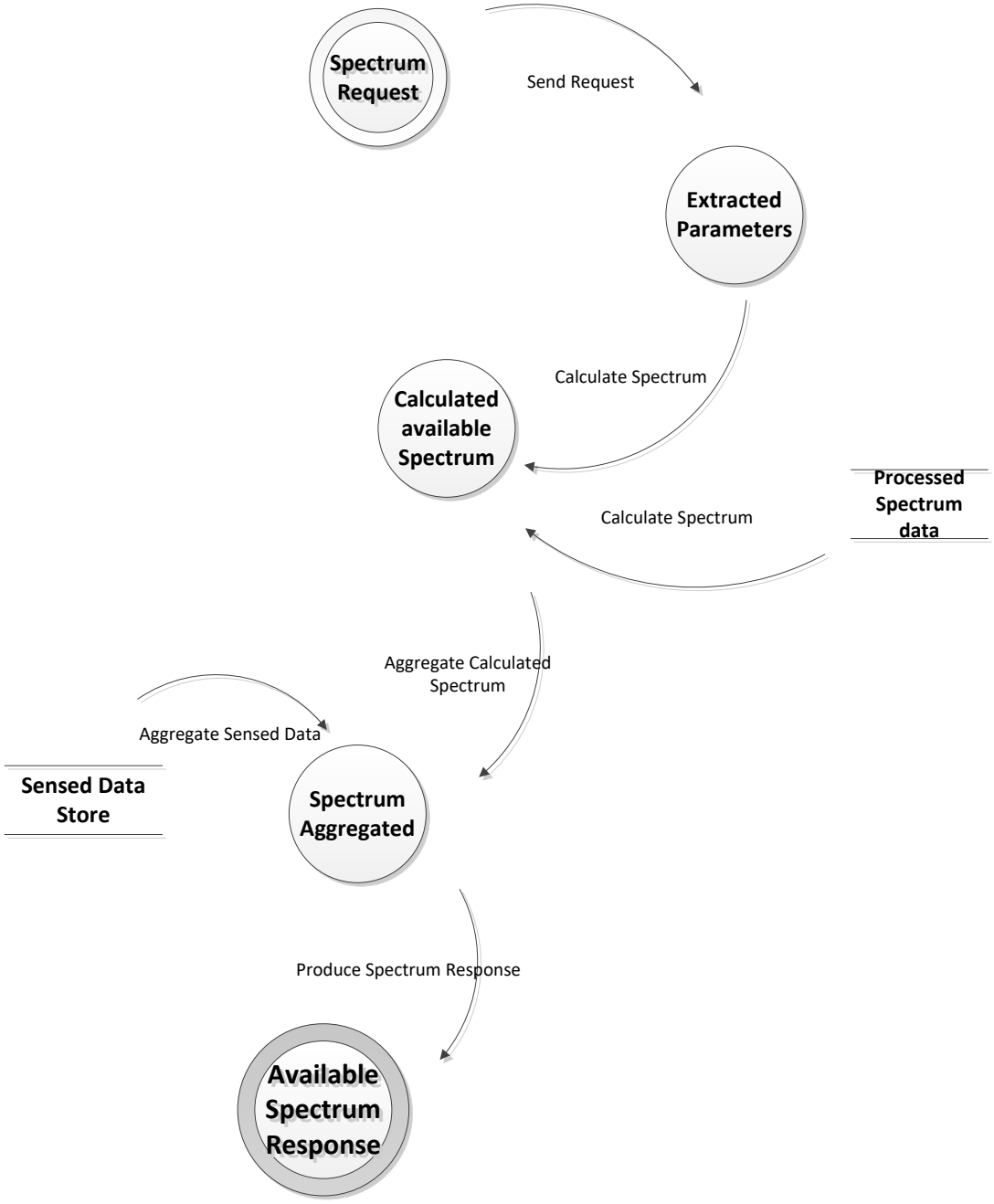


Figure 11: UML state diagram of the SSEWDB

Figure 11 above shows how the SSEWDB fulfils its main function of calculating and then providing free un-used spectrum to secondary users. Figure 11 shows how the SSEWDB

fulfils its main function by showing the transition in state of a request, up until it becomes a response to the end user. The end-user in this case can be a human user or a WSD.

The starting state is a request being made to the SSEWDB. An action of sending this request, which is made by the end user over the internet, prompts a change in state of the requested once it starts being processed by the SSEWDB. In this state, parameters are extracted, and these parameters will serve as input to the system. The extracted parameters are then transformed into calculated available spectrum by an action of calculating spectrum. Additional input to calculating spectrum is processed spectrum data. The processed spectrum data which is kept in a data store can be television station properties and terrain files.

An action of aggregate calculated spectrum changes the state of the calculated available spectrum to now being, spectrum aggregated. The most recently sensed spectrum data from the user's location is an additional input to aggregating calculated spectrum data. What the aggregated spectrum state means is that the calculated spectrum was cross-referenced with sensed data using some technique to increase accuracy of the SSEWDB. The techniques used to aggregate spectrum are however beyond the scope of this research. An action to produce spectrum response transforms the aggregated spectrum into available spectrum which is now useful to the querying user.

4.3.3.2. Physical view

This view shows the mapping of software into hardware and it also shows how the system is deployed in its environment. To achieve this, a UML deployment diagram is usually used. The separate system hardware entities are shown as blocks and these blocks serve as platforms to run different software that the system needs. There is also communication between these blocks and sometimes relationships. Figure 12 below shows the UML deployment diagram of the SSEWDB. The main blocks are the SSEWDB, which is where the system resides (CHPC server at the moment), Remote Map Server, Remote server, gateway platform, Prediction Environment, Low-powered sensor, Browser client and Device Client.

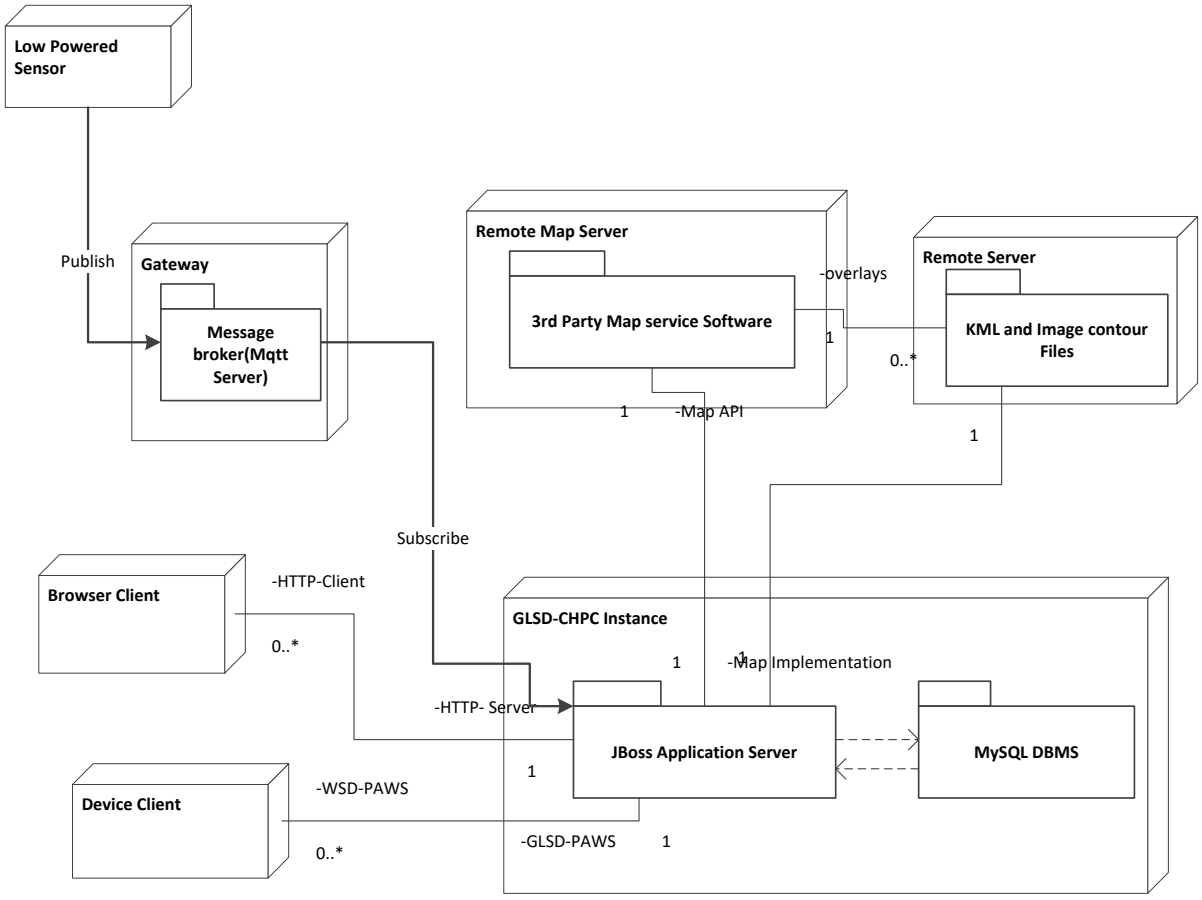


Figure 12: UML deployment diagram of the SSEWDB

The Deployment diagram in figure 12 shows the SSEWDB in a physical view as pointed out earlier. The SSEWDB has MySQL as the relational database management system. In this, all data relating to the transmitters, white space devices (WSDs) and all other related information is stored here. The database store (MySQL) is used by the Application housed in the Glass Fish Application Server. All software implementation is housed here, from the channel allocation algorithm, Map API implementation, PAWS protocol implementation to the general functionality of the SSEWDB, it is all here.

The SSEWDB communicates with the WSDs through PAWS, which is a protocol that uses JSON schemas over HTTP to establish communication. There is also the human client who communicates with the system through a browser. The system also communicates with a 3rd party Map service provider; this may be Google Maps, Microsoft Bing, open street maps

or any other. To establish communication with the maps service, map APIs are used. These APIs also communicate with Keyhole Mark-up Language (KML) files and image contour files on a remote server, to help 3rd party maps overlay that information on a map.

The prediction tools from the prediction environment generate all needed data that will be stored in MySQL, so there is a feed relationship between the two. The Image contours and the KML files also come as a result of the prediction process. And so, the KMLs and image file are then stored on a remote server where they can be accessed and processed both by the map API and 3rd Party Map service provider.

4.3.3.3. Process view

Figure 13 shows a sequence diagram of the SSEWDB. It describes how events will unfold as a spectrum is being queried from the system. Data is being read from the sensors to the server sensing interface. From the sensing interface, it is then stored in a database after being sensed over a period of time and being aggregated to ensure accuracy. In the database, it awaits being read and used by a device (WSD) through the PAWS protocol or a human user through a browser, when spectrum is queried. Queries for spectrum through a browser are almost the same as queries for spectrum through the PAWS protocol, the only difference being, device queries need more connections to the SSEWDB as they initialize and validate a device before it is served. This is to ensure security and it is also a PAWS protocol feature (IETF 2015). This is illustrated in figure 13 below.

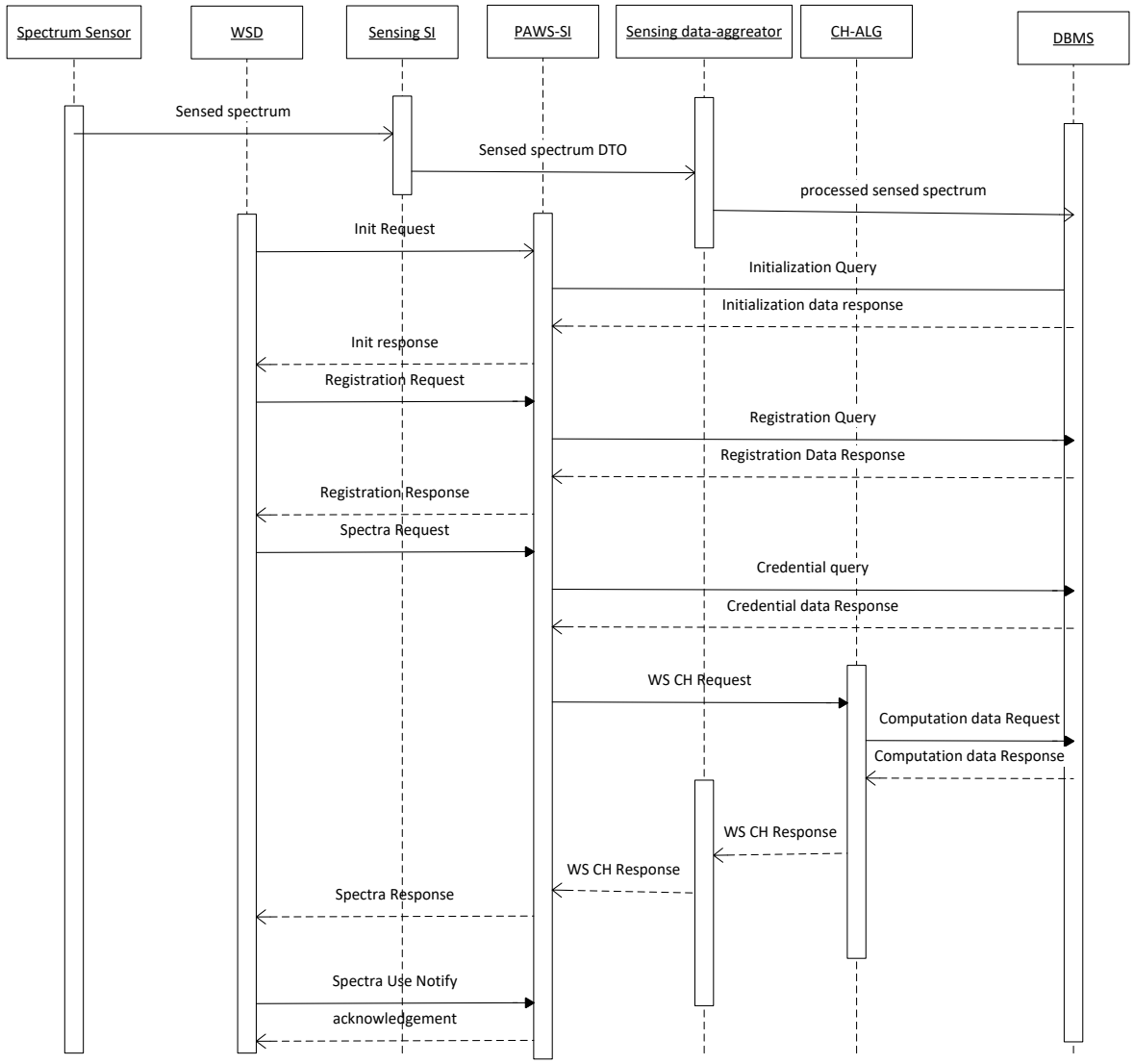


Figure 13: UML Sequence diagram of the SSEWDB

The WSD will start off by the initialization request, init Request. Then the PAWS-SI will pose an initialization query to the DBMS storing all device and user information. The DBMS will then provide a response to the PAWS-SI which will then respond to the user with init response. A registration query will be posed by the WSD and a similar set of events will occur with the init being replaced by registration.

After a registration response, the WSD will pose a spectra request. The PAWS-SI will still check the credentials of the device and verify that the WSD is indeed authorised to ask for

spectra. One may ask why even at this stage, verification is important. This is so because an attacker may jump some of these steps and try to use the system, so the security principle of mutual suspicion is applied here (no one process, or step can trust that the WSD is verified because it completed the previous processes).

The PAWS-SI will then request white space channels (WS CH Request) from the channel allocation algorithm (CH-ALG). The algorithm will query data needed to carry out its computation from the DBMS and then make a response to the PAWS-SI. The PAWS-SI will then make a spectra response to the WSD, which will then make a request for specific White space channels (WSCH). The PAWS-SI will then forward this request to the channel allocation algorithm which will then respond with specific channels.

The PAWS-SI will then make a specific channels allocation to the WSD. The WSD will notify the Geo-location spectrum database that it will use the spectra, and then through the PAWS-SI, the Geo-location spectrum database sends an acknowledgement. Thus, concluding the WSD communication with the Geo-location spectrum database.

4.3.3.4. Development View

This view depicts the SSEWDB software with its main modules. The internal working of the SSEWDB is represented in modular form and how those modules relate is also shown. The external systems interacting with the SSEWDB will also be shown as modules. Figure 14 below shows a UML component diagram of the SSEWDB. The components named using bolded font are the ones which will be explained in the following sections as they form the main components of the system.

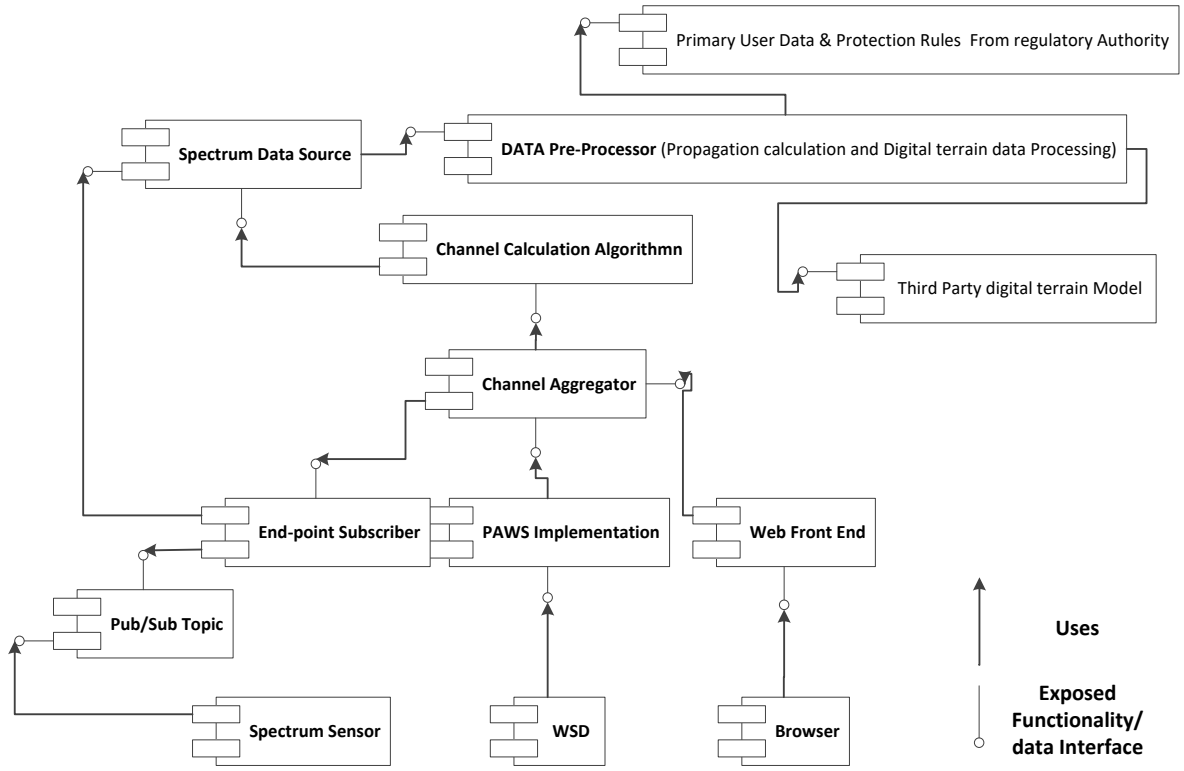


Figure 14: UML component diagram of the SSEWDB

The spectrum sensor is the first module that needs to have an interface to the system. The spectrum sensor interfaces with the system through a publish-subscribe integration pattern. The publish-subscribe integration module is housed in a gateway. There may be a number of gateways as the sensed space increases. Spectrum sensors will be publishing to a topic that the SSEWDB end-point module will be subscribing to. The end-point module will be pushing sensed data to a data-source for persistence.

The data-source also has data from the pre-processor module. The pre-processor module has data mainly in two forms. Primary user data and protection rules from regulatory authorities like Independent Communications Authority of South Africa (ICASA) and The Office of Communications (OFCOM) are the first form. The second type of data is third party digital terrain models that help determine the surface area where spectrum prediction is performed. This data is pre-processed, and the results stored in the data-source, and is used for spectrum calculation once a request is made.

A request can either be from a WSD and a human user through a browser. These two request channels have two distinct interfaces to the SSEWDB. The WSD queries the system using Protocol to Access whitespace IETF (2015). This interface follows a request-response integration pattern and interfaces with the SSEWDB through the Paws Implementation module. The browser, on the other hand, interfaces with the SSEWDB through the web-front end. An example of what this looks like can be seen at CSIR (2013).

Both front-end modules have an interface to the channel aggregator module. This module has an interface to the channel calculation module. The channel calculation module, using patented channel allocation algorithm (International Patent Office 2012) together with other calculation algorithms which are beyond the scope of the research, calculate free spectrum. This uses the request and the stored data from the data-source as input. The channel calculation module then forwards the results to the channel aggregator which then also uses the stored sensed data to cross reference the results of the calculation using some predefined scheme. The schemes used to cross reference calculated spectra with stored sensed data are beyond the scope of this research. The aggregator then forwards its results to either the web front-end or Paws Implementation depending on where the original request came from. The result is then given out.

4.3.3.5. Scenario

This view validates the architecture by showing all available scenarios of the system. This is done through depicting scenarios which are consistent with the other four views of the methodology. The way in which the architecture will be shown is through depicting the details of major scenarios through the use-case diagram. Figure 15 shows a scenario of a white space device successfully requesting spectra from the SSEWDB.

Figure 15 shows the UML use case diagram of the SSEWDB. This is done through a use case diagram other than a detailed scenario diagram for each use case. The several actors on the SSEWDB are depicted by stickmen which are a UML standard. The individual use cases are in the circles inside the SSEWDB system. There is a “uses” relation between the actors and

the respective use cases. The use cases may also use other independent use cases in the system, thus there is also a “uses” relationship among the use cases inside the system.

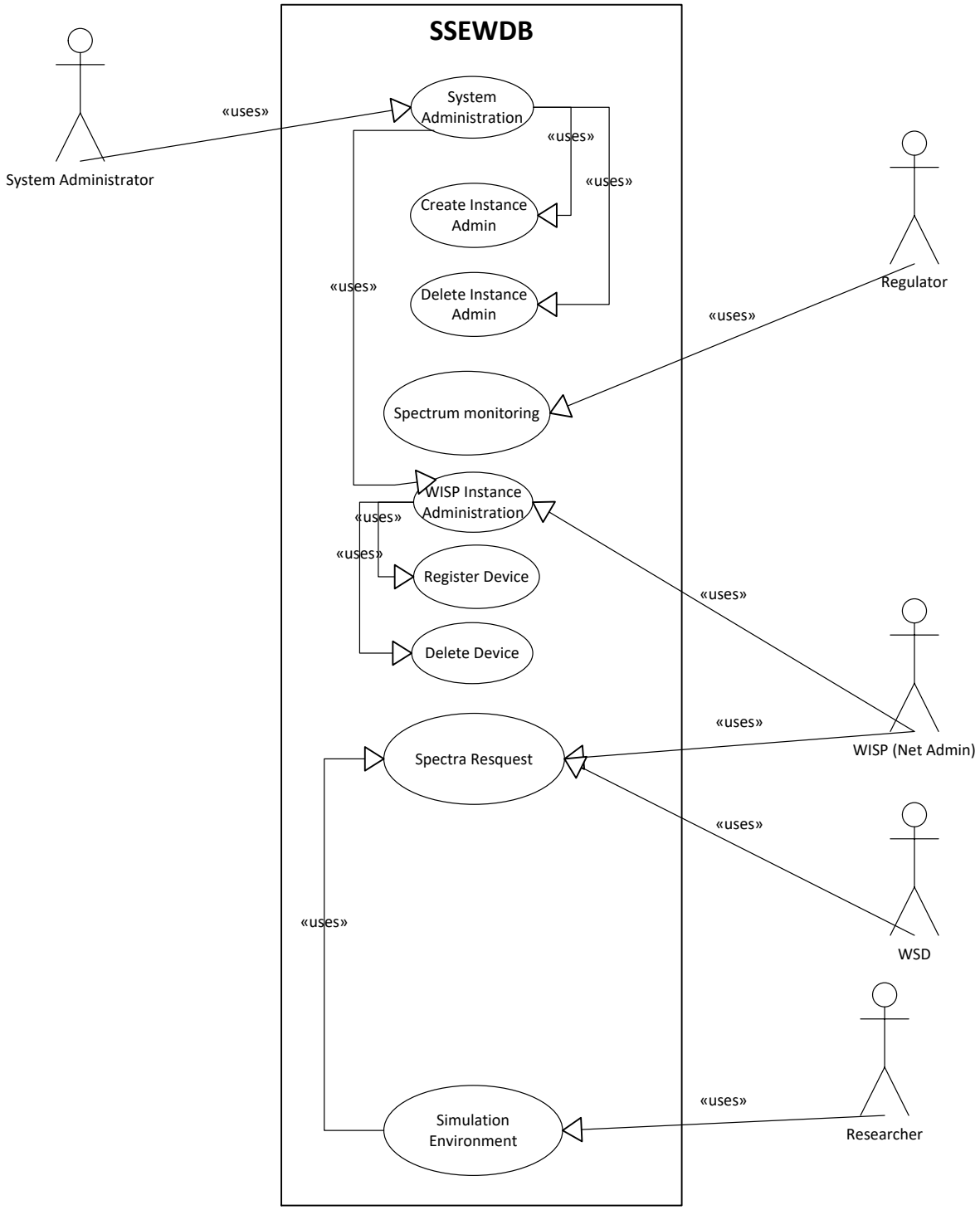


Figure 15: UML use case diagram of the SSEWDB

From the above use case diagram in figure 15, the system administrator uses the system for general system administration tasks. This also encompasses creating and deleting instance administration for WISPs. These instance-administrations for WISPs allow the WISPs to register and delete devices within a certain section or partition of usage they are provided. This means that WISPs cannot register and delete devices outside their instance, that is, other WISPs devices.

The WISP, together with the WSD, can also use the system by requesting spectra. This use case is also used by the simulation environment. The simulation environment is used by the research community when research on white spaces or generally on spectrum is made.

4.4 EXPERIMENT

4.4.1. Introduction

In this section, the system implementation of the software architecture design presented in section 4.3 is tested. The first part of this section explains the experiment set up and substantiates the rationale behind choosing the components chosen for the experiment. The second part describes the experiment. The last part concludes the experiment subsection.

4.4.2. Goal of Experiment

The overall goal of this experiment is to test whether the system met client requirements after being developed following the approach proposed in this research. In section 4.2 and section 4.3, it has already been shown how the system met client requirements, however, the most important requirements to the client require additional practical testing of the system.

The aim of the experiment was to test the system against the most important requirements of the client. Table 21 shows the client requirements tested during the experiment. These

requirements were extracted from the initial requirements of the SSEWDB provided by the client with their corresponding rankings from the client.

Table 21: Tested Client Requirements

Requirement Index	Requirement	Quality Attributes	Stakeholder Rank
R_4	Maximum delay in fulfilling a request should not be more than 4 seconds.	Performance	5
R_8	Use only open source software	Cost	5
R_{10}	Use 100% accuracy on sensed spectrum	Accuracy	5
R_1	Handle x (10 000) concurrent users.	Performance	4
R_3	Allow not more that 3% failure rate of request due to architectural faults.	Reliability	4

Only requirements that are practical and are most important to the client were chosen for testing. These are requirements given a score of 5 by the client. Requirement R_1, however had a rank of 4 but was chosen as it relates to R_4 because of performance. That is, it would be desirable to test if maximum delay is still less than 4 seconds per user when x10 000 concurrent requests are made. Requirement R_3 is practical and has a relation to performance of the system in that it would be desirable to test the reliability of the system having meet its performance requirements.

The other remaining requirements are not as practical and those are achieved in the design phases (sections 4.2 and 4.3). The remaining practical requirement is that of 99.5% availability, and this is covered by the Service Level Agreement (SLA) which the CHPC has with the CSIR on hosting the SSEWDB. This document is confidential, but during the course of the experiment, no downtime was experienced.

4.4.3. Significance

The significance of this experiment was to achieve the goal stated in 4.4.2 with consideration of early design decisions in 4.3.2 in table 12 where it was stated that scaling up is the preferred method for meeting R_4. That is, when the system is overwhelmed with requests at high speeds, it will drop requests that it is unable to process, this is the significance of message queues. Keeping requests waiting to be processed in a queue.

In order to handle large volumes of data (R1) and ensure minimum requests are discarded R3, Enterprise systems use message queues (Kim et al. 2016). Message queues, however, compromise performance (R_4) (Sadooghi 2015). Following the ranked software quality attributes in table 21, R4 has a higher rank that R1. So, to fulfil both software quality attributes, scaling up in order not to drop any unprocessed resources was chosen in the early design decision section.

4.4.4. Inputs to the Experiment

The experiment will have a spectrum request and hosting platform resources as inputs. The spectrum request will remain fixed as this follows a standard (IETF 2015). The hosting platform resources will be the control variables that will help us determine the optimum performance of the system following the priority of quality attributes imposed by the ranked list.

Table 22 below shows the input break-down of the spectrum request of the SSEWDB. The input break-down shows the details of the request made by which WSD to the SSEWDB.

Table 22: Input Break-down of Spectrum Request of the SSEWDB

Parameter name	**Parameter value**
Request name	Spectrum Request
Request Size	536 Bytes

Request type	HTTP – Request
Messaging Protocol	PAWS (IETF 2015)
Message Body	{"method":"spectrum.paws.getSpectrumSs","id": "req-id-01","params":{"antenna":{"heightType":"AGL","height":10.2}, "location":{"point":{"center":{"latitude":-24.0,"longitude":29.0}}}, "type":"AVAIL_SPECTRUM_REQ","version":"1.0", "deviceDesc":{"serialNumber":"SN504","modelId":"MN110","fccId":"FCC110"}}, "jsonrpc":"2.0"}

The message body from table 22 above shows all the parameters the SSEWDB needs in order to calculate the available spectrum the WSD making the request. How this happens from a software architecture perspective has been described in section 4.3 but how it happens from an application functionality perspective is beyond the scope of this research. It is important to note that this is the main functionality of the SSEWDB and most computationally intensive.

Table 23 shows the SSEWDB resources break down from the CHPC hosing platform. These are the control variables, as they will be increased to meet the requirements of the system. As they are increased, the performance of the system will be monitored, and the results of the request serviced without being dropped and average time per request will be monitored.

Table 23: SSEWDB Resources Break-Down from the CHPC Hosing Platform

Parameter Name	**Parameter value**
Resource Bundle	M1.Large
Virtual CPU	4 Virtual CPUs
CPU speed	
RAM	8 Gigabytes
Hard Disk space	40 Gigabytes
Resource Bundle	M1.XXLarge
Virtual CPUs	16 Virtual CPUs
CPU speed	

RAM	32 Gigabytes
Hard Disk space	40 Gigabytes

As stated in section 4.2, scaling up resources is the chosen method of meeting the client requirements. This is achieved through changing the resource bundles that are provided by the CHPC platform. Table 23, above shows two bundles of resources that were used in the experiment. Changing these resources and recording results when the experiment is running is the procedure of the experiment. That is, the above bundles are the control variables of the experiment.

4.4.5. Experiment Setup

Requirement R_8, which corresponds to reduction of cost by using only open source software for the development of the SSEWDB is addressed in section 4.2 and section 4.3. In this section, it was shown that all components chosen in the development of the SSEWDB were open source. To test the remaining quality attributes which were performance and accuracy, open source tools were also chosen. The open source tools chosen were; JUnit for testing accuracy and JMeter for performance testing. These are all open source java-based tools.

Figure 16 shows the SSEWDB test bed setup of four lap-tops each spawning 2500 threads. Each laptop was running the JMeter. JMeter was configured to query a public Internet-Protocol address (ip-address) of the CHPC platform where the SSEWDB is hosted. The queries went from the four laptops through the internet to the CHPC. The 2500 concurrent threads from each of the four laptops made the total concurrent threads to add up to 10 000. These threads represent WSDs in the real world.

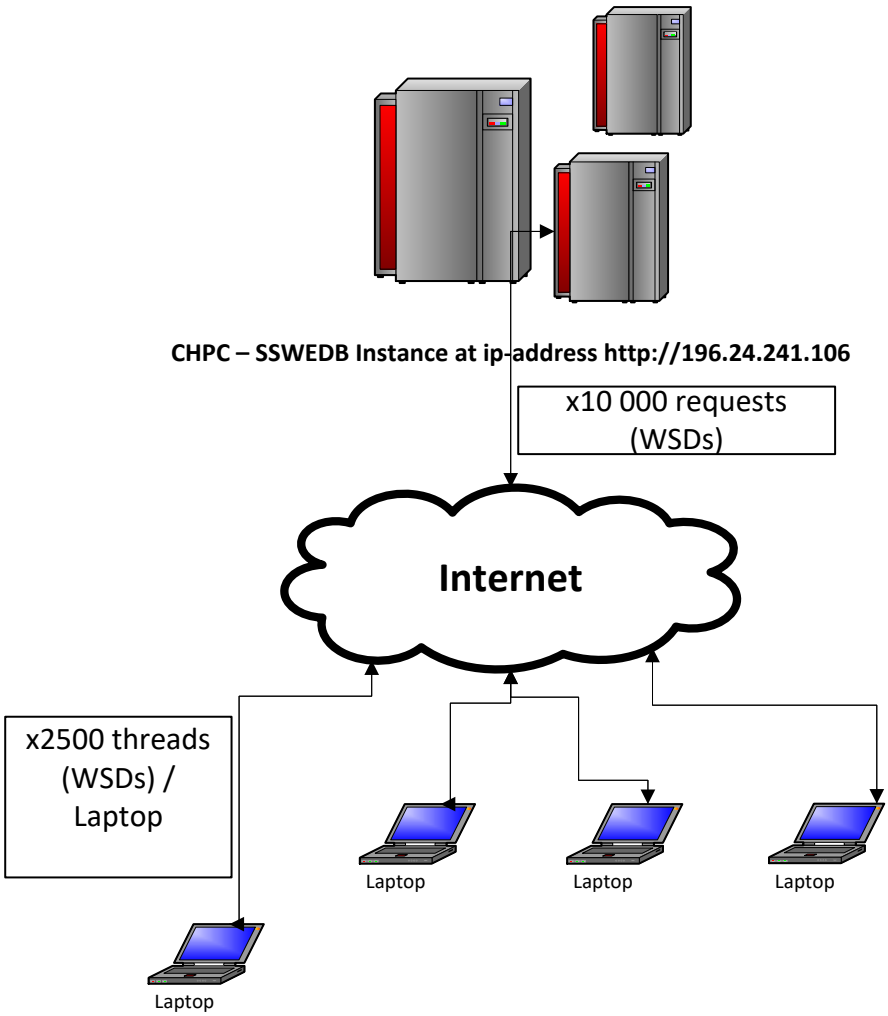


Figure 16: SSEWDB test bed setup

The laptops in figure 16 above spawned 2500 threads each, which would generate requests the desired amount of times. All these requests would travel to the CHPC where it was expected that the system would be able to handle the load of x10 000 whilst still performing at response rates of under 4 seconds each, as specified by R_4 and R_1.

To achieve the 2500 requests coming from each laptop, JMeter was installed on each laptop. Figure 17: JMeter thread group configuration shows how each laptop is configured to spawn 2500 threads which would each create a request. The total number of requests needed from all the threads is 2500 per laptop. Each thread represents a White Space Device in real life.

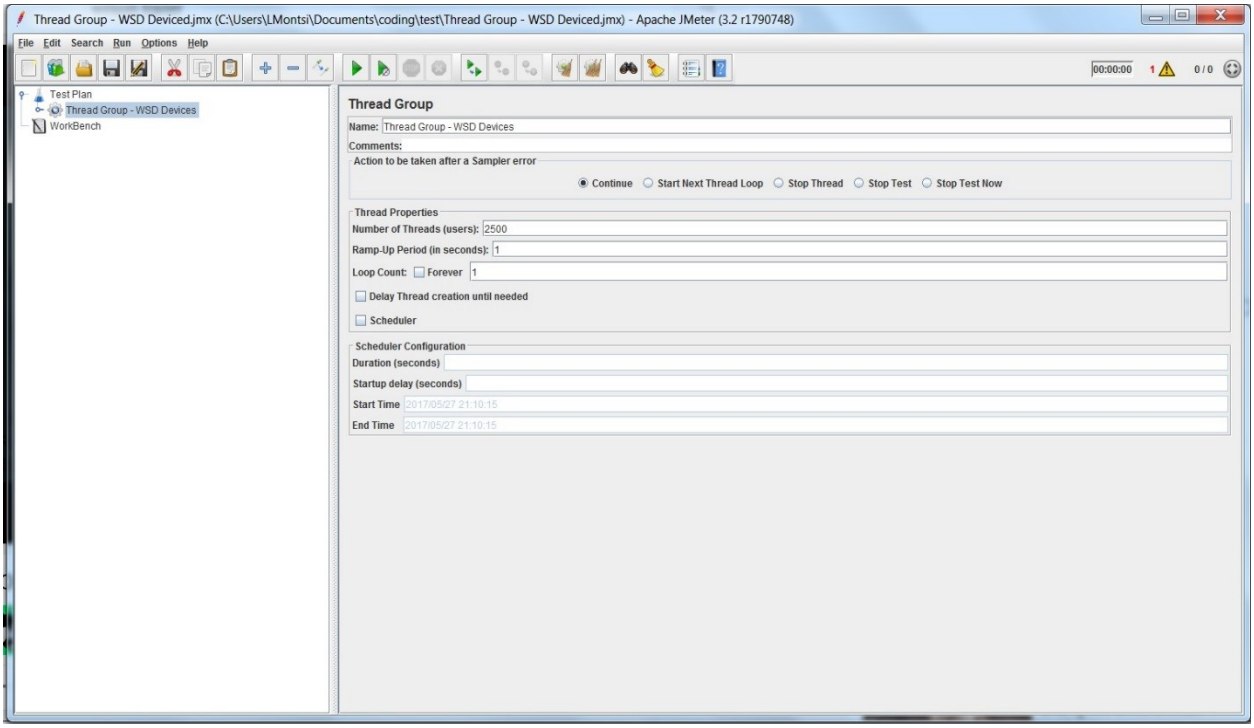


Figure 17: JMeter thread group configuration

Each of the 2500 threads will generate a single request. The request to the SSEWDB needs to also be configured. The variables to configure on JMeter for each request are:

- Protocol
- IP - address of destination server
- message end-point listener on the server
- and the message payload

Figure 18: The JMeter request configuration shows the configuration of each request a single thread was sending to the SSEWDB. The WSDs will be communicating with the SSEWDB using PAWS as specified in the design in section 4.3.2 in table 18 as this addresses R_9. The protocol which the devices use to communicate with the SSEWDB is HTTP as specified in the same table as PAWS.

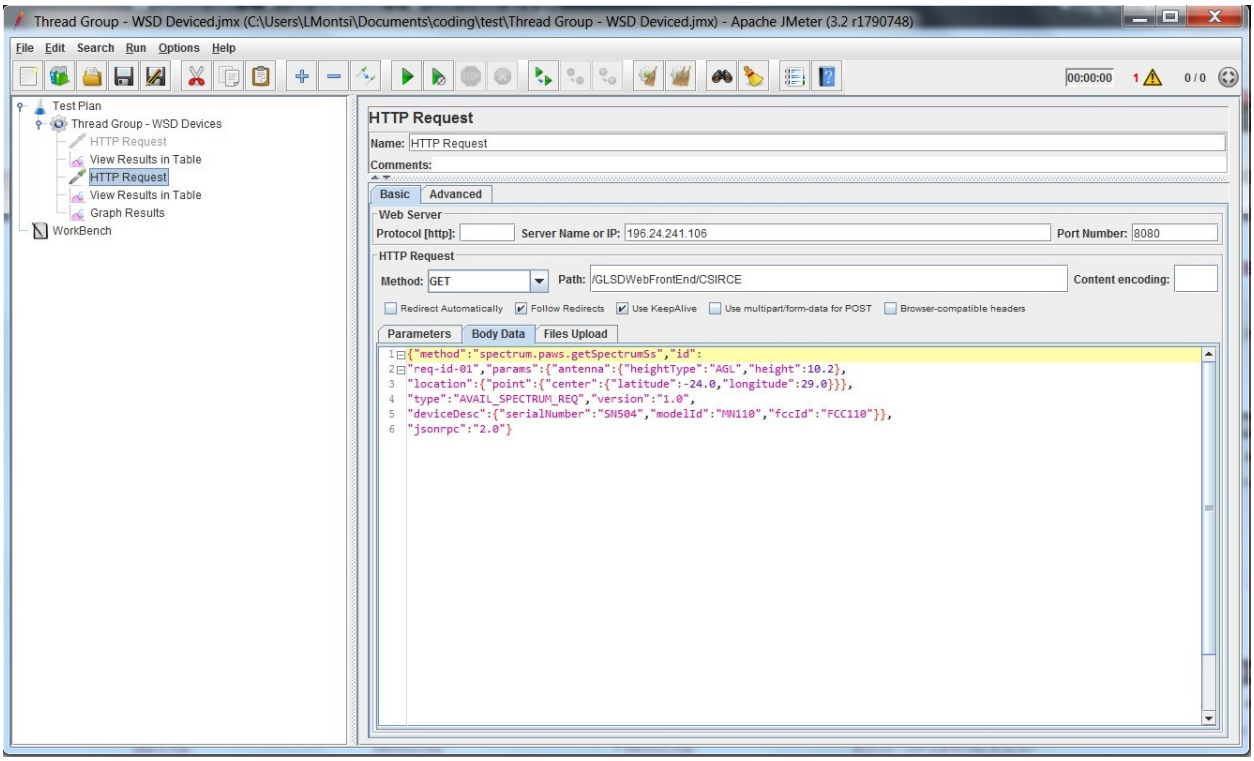


Figure 18: JMeter request configuration

The message body in figure 18 above is the spectrum request as specified by IETF (2015). The only enhancement done is on the method name "*spectrum.paws.getSpectrumSs*" as IETF (2015) has not yet catered for spectrum sensing enhanced queries. The IP-address to the SSEWDB is *196.24.241.106*. The SSEWDB is listening on port 8080 with the path named "*GLSDWebFrontEnd/CSIRCE*". The full Uniform Resource Locator (URL) to the SSEWDB is: http://196.24.241.106:8080/GLSDWebFrontEnd/CSIRCE.

4.4.6. Experiment Results

Running the Experiment with the control variable of resource bundle m1.Large as specified in the table 23, the first batch of results were obtained. Figure 19: JMeter table results for 2500 requests of the experiment running with resource bundle m1.large.

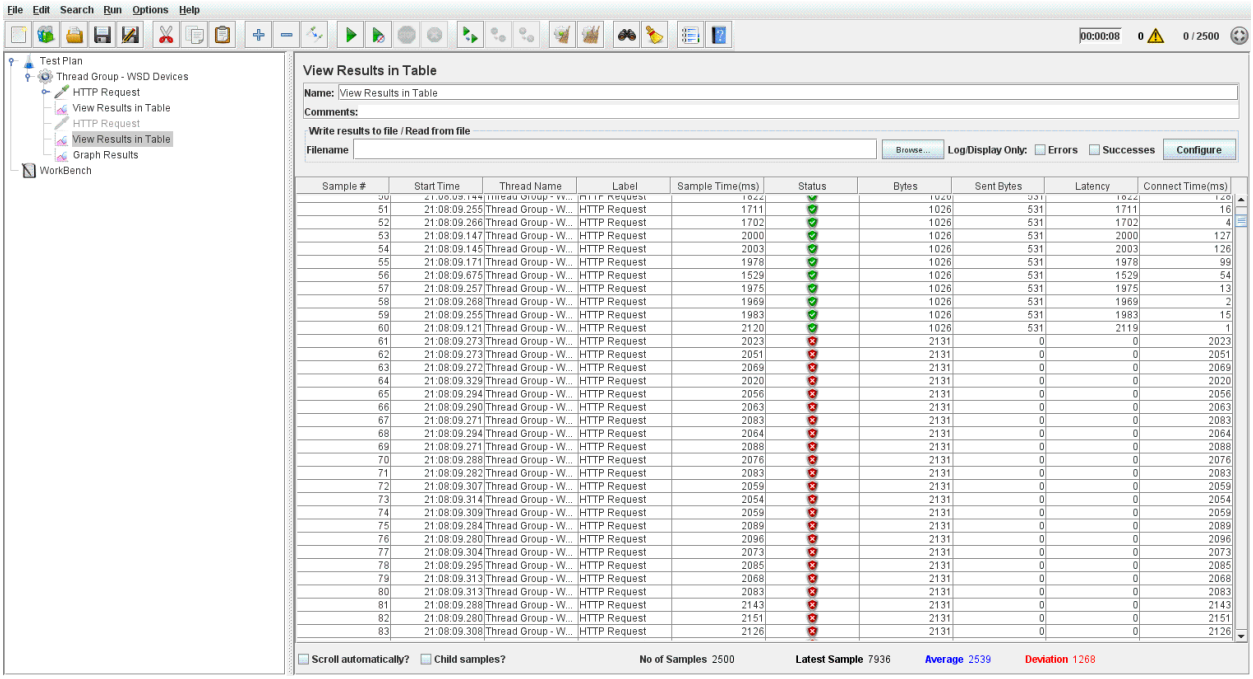


Figure 19: JMeter Table Results for 2500 requests

In figure 19 above, one of four laptops showing results of 2500 requests made to the SSEWDB. The red icons on the status column show failed requests made to the SSEWDB. Out of a total of 10 000 request, 7281 were successful. The total success percentage of 73% means that the system failed to fulfil requests and the failure percentage was 27% which is more than the 3% maximum required by requirement R_3. By the result in figure 19, the system then failed requirement R_3 andR_1. Results in figure 19 showed an average response rate of 2539 milliseconds (2.539 seconds), which is faster than the required 4 seconds, which means that R_4 passed.

The results in figure 19 above warranted a change in the control variables. The resource bundle was changed from m1.large with specification shown from table 23, to m1.XXLarge. Figure 20: CHPC cloud resource configurations show the change variables being modified. This is to increase the processing power and memory so that all requests are served as the system is no longer overwhelmed.

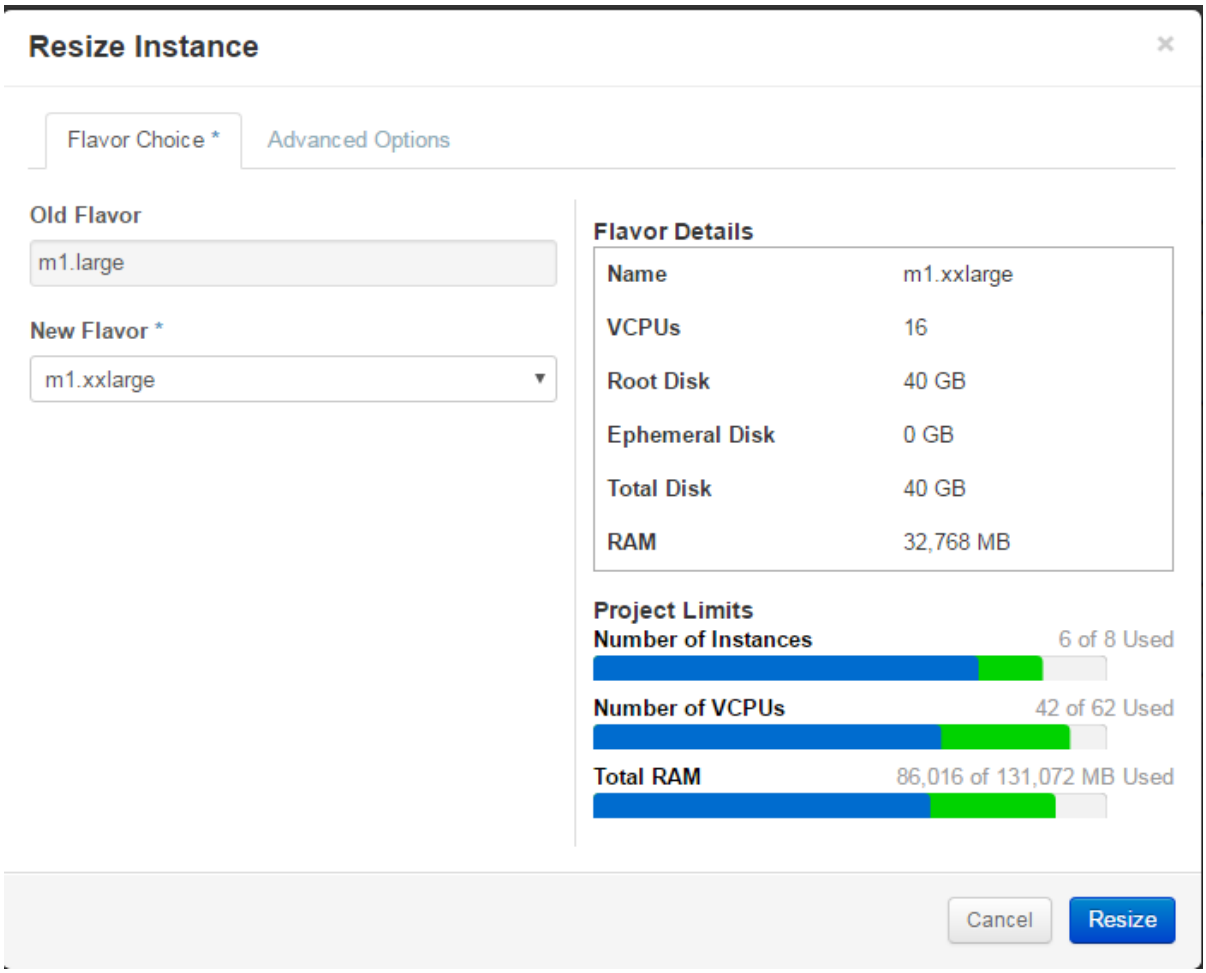


Figure 20: CHPC cloud resource configuration

The CHPC instance of the SSEWDB was now reconfigured to scale up in terms of increased resources as shown in figure 20 above and table 23, under m1xxlarge. The same test was run again. Figure 21: JMeter Table Results for 2500 requests after Resource Increase, shows the results of 2500 request coming from one out of four laptops which made a total of 10 000 requests to the SSEWDB.

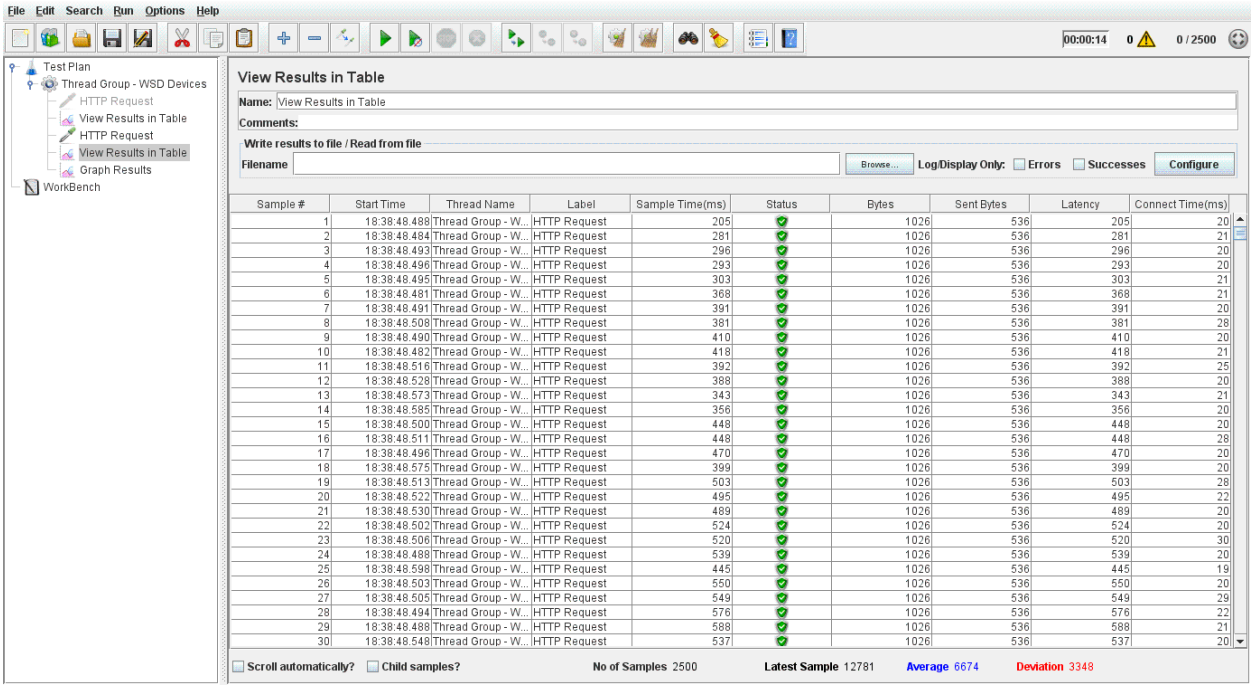


Figure 21: JMeter Table Results for 2500 requests after Resource Increase

Figure 21 shows the results of 2500 requests per laptop being sent to SSEWDB. Out of a total of 10 000 requests, none of the requests were dropped. This therefore implies a 100% success rate of requests which is an indication of R_3 andR_1 having passed. However, because all the requests were being served, this decreased the response time for all the requests. An average of 6650 (6.65 seconds) across all for nodes was recorded as the response time. This experiment was repeated 5 more times and the response time averaged 6672 milliseconds (6.67 seconds). This is higher than the time specified in R_4.

Figure 22: JMeter Graph results of 2500 requests after Resource Increase show how performance over time of the SSEWDB decreased in serving requests. The decrease in performance is attributed to the fact that now, all requests are getting served as opposed to previously, when the SSEWDB was being overwhelmed. So, response time for processing the request as opposed to just dropping them is a bit higher by an average of 6.67-2.539 = 4.131 seconds.

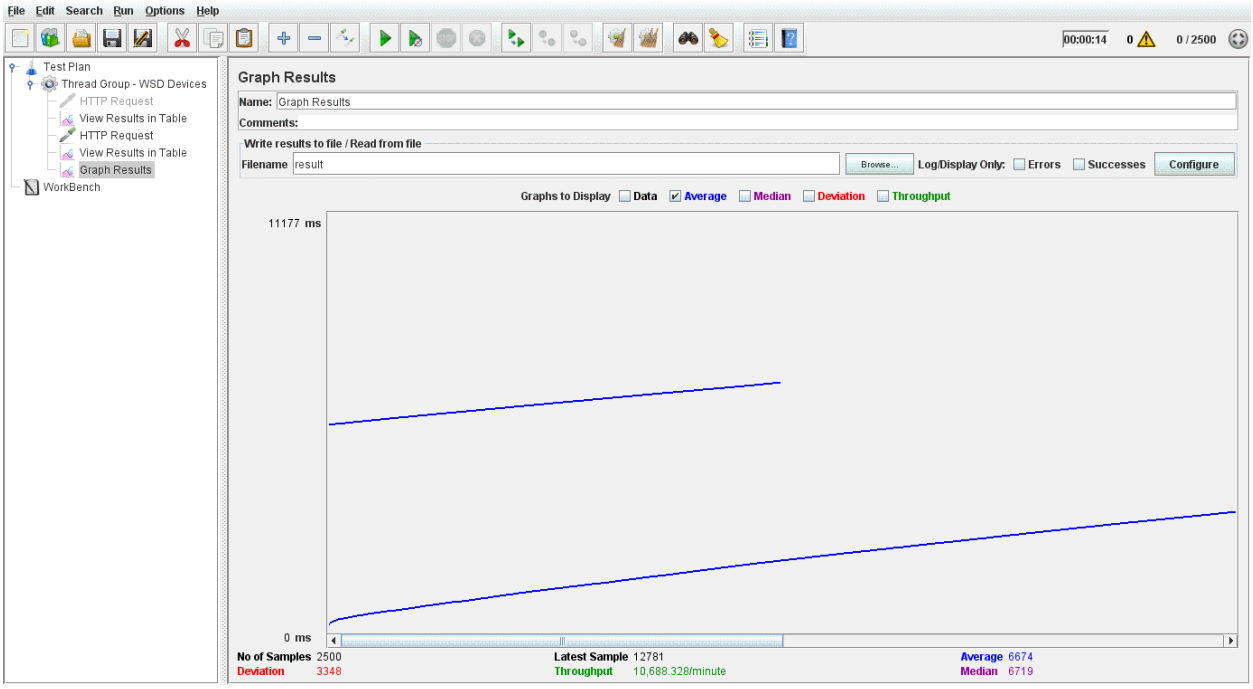


Figure 22: JMeter Graph results of 2500 requests after Resource Increase

From figure 22 above, it is evident that as the response time increases the number of requests the SSEWDB processes increases. This means that the SSEWDB starts off satisfying R_4 but as the limit of number of requests approaches 10 000, and possibly beyond, performance decreases.

4.5. CONCLUSION

In this section, the results of the dissertation have been presented. The processes of architecting a real-time big data system is presented in detail in this chapter. From ranking requirements using the chosen mathematical method. Afterwards, drawing early design decisions from those ranked requirements and then designing an architecture using a standard architecture description language. Subsequently, testing the implementation of that architecture using appropriate tools like JMeter for performance and reliability testing and Htop Linux tool for resource monitoring. The research sub question which this chapter mainly addresses is that which reads "*How can ranked quality requirements be used to develop a software architecture for real-time Big Data systems?*"

5. DISCUSSION

5.1. INTRODUCTION

In this chapter, the results presented in section 4, will be analysed and interpreted. A method to be used when developing software architectures for real-time Big Data systems is proposed based on the findings after discussing the results. The topics under discussion are outlined as they are in section 4. The ranked quality attribute results of the SSEWDB and other systems in literature are discussed first. The architecture design and experiment then follow suit. A method of architecting real-time Big Data systems which was used in this research is then formally proposed as a method that can be used for any other real-time Big Data system. Limitations of the study and a conclusion for this chapter follow last.

Quality attributes of real time Big Data systems from literature that were presented in section 2.3 are going to be discussed. Quality attributes from the case study are also going to be discussed. To validate the quality attributes in the case study, the architecture and how the client received the architecture will be presented and discussed. Furthermore, results from the experiment will also be discussed. Then, quality attributes from both the case study and literature will be compared and aggregated to come up with universal quality attributes for all real-time Big Data systems. A more detailed explanation is given in each subsection.

5.2. RANKED ATTRIBUTES RESULTS

In this section, quality attributes from literature are going to be discussed in three ways. The first approach is to present and discuss those that were presented in a ranked list from literature. The second way is to present the frequency (number of mentions) of each quality attribute in literature and derive a ranked list from that using the frequency of each quality attribute. The third way is to use the author's impression on the emphasis placed on a particular quality attribute in literature to rank these quality attributes.

5.2.1. Ranked Attributes from Literature

From the literature in section 2.3.2, the real-time processing of Big Data streams system was the only one presented with its quality attributes ranked explicitly. The quality attributes of this system are the only ones considered for explicit ranking. Table 24 shows the ranked quality attributes of the real-time Big Data systems from literature.

Table 24: Ranked Quality attributes of real-time Big Data systems from literature

Quality Attribute	Rank
Performance	1
Interoperability	2
Reliability	3
Maintainability	4
Security	5

From table 24 above, we can see that the most important quality attributes in terms ofa ranked list of requirements from literature are:

- performance
- interoperability
- Reliability
- Maintainability
- Security

Respectively in the order in which they are presented.

5.2.2 Frequency of Quality attributes in Literature

In this section of the discussion chapter, we look at the quality attributes of real-time Big Data systems based on frequency (number of mentions) of attributes in literature. The frequency is based on the architectural requirements mentioned in literature that fall under a particular quality attribute. Then these architectural requirements falling under a particular quality attribute are cumulatively summed to form the frequency of that

particular quality attribute. The cumulative sum of the frequencies is computed from the quality attributes reported in section 2.3 of the literature review. Table 25: quality attribute frequencies below show the quality attributes of real-time Big Data systems in literature ranked by the frequency of quality attributes.

Table 25: Quality Attributes frequencies from literature

Quality Attribute	Frequency
Performance	7
Interoperability	2
Reliability	3
Maintainability	2
Security	1
Extensibility	1
Accuracy	1

The frequencies of quality attributes in literature can further be presented on a more visually descriptive manner by means of a bar chart. Figure 23shows the literature quality attribute frequency bar chart.

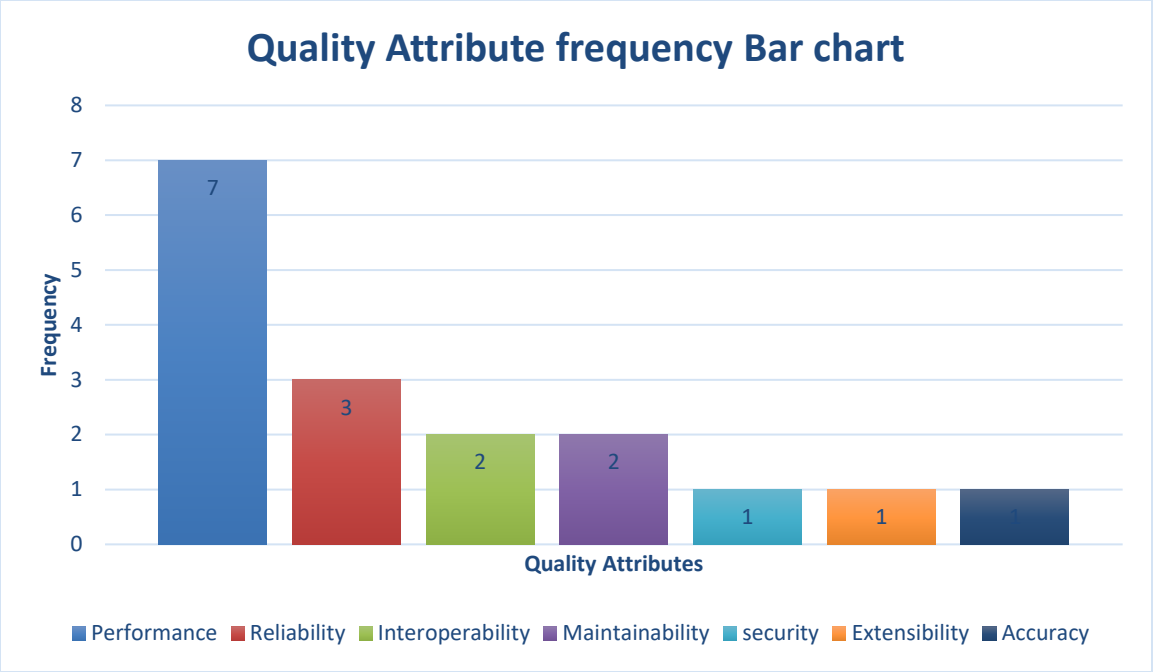


Figure 23: Quality Attribute Frequency Bar Chart

From the figure 23 above, it then follows that the ranked list of quality attributes most important to real time Big Data systems based on frequency of quality attribute in literature are as presented by table 26 below. In this section, the most important quality attribute is determined by the frequency. That is, increase in frequency and rank are directly proportional.

Table 26: Ranked Quality Attributes with Frequencies

Quality Attribute	Frequency	Rank
Performance	7	1
Reliability	3	2
Interoperability	2	3
Maintainability	2	3
Security	1	4
Extensibility	1	4
Accuracy	1	4

5.2.3. Authors Implied Rank of Quality Attributes

In this section, the author will give his implied rank of quality attributes in the literature. This means that even though the ranking was not explicit, the author provides his implicit ranking of quality attributes based on the emphasis which was placed on each quality attribute throughout the papers reviewed in the literature. That is, many quality attributes may have been reported but the central theme of quality attributes addressed was skewed unevenly across different quality attributes.

Table 27: Authors Implied Rank of Quality Attributes

Quality Attribute	Rank
Performance	1
Reliability	2
Interoperability	3
Accuracy	4
Security	5
Maintainability	6
Extensibility	6

5.2.4. Analysis of Ranked Software Quality Attribute Results

In this section, an analysis of ranked quality attribute results is performed. The ranked quality attributes from the case study are compared to the ranked attributes in literature. Figure 24 shows the comparison of ranked software quality attributes from literature and the case study. Software quality attributes in literature have been ranked using three methods in section 5.2.1 to 5.2.3.

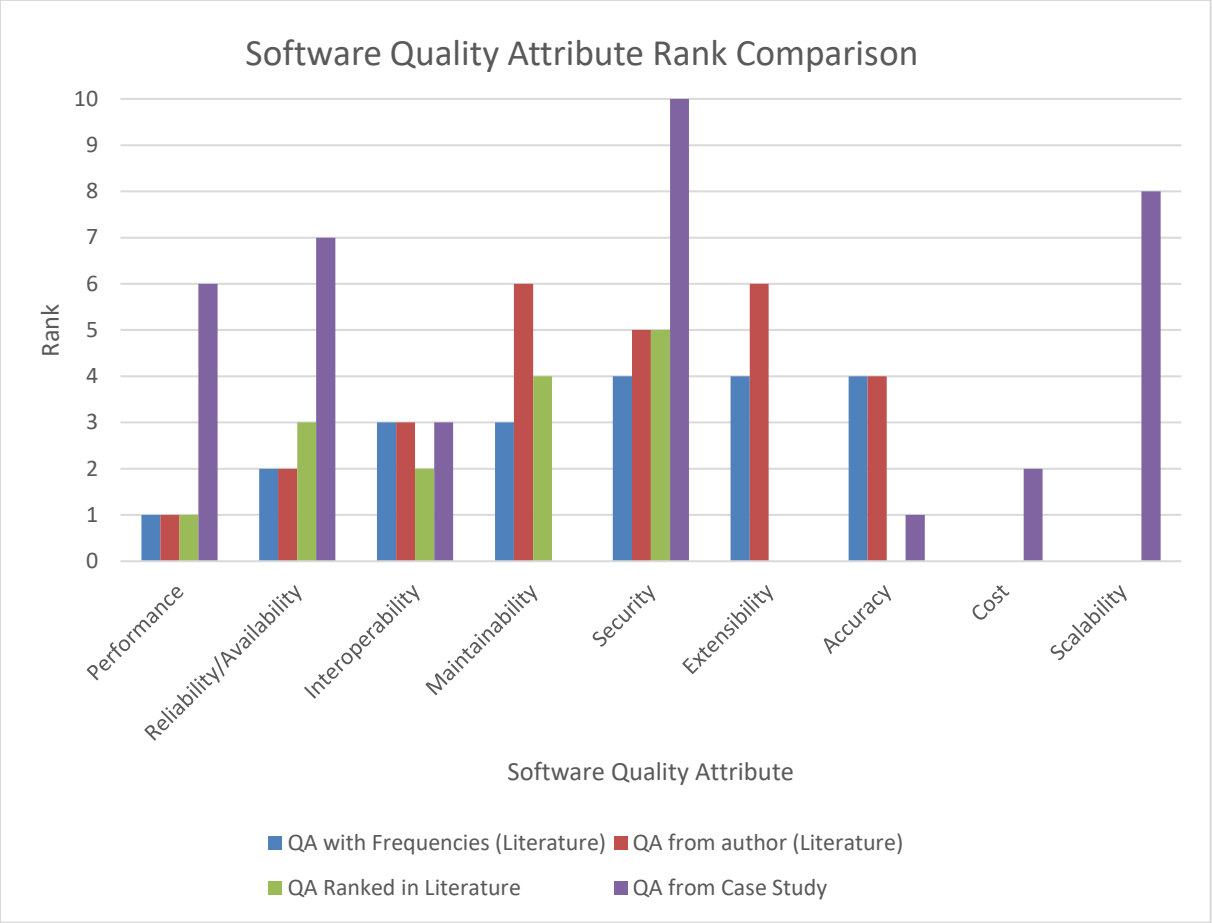


Figure 24: Software Quality Attribute Rank Comparison

From figure 24 above, software quality attributes with shorter bars indicate higher rank while longer bars indicate lower rank. Higher ranks imply higher priority. A higher priority means a quality attribute takes precedence over others in terms of implementation order or importance. From figure 24 above, it can be deduced that from literature, the most important quality attributes of real-time Big Data systems given in order of importance are:

- Performance
- Reliability/Availability
- Interoperability
- Accuracy
- Security
- Extensibility

There is a clear difference between this list and the results obtained from the case study. In the study, as shown by figure 24, accuracy is the highest, followed by cost and other software quality attributes respectively. This is mainly due to the ranking method used. The method, by definition, ranks quality attributes or software requirements based on their impact on the architecture process Galster & Eberlein (2011). This therefore means that the requirement that took higher priority over the ones presented as highest in literature would make the architecture process much smoother without compromising the overall architecture as the method entails. The second difference is caused by some requirements not being presented in the literature whilst being present in the case study and vice versa. Cost and scalability are such requirements that were present in the case study but not in literature.

5.2.5. Proposition of Ranked Software Quality Attributes Using Average Rank and Comparison

In section 5.2.4 it was shown that the method of ranking takes into consideration the impact a quality attribute will have on the process of ranking. This implies that there might be differences in the rank of what is in literature and what is in the case study. To harmonize these two approaches as a list or ranked attributes is still required for real time Big Data systems, this section proposes using two important traits of quality attributes to determine a list from which an inference can be drawn. In determination of the most important quality attributes to real-time Big Data systems, there are two factors to consider:

- Rank
- Commonality

The rank implies that the software quality attribute is considered most important and therefore more important than others. The average rank will be computed per quality attribute from all data sources. Commonality indicates that the quality attribute is common throughout all sampled real-time Big Data systems and methods of acquiring rank. Commonality counts the frequency a quality attribute appears in the three methods of

ranking from literature and in the case study. The total expected frequency is four as shown in figure 24. The frequency of appearance in results of ranking methods is influenced by whether the quality attribute was a requirement to the systems in question. Hence considering this when making an inference to the general population of real-time Big Data systems in important. Table 28: *Software Quality Attribute Average Rank and Commonality Comparison,* shows the proposed list of quality attributes that are most important to real-time Big Data systems.

Table 28: Software Quality Attribute Average Rank and Commonality Comparison

Rank	**Software Quality Attribute**	**Average Rank**	**Commonality (Frequency of appearance in results of ranking methods)**
1	Performance	2.25	4
2	Interoperability	2.75	4
3	Availability/Reliability	3.50	4
4	Security	6	4
5	Maintainability	4.33	3
6	Accuracy	3	2
7	Extensibility	5	2
8	Cost	2	1
9	Scalability	8	1

From table 28 above, taking commonality as the first filter and rank as the sub-filter, we now have the ranked quality attributes of real-time Big Data systems. From this list, an inference into the general population can be made. This list comes as a result of considering average rank and commonality of software quality attributes.

5.3. SOFTWARE ARCHITECTURE DESIGN

5.3.1. Early Design Decisions

The software architecture design process had two main phases. The first phase was early design decisions and the second phase was architecture presentation. Competing design decisions were listed at each point of the software architecture design. It is shown how software quality attributes with higher rank are chosen as opportunity cost over software quality attributes with lower rank. This process shows how the architecture process has been simplified from a previously complicated process where the criterion for choosing between competing software quality attributes was unclear.

5.3.2. Software Architecture Presentation

Software architecture presentation is derived from the early design decisions and presented in a formal software architecture description language. The software architecture description language chosen for this research was the Krunchten 4+1 architectural views. This helped depict the software architecture of the SSEWDB from different perspectives. This was especially useful because it carried elements of application functionality such as use cases as shown in section 4.3.3 under the scenario view. This software architecture description language and others can easily be used to present software architecture of any system as a form of expansion from the early design decisions.

5.4. EXPERIMENT

The experiment in section 4.4.5 revealed that the SSEWDB, which is a real-time Big Data system developed using a method of ranking quality requirements, conformed to user requirements. This was achieved through practically testing the client's most important quality requirements of the SSEWDB. The most important quality attributes of the SSEWDB from the client's perspective were:

- Scalability (R_1)

- Performance (R_4)
- Cost (R_8)
- Accuracy (R_{10})
- Reliability (R_3)

Table 29 below shows the outcome of the experiment in terms of the software architecture requirements that were met by the SSEWDB.

Table 29: Experiment Outcomes of the SSEWDB

Requirement Index	**Requirement**	**Quality Attributes**	**Outcome from experiment**
R_4	Maximum delay in fulfilling a request should not be more than 4 seconds.	Performance	Not met but within error margins of client
R_8	Use only open source software	Cost	met
R_{10}	Use 100% accuracy on sensed spectrum	Accuracy	met
R_1	Handle x (10 000) concurrent users.	Performance	met
R_3	Allow not more that 3% failure rate of request due to architectural faults.	Reliability	met

From table 29 above, all software architectural requirements were met except for performance. This was off by 2 seconds. When the client was shown this, they were happy and stated that this fell within acceptable error margins. As evidence of the client's satisfaction and of the SSEWDB having conformed to user requirements, the sign off document in appendix C was signed by the Chief Scientist of the CSIR, Dr. Fisseha Mekuria. Dr. Mekuria is the project leader of the SSEWDB and the main client of this study.

5.5. PROPOSITION OF A REAL-TIME BIG DATA SYSTEM ARCHITECTING METHOD

In section 4, a process was followed when developing the architecture from the requirements and testing the SSEWDB which is a system developed from these requirements. The results chapter in section 4 describe a software architecture development life cycle method from

- Requirements elicitation,
- Software architecture design
- And testing.

This method made the process of developing a software architecture easier through using known ranking methods to make the process of choosing competing quality attributes trivial.

Figure 25: *Proposed method for software architecture development of real-time Big Data systems* describes the steps to be taken when developing a real-time Big Data system.

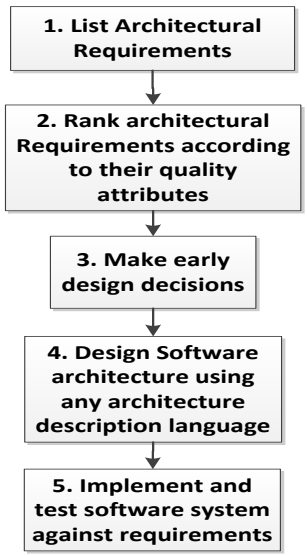


Figure 25: Proposed method for software architecture development of real-time Big Data systems

In step 1, Software architecture requirements are gathered from the client and listed. In step 2, architectural requirements are ranked according to the quality attributes they

address, using a known scientific ranking method. In this research, *'Facilitating Software Architecting by Ranking Requirements based on their Impact on the Architecture Process'* by Galster & Eberlein (2011) was chosen. Step 2 was done in section 4.2. In step 3, based on the ranked list of quality attributes in step 2, early design decisions are made. In step 4, after early design decisions, a software architecture is drafted using an architectural description language. Both step 3 and step 4 were done in section 4.3. In step 5, the developed software architecture is implemented and tested against the initial client requirements. Step 5 was done in section 4.4.

5.6. LIMITATIONS

The research had a small sample size of real-time Big Data systems reviewed in literature. This was mainly because the research was multifaceted in that, it was not just a literature survey, but it encompassed practical work and a case study and time was limited. Instead of just reviewing work done by other researchers, an instance of the same class of problems had to be solved and compared to other problems in that particular class through a case study and a literature review.

The employed ranking method was chosen because it was the least subjective among others examined in the research. However, there is still some element of subjectivity in that the software development team`s opinion on the impact each non-functional requirement will have on the software development process is still taken as one of the inputs. This cannot be avoided as human experience on how the implementation of certain requirements affect the software development process is key to ranking. There is no way of completely eliminating subjectivity, but the ranking methodology minimizes this by its design as presented in section 3.5.2. Comparing the results with other researchers` work also minimizes subjectivity.

5.7. CONCLUSION

In this chapter, the results from section 4 were discussed. Ranked quality attributes from the case study were compared with ranked quality attributes from literature and these were found to be consistent. The software architecture design and how it was derived from early design decisions and presented using the krunchten 4+1 architectural views was discussed. An experiment to test the implemented architecture was also discussed to examine whether the architecture requirements specified were met in practical terms. Because of the design science paradigm followed in this dissertation where deeper knowledge about the process of architecting real-time big data systems was uncovered as the dissertation work continued, in section 5.5, a real-time big data system architecting method is proposed. In section 5.6, the limitations were discussed.

6. CONCLUSION

6.1. INTRODUCTION

In this chapter, we conclude the work presented in this dissertation. This is done through four subsections. Firstly, the research questions are addressed. Secondly, objectives to the study are revisited. Next, the contributions of the research are stated. Lastly, future work of the study is presented.

6.2. REVISITING RESEARCH QUESTIONS

In this section of the conclusion, the research questions are addressed. Firstly, the research question at hand is stated and the corresponding findings in the research which address that question are provided. The research has one main research question and two sub-questions. These are:

Research question: What are the most important quality attributes for real-time Big Data systems?

This question was addressed from section 5.2.5 of the research, it was deduced that the general quality attributes that are most important to real-time Big Data systems are:

Quality Attribute	***Rank***
Performance	*1*
Interoperability	*2*
Reliability/Availability	*3*
Security	*4*
Maintainability	*5*

Sub-question 1: What are quality-attributes of a typical real-time Big Data system? *All the quality attributes of real-time Big Data systems that were uncovered in the review of*

literature and case study are typical quality attributes of a real-time Big Data system. These quality attributes include but are not limited to.

- *Performance*
- *Interoperability*
- *Reliability*
- *Maintainability*
- *Security*
- *Accuracy*
- *Extensibility*

These are typical quality attributes of any software system. What makes them unique to real-time Big Data systems is the order uncovered when answering the first research question.

Sub-question 2: How can ranked quality requirements/attributes be used to develop a software architecture for real time Big Data systems?

> *Ranked quality requirements influence early design decisions of developing a software architecture of real-time Big Data systems. The influence which these ranked quality requirements show is making obvious the opportunity cost between competing design decisions by using rank. The requirement with the higher rank becomes the obvious choice in software architecture development of real-time Big Data system. This was shown in section 4.3.2 of the research.*

6.3. REVISITING OBJECTIVES

In this section, objectives of the study are stated. Then all objectives are enumerated, and it is discussed whether they were achieved or not achieved. The objectives were:

1. Study and investigate existing and emerging ranking methods for software architecture quality attributes.

2. Design a software architecture for a real-time Big Data system.
3. Implement and evaluate a real-time Big Data system test bed.
4. Propose an approach for selecting the most important quality attributes for real-time Big Data systems

Objective 1 *was achieved in Section 2.2 Ranking Methodologies. A number of ranking methods were discussed and compared to the ranking method that was ultimately chosen for this research.*

Objective 2 *was achieved in section "4.3 Architecture Design". This section details the software architecture design of a real-time Big Data system. The design starts from early design decisions which were influenced by the ranked quality attributes of the SSEWDB from section "4.2 Ranked requirements of the SSEWDB". The early design decisions are then carried forth into section "4.3.3 Architecture presentation". In this section, the software architecture of the SSEWDB is presented by an architectural description language known as the Krunchten 4+1 architectural views.*

Objective 3 *was achieved throughout section 4 Results. The presentation of the test bed and evaluation is especially documented in the section 4.4 experiment.*

Objective 4 *is achieved in section "5.5. Proposition of Real-time Big Data System Architecting Method". It is shown that by ranking quality attributes, it eases the processes of having to pick between competing requirements or quality attributes in early design decisions. After early design decisions are made, they are then easily implemented and presented using an architecture description language of ones choosing. A depicted summary of this approach and all its steps is shown in "Figure 25: Proposed method for software architecture development of real-time Big Data systems".*

6.4. CONTRIBUTIONS

Contribution 1: An approach to the process of architecting real-time Big Data systems - The study made a significant contribution to the process of architecting real-time Big Data systems. This was done by providing a process model or an approach to help architect real

time Big Data systems. That is, the process used in architecting the SSEWDB (case study) can also be used in other real-time Big Data systems. This process helped to clearly prioritise quality attributes. Then it helped in showing how the prioritised list of quality attributes can be used in architecting a real-time Big Data system as shown in chapter 4. How to present software architecture once early decisions have been made is also shown. A summary of this approach is detailed in section *"5.5. Proposition of Real-time Big Data System Architecting Method".*

Contribution 2: Most important Quality attributes of real-time Big Data systems– this study has helped in uncovering the most important quality attributes of real-time Big Data systems. This helps architects to know what quality attributes to typically address when architecting real-time Big Data systems. This also lays the foundation to develop a software reference architecture for real-time Big Data systems. That is, as it shown in section 1.1.3, all sectors in ICT have a reference architecture to ease design and solution development. It is also shown that the reference architecture in all sectors across ICT were addressing typical quality requirements in those sectors. Furthermore, even in the case of real-time Big Data systems, such quality attributes have been uncovered because of this dissertation.

6.5. RESEARCH OUTPUTS

A paper titled, "A REAL-TIME BIG-DATA SYSTEM FOR SMART SPECTRUM SHARING: ENABLING AFFORDABLE BROADBAND IN 5G FOR UNDER-SERVED COMMUNITIES" accepted and presented to the IEEE Global Wireless Summit (GWS) 2017.

6.6. FUTURE WORK

A more extensive literature-oriented study into the quality attributes of real-time Big Data systems could be made to draw more insights into the quality attributes addressed by other researchers when designing software architectures for real-time Big Data systems. Such a study would not have any emphasis on a case study but be purely a literature review of work done by other researchers. A considerably large sample size would increase the

accuracy of the inference of quality attributes of the sample size would make to the general population of real-time Big Data systems.

A real-time Big Data software reference architecture development – Having uncovered the quality attributes of real-time Big Data systems, it lays the foundation of developing a software reference architecture to address these quality attributes. This would serve the intent of this dissertation, of having built the foundation for a real-time Big Data system software reference architecture to be developed.

A. REFERENCES

Alebrahim, A., Hatebur, D. and Heisel, M., 2011, December.A method to derive software architectures from quality requirements.In Software Engineering Conference (APSEC), 2011 18th Asia Pacific (pp. 322-330).IEEE.

Al-Naeem, T., Gorton, I., Babar, M.A., Rabhi, F. and Benatallah, B., 2005, May. A quality-driven systematic approach for architecting distributed software applications. In Proceedings of the 27th international conference on Software engineering (pp. 244-253).ACM.

AUTOSAR, G. (2007). Specification of RTE. Part of Release, 3.

Batyuk, A. and Voityshyn, V., 2016, August.Apache storm based on topology for real-time processing of streaming data from social networks. In Data Stream Mining & Processing (DSMP), IEEE First International Conference on (pp. 345-349). IEEE.

Bo, H., Hui, D., Dafang, W., &Guifan, Z. (2010, May).Basic concepts on AUTOSAR development.In Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on (Vol. 1, pp. 871-873).IEEE.

Bunge, M., 1998. Philosophy of science: from problem to theory (Vol. 1). Transaction Publishers.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., &Stal, M. (1996). Pattern-oriented software architecture, volume 1: A system of patterns.

Chen, Y. (2006). Multiple Criteria Decision Analysis: Classification Problems and Solutions. UWSpace.Available from:http://hdl.handle.net/10012/2892.[Accessed: 10/2/2017].

CSIR. (2013). whitepsace database. Available from: http://whitespaces.meraka.csir.co.za/. [Accessed: 19/2/2017].

Data, B., 2014.Principles and best practices of scalable realtime data systems. N. Marz J. Warren. Henning.

De Silva, L., Goonetillake, J., Wikramanayake, G., Ginige, A., Ginige, T., Vitiello, G., Sebillo, M., Di Giovanni, P., Tortora, G. and Tucci, M., 2014, September. Design science research based blended approach for usability driven requirements gathering and application development. In Usability and Accessibility Focused Requirements Engineering (UsARE), 2014 IEEE 2nd International Workshop on (pp. 17-24). IEEE.

Doshi, K. A., Zhong, T., Lu, Z., Tang, X., Lou, T., & Deng, G. (2013, October). Blending SQL and NewSQL approaches: reference architectures for enterprise big data challenges. In Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on (pp. 163-170). IEEE.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T., 1999. Hypertext transfer protocol--HTTP/1.1 (No. RFC 2616).

Fiorano. (2017). Services Oriented Architecture Implementation Frameworks. Available from: http://www.fiorano.com/whitepapers/soaesb/fiorano_soaif.php. [Accessed: 13/09/2017].

Galster, M. and Eberlein, A., 2011, April. Facilitating software architecting by ranking requirements based on their impact on the architecture process. In Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on (pp. 232-240). IEEE.Fant, J. S. (2011, May). Building domain specific software architectures from software architectural design patterns. In Software Engineering (ICSE), 2011 33rd International Conference on (pp. 1152-1154).IEEE.

Geerdink, B. (2013, March). A reference architecture for big data solutions introducing a model to perform predictive analytics using big data technology. In Information Science and Technology (ICIST), 2013 International Conference on (pp. 71-76). IEEE.

Google US. 2016. Google spectrum database. [ONLINE] Available at: https://www.google.com/get/spectrumdatabase/channel/. [Accessed 28 September 2016]

Greco, S., Ehrgott, M. and Figueira, J.R. eds., 2010. Trends in multiple criteria decision analysis (Vol. 142). Springer Science & Business Media.

Harrison, K. and Sahai, A., 2014, April.Allowing sensing as a supplement: An approach to the weakly-localized whitespace device problem.In Dynamic Spectrum Access Networks (DYSPAN), 2014 IEEE International Symposium on (pp. 113-124).IEEE.

Hurwitz, J., Nugent, A., Halper, F., & Kaufman, M. (2013). *Big data for dummies*.John Wiley & Sons.

IEEE STANDARDS ASSOCIATION . (2017). P802.22.3 - Standard for Spectrum Characterization and Occupancy Sensing. Available from: https://standards.ieee.org/develop/project/802.22.3.html. [Accessed: 27/2/2017].

IEEE Standards Association, IEEE 1900.6 Working Group on Spectrum Sensing Interfaces and Data Structures for Dynamic Spectrum Access and other Advanced Radio Communication Systems. 2016

IETF (2015) Protocol to Access White-Space (PAWS) Databases, Request for Comments (RFC) 7545, Internet Engineering Task Force (IETF).

International Patent Office. (2012). A method and system: back-tracking forward-looking (BTFL) dynamic channel allocation mechanism for white space radio networks. Available from: https://www.ipo.gov.uk/p-find-publication-getPDF.pdf?PatentNo=GB2522927&DocType=A&JournalNumber=6586. [Accessed: 25/2/2018].

ISO/IEC JTC 1 Information technology. (2016). ISO/IEC 20922:2016 Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1. Available from: https://www.iso.org/standard/69466.html. [Accessed: 27/2/2017].

Kaartinen, J., Palviainen, J. and Koskimies, K., 2007, April.A Pattern-Driven Process Model for Quality-Centered Software Architecture Design--A Case Study on Usability-CenteredDesign.In Software Engineering Conference, 2007.ASWEC 2007. 18th Australian (pp. 17-26). IEEE.

Kaisler, S., Armour, F., Espinosa, J. A., & Money, W. (2013, January). Big data: Issues and challenges moving forward. In System Sciences (HICSS), 2013 46th Hawaii International Conference on (pp. 995-1004).IEEE.

Katal, A., Wazid, M., &Goudar, R. H. (2013, August). Big data: Issues, challenges, tools and Good practices. In Contemporary Computing (IC3), 2013 Sixth International Conference on (pp. 404-409). IEEE.

Kim, K.H., Hwang, M.S., Jae, E.Y., Jun, S.H. and Kwon, M.C., 2016. A Study on Message Queue Safe Proper Time for AI Open API Fintech Architecture.

Klein, J., Buglak, R., Blockow, D., Wuttke, T. and Cooper, B., 2016, May. A reference architecture for big data systems in the national security domain. In Proceedings of the 2nd International Workshop on BIG Data Software Engineering (pp. 51-57). ACM.

Kum, D., Park, G. M., Lee, S., & Jung, W. (2008, October).AUTOSAR migration from existing automotive software.In Control, Automation and Systems, 2008.ICCAS 2008. International Conference on (pp. 558-562). IEEE.

Lloyd, P.T.L. and Galambos, G.M., 1999. Technical reference architectures. IBM Systems Journal, 38(1), pp.51-75.

Lysko, A.A., Masonta, M.T., Mofolo, M.R., Mfupe, L., Montsi, L., Johnson, D.L., Mekuria, F., Ngwenya, D.W., Ntlatlapa, N.S., Hart, A. and Harding, C., 2014, October. First large TV white spaces trial in South Africa: A brief overview. In Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2014 6th International Congress on (pp. 407-414). IEEE.

Masonta, M.T., Kola, L.M., Lysko, A.A., Pieterse, L. and Velempini, M., 2015, September. Network performance analysis of the Limpopo TV white space (TVWS) trial network. In AFRICON, 2015 (pp. 1-5). IEEE.

Mfupe, L., Montsi, L. and Mekuria, F., 2013. Spectrum database as a service (SDaaS) for broadband innovation and efficient spectrum utilization.

Mfupe, L., Montsi, L., Mzyece, M. and Mekuria, F., 2013, September. Enabling dynamic spectrum access through location aware spectrum databases. In AFRICON, 2013 (pp. 1-6). IEEE.

Mfupe, Luzango, FissehaMekuria, Litsietsi Montsi, and MjumoMzyece. "Geo-location White Space Spectrum Databases: Review of Models and Design of a Dynamic Spectrum Access Coexistence Planner and Manager." In White Space Communication, pp. 153-194. Springer International Publishing, 2015.

Microsoft. 2015. Microsoft whitespaces database. [ONLINE] Available at: http://whitespaces.microsoftspectrum.com/. [Accessed 28 September 2016]

Naumann, N. (2009). AUTOSAR Runtime Environment and Virtual Function Bus. Hasso-Plattner-Institut, Tech. Rep.

Nelson, M. L. (1999). A design pattern for autonomous vehicle software control architectures. In Computer Software and Applications Conference, 1999.COMPSAC'99.Proceedings. The Twenty-Third Annual International (pp. 172-177). IEEE.

NIST Big Data Public Working Group Reference Architecture Subgroup, "NIST Big Data Interoperability Framework: Volume 6: Reference Architecture." National Institute of Standards and Technology, Special Publication, 1500-6, 2015

Oracle Information Architecture: An Architect's Guide to Big Data, An Oracle White Paper in Enterprise Architecture August 2012. Retrieved from http://www.oracle.com/technetwork/topics/entarch/articles/oea-big-data-guide-1522052.pdf

Perry, D.E. and Wolf, A.L., 1992. Foundations for the study of software architecture. ACM SIGSOFT Software engineering notes, 17(4), pp.40-52.

PWG, N. B. D. (2014). Draft NIST Big Data Interoperability Framework. Reference Architecture.

Rathore, M.M., Ahmad, A., Paul, A. and Thikshaja, U.K., 2016, May. Exploiting real-time big data to empower smart transportation using big graphs.In Region 10 Symposium (TENSYMP), 2016 IEEE (pp. 135-139).IEEE.

Rathore, M.M.U., Paul, A., Ahmad, A., Chen, B.W., Huang, B. and Ji, W., 2015.Real-time big data analytical architecture for remote sensing application.IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 8(10), pp.4610-4621.

Sadooghi, I., Wang, K., Patel, D., Zhao, D., Li, T., Srivastava, S. and Raicu, I., 2015, December. Fabriq: Leveraging distributed hash tables towards distributed publish-subscribe message queues. In Big Data Computing (BDC), 2015 IEEE/ACM 2nd International Symposium on (pp. 11-20). IEEE. Vancouver

Solms, F. (2012, October). What is software architecture?.In Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference (pp. 363-373).ACM.

Triantaphyllou, E. and Mann, S.H., 1995. Using the analytic hierarchy process for decision making in engineering applications: some challenges. International Journal of Industrial Engineering: Applications and Practice, 2(1), pp.35-44.

Viana, P., & Sato, L. (2014, September). A Proposal for a Reference Architecture for Long-Term Archiving, Preservation, and Retrieval of Big Data. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on (pp. 622-629). IEEE.

von Alan, R. H., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS quarterly, 28(1), 75-105.

Weiss, M.B., Delaere, S. and Lehr, W.H., 2010. Sensing as a service: An exploration into practical implementations of DSA.Institute of Electrical and Electronics Engineers.

Xhafa, F., Naranjo, V. and Caballé, S., 2015, March. Processing and analytics of big data streams with yahoo! s4. In Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on (pp. 263-270). IEEE.

Zhao, X. and Zou, Y., 2010, July.A Business Process Driven Approach for Generating Software Architecture.In Quality Software (QSIC), 2010 10th International Conference on (pp. 180-189).IEEE.

Zhaoxu, S. and Min, H., 2010, June. Multi-criteria decision making based on PROMETHEE method. In Computing, Control and Industrial Engineering (CCIE), 2010 International Conference on (Vol. 1, pp. 416-418).IEEE.

B. APPENDICES

B.1. APPENDIX A: LETTER OF CONSENT FROM THE CSIR

Dr. Fisseha Mekuria

PO Box 395 Pretoria 0001 South Africa
Tel: +27 12 841 4604
Fax: +27 12 841 4720
Email:fmekuria@csir.co.za

10 October 2016

To Prof. Barry Dwolatzky
Department of Electrical Engineering
Faculty of Engineering and Built Environment
University of the Witwatersrand (Wits)
Johannesburg.

Mr. Litsietsi George Montsi, Student Number: 767034, MSc in Engineering.

This is to confirm that Mr. Montsi who is an employee of the CSIR Meraka Institute and also a registered postgraduate student at Wits is allowed to use and publish datasets, and results, from the Geo-location spectrum database (GLSD) project for the sole purpose of his MSc research work, titled "A software architecture for a real-time big data system: A case study of a spectrum sensing enabled whitespace database".

Please don't hesitate to contact me for any aspect of this letter.

Yours faithfully,

Dr. Fisseha Mekuria (Chief Researcher and GLSD Project Leader)

Council for Scientific & Industrial Research, CSIR,
Wireless Computing & Networking, CSIR Meraka ICT Institute.
Adj. Prof. University of Johannesburg, Dept. of EEES,

Mering Naude Road,Scientia Campus,Building 43,
P.O.Box 395,Pretoria 0001,South Africa.
Tel: +27 12 8414606,
Fax: +27 12 8414720
E-mail: fmekuria@csir.co.za
www.csir.co.za/meraka.

B.2. APPENDIX B: ETHICS CLEARANCE LETTER

Human Research Ethics Committee (Non-Medical)

Research Office Secretariat: Senate House Room SH 10004, 10th floor.
Tel No. 011 717 1408

15 February 2017

To whom it may concern,

I wish to confirm that Mr Litsietsi Montsi (767034) does not require ethical clearance for his project.

Please feel free to contact me should further information in this regard be required.

UNIVERSITY OF THE WITWATERSRAND
OFFICE OF THE DEPUTY VICE CHANCELLOR RESEARCH
15 FEB 2017
RESEARCH OFFICE
HREC (Non-Medical), AESC & AECC
RECEIVED
JOHANNESBURG

Yours Sincerely

Lucille Mooragan
Non-medical and Animal Ethics Committee

E-Mail: Lucille.Mooragan@wits.ac.za

B.3. APPENDIX C:USER ACCEPTANCE SIGN-OFF DOCUMENT

CSIR

our future through science

User Acceptance Sign Off Document

Software architecture design of a spectrum sensing enabled real-time whitespace database system

Executive summary

The software architecture design of the spectrum sensing enabled real-time whitespace database system aims to provide the architecture specification which will then be used by the CSIR to implement the aforementioned system. Whitespace databases are used worldwide as broadband enablers through making dynamic spectrum access possible. The CSIR aimed to augment the predictive models used by such systems by sensing in real-time, spectrum data and giving refined output. These posed an architectural challenge of potentially having to service a large number customers, taking information from potentially, large number of sensing devices then computing large amounts of data and giving results in real-time. Hence the need of a software Architecture design that addressed such a problem.

The Software Architecture design covers all the architectural responsibilities being, access and integration strategies, execution environment and persistence. The design addresses the real-time nature of sensed spectrum data together with the need to service potentially large number of customers, giving real-time responses to requests.

Project team Members

Senior Engineer/MSc Student: Litsietsi Montsi Competency Area: Networks and Media

Submitted on: Reviewed on:

Sign off Section

This Software Architecture design addresses all requirements and conforms to agreed specifications.

Signature of Executive Sponsors

Dr Fisseha Mekuria, Chief Researcher, CSIR

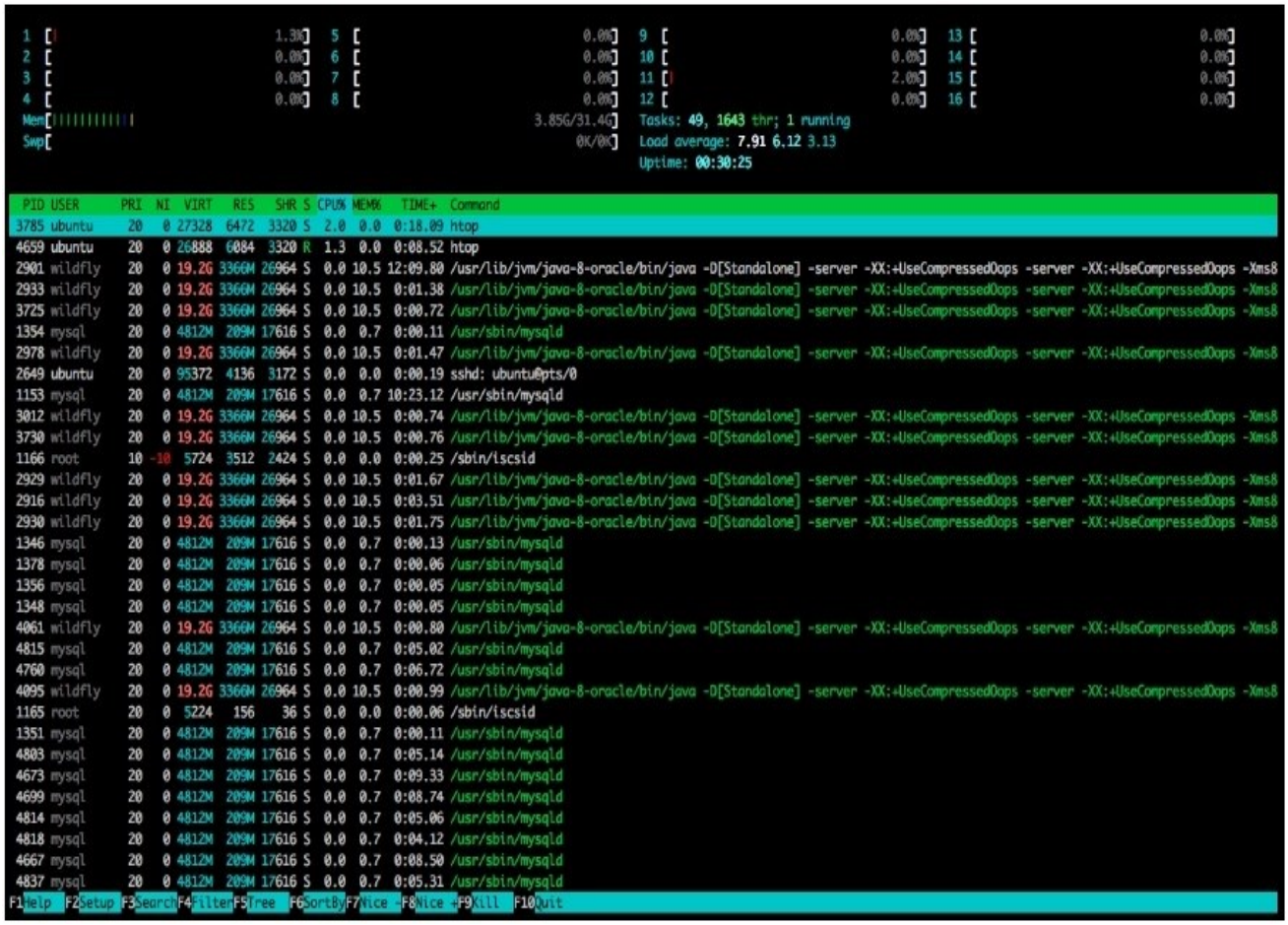
Name/Position Date 2017-05-24 Contact 0714596676
fmekuria@csir.co.za

MR. MARTIN MHLANGA, SENIOR PROJECT MANAGER, CSIR

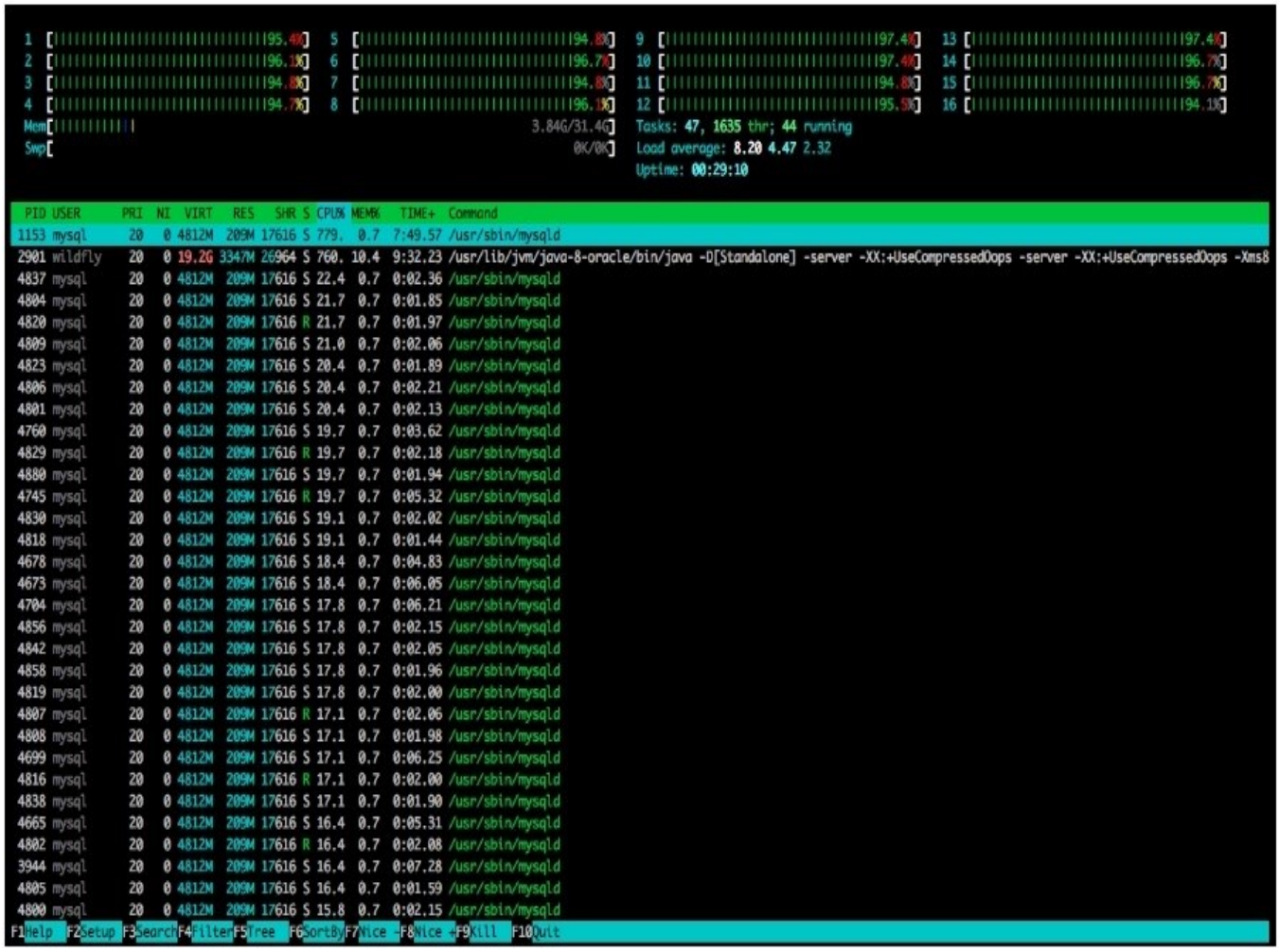
Name/Position Date 2017-05-25 Contact 0720881815
MMHLANGA@CSIR.CO.ZA

meraka INSTITUTE
PO Box 395, Pretoria 0001

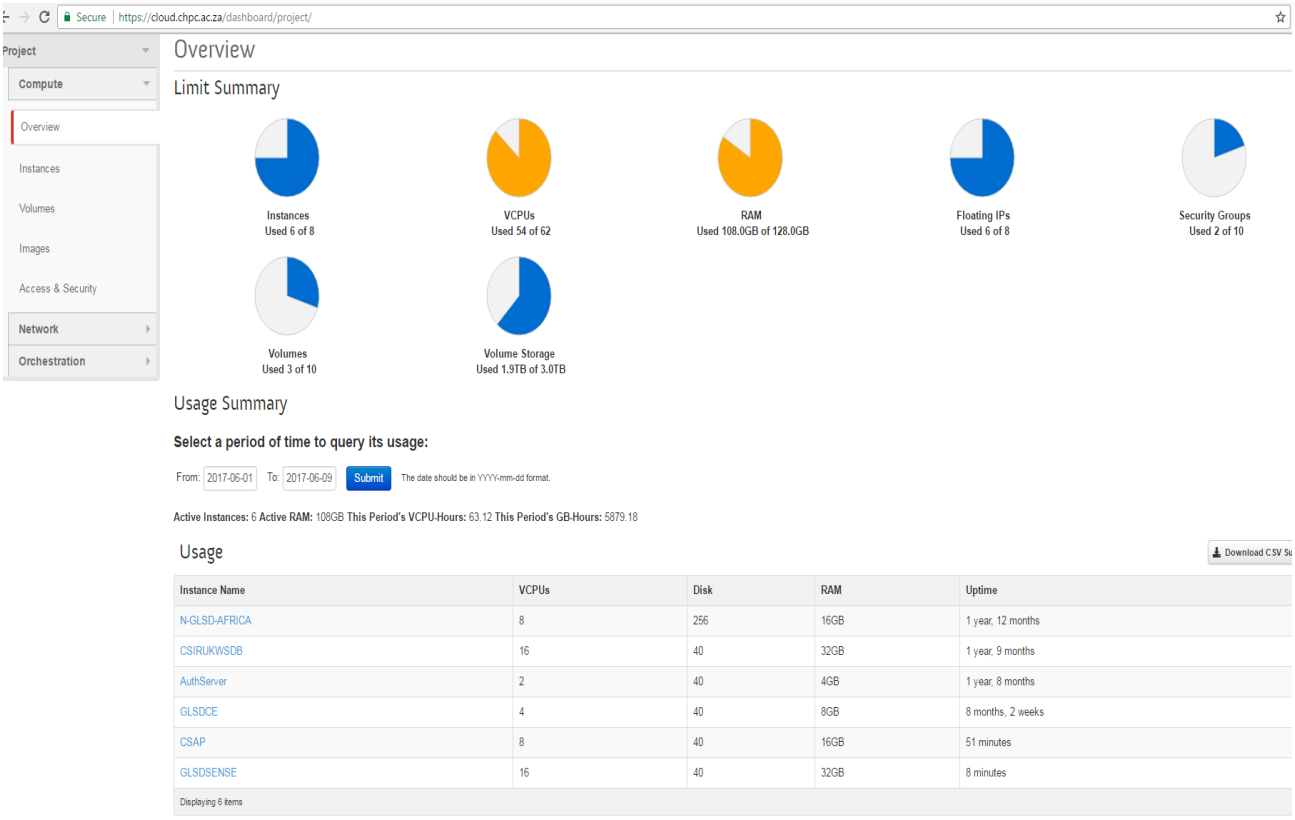
B.4. APPENDIX D: CPU CORES USAGE BEFORE EXPERIMENT



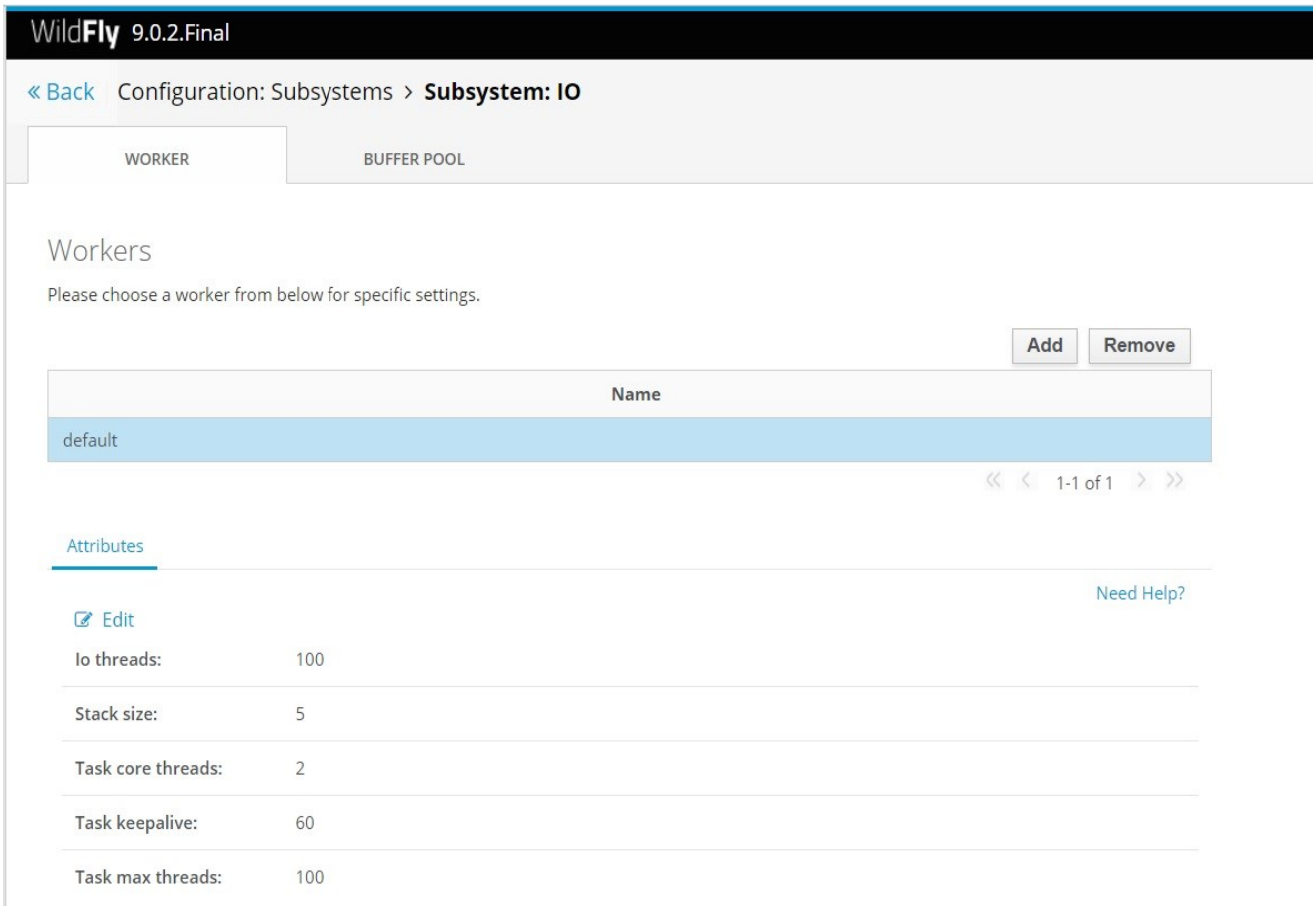
B.5. APPENDIX E: CPU CORES USAGE DURING EXPERIMENT



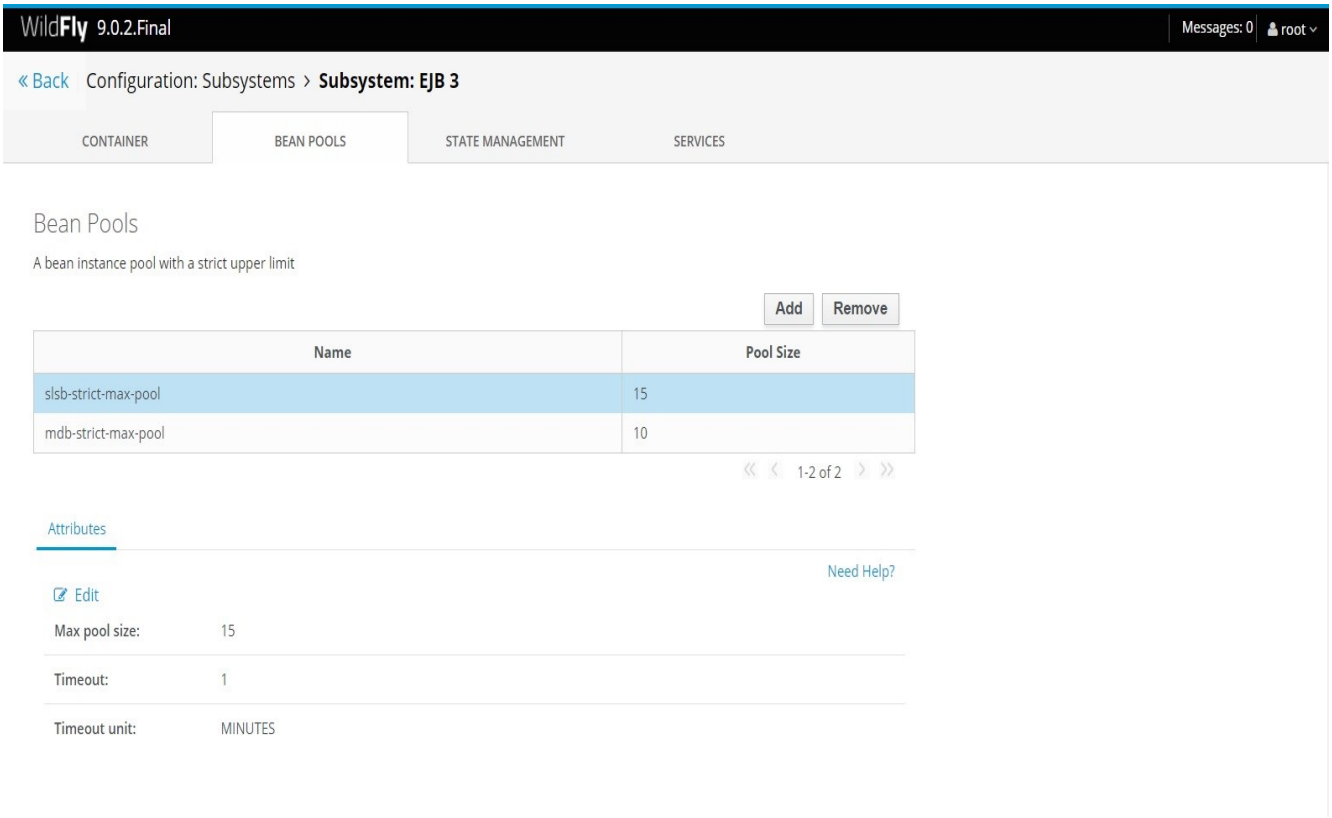
B.6. APPENDIX F: CHPC DASHBOARD



B.7. APPENDIX G: WILDFLY APPLICATION SERVER IO THREADS CONFIGURATION



B.8. APPENDIX H: WILDFLY EJB POOL CONFIGURATION



B.9. APPENDIX I: WILDFLY JDBC CONNECTION POOL CONFIGURATION