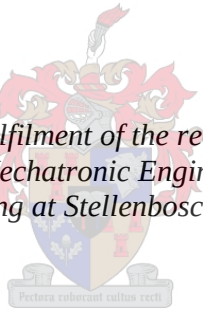
Integration of Digital Twins and Virtual Reality for Data Visualisation

by
Gabriel Santos da Silva



Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Mechatronic Engineering) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr Karel Kruger
Co-supervisor: Prof Anton Herman Basson

December 2022

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at are those of the author and are not necessarily to be attributed to the NRF.

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 14 November 2022

Abstract

Integration of Digital Twins and Virtual Reality for Data Visualisation

G.S. da Silva

Department of Mechanical and Mechatronic Engineering
Stellenbosch University
Thesis: MEng (Mechatronic Engineering)
December 2022

This thesis considers an integration of digital twins (DTs) and virtual reality (VR) for enhancing the data driven decision-making process. An integrated DT and VR system is designed according to an available complex DT system design framework. The Six Layer Architecture for Digital Twins with Aggregation (SLADTA) is used for the internal architecture of the DTs in the system. A custom developed VR application is used to visualise the information using VR equipment.

The Facilities Management (FM) Division at Stellenbosch University serves as the chosen case study context for the evaluation. The energy usage information for various facilities of FM is visualised in VR. The DT method of transferring information to VR is compared to a method that does not make use of DTs.

Three experiments are used to evaluate the two implementation methods to allow for an adequate comparison of the two methods. The experiments focus on, respectively, latency, computer resource utilisation (in terms of RAM and CPU usage), and reconfigurability when a new feature is to be added to the system. The experiment results indicate that the DT method has lower latencies, the two methods have similar computational resource needs, and the non-DT method is more reconfigurable than the DT method.

However, the DT method offers other advantages such as allowing for two VR experiences to visualise the same information, or allowing for a different visualisation tool, other than VR, to be integrated seamlessly into the system. The DT method also allows for a distributed operation functionality that reduces the computational load required from a single hardware device. The Non-DT method does not offer such advantages. The thesis concludes that the integration of DTs and VR for data visualisation is possible and is favourable for a system that will not only use VR as a data visualisation means.

Uittreksel

Integrasie van Digitale Tweelinge en Virtuele Realiteit vir Datavisualisering

G.S. da Silva

Departement Meganiese en Megatroniese Ingenieurswese
Universiteit Stellenbosch
Tesis: MIng (Megatroniese Ingenieurswese)
Desember 2022

Hierdie tesis oorweeg 'n integrasie van digitale tweelinge (DT'e) en virtuele realiteit (VR) vir die verbetering van die data-gedrewe besluitnemingsproses. 'n Geïntegreerde DT- en VR-stelsel is ontwerp volgens 'n beskikbare ontwerpraamwerk vir komplekse DT-stelsels. Die *Six Layer Architecture for Digital Twins with Aggregation* (SLADTA) word gebruik vir die interne argitektuur van die DT'e in die stelsel. 'n Pasgemaakte VR-toepassing word gebruik om die inligting met behulp van VR-toerusting te visualiseer.

Die Afdeling Fasiliteitsbestuur (FB) aan die Universiteit Stellenbosch dien as die gekose gevallestudiekonteks vir die evaluering. Die inligting oor energieverbruik vir verskeie fasiliteite van FB word in VR gevisualiseer. Die DT-metode om inligting na VR oor te dra, word vergelyk met 'n metode wat nie van DT'e gebruik maak nie.

Drie eksperimente word gebruik om die twee implementeringsmetodes te evalueer om 'n voldoende vergelyking van die twee metodes moontlik te maak. Die eksperimente fokus op, onderskeidelik latensie, rekenaarhulpbronbenutting (in terme van geheue- en mikroverwerker-gebruik), en herkonfigureerbaarheid wanneer 'n nuwe kenmerk by die stelsel gevoeg moet word. Die eksperimente se resultate dui aan dat die DT-metode laer latensie het, die twee metodes soortgelyke berekeningshulpbronbehoeftes het, en die nie-DT-metode meer herkonfigureerbaar as die DT-metode is.

Die DT-metode bied egter ander voordele, soos om twee VR-ervarings toe te laat om dieselfde inligting te visualiseer, of om toe te laat dat 'n ander visualiseringsinstrument, anders as VR, naatloos in die stelsel geïntegreer word. Die DT-metode maak ook voorsiening berekeningslading te versprei wat die berekeningslading wat van 'n enkele hardeware-toestel benodig word, verminder. Die nie-DT metode bied nie sulke voordele nie. Die tesis kom tot die gevolgtrekking dat die integrasie van DT'e en VR vir datavisualisering moontlik en gunstig is vir 'n stelsel wat nie net VR as 'n datavisualiseringsmiddel sal gebruik nie.

Acknowledgements

To my supervisors, Dr Karel Kruger and Prof Anton Basson, thank you for your wise guidance and exemplary support. Your passion and love for research has been an inspiration to me and is something that I will take with me in my life. You have showed me how to find joy in what I do and how to do it with excellence. To Dr Anro Redelinghuys, thank you for all your help with the new skills I had to learn and always being available for my queries.

To the MAD research group members, thank you all for fun coffee breaks and conversations. The work environment you guys created really helped me to enjoy and progress well with my work. Thank you all for being there to bounce ideas off of and help me enjoy the two years I spent on this project.

To my flatmates and friends, thank you for being my constant support and encouraging me to carry out my work with excitement and faith. Thank you for picking me up when I am down and for showing me that the most important things in life are the relationships we have with the people around us. You have all truly helped shape me, over the past few years, into who I am and I hope that that will continue.

To my parents, Dad and Mom, thank you for your unwavering and unconditional love and support. Thank you for allowing me to follow my dreams and go in the direction I want and feel led to go into. The pride you have for me and my work has been important in helping me accomplish what I have with my studies. To my brother, Samuel, thank you for making life challenging and never making it easy, you have helped mould me in a way that cannot be expressed. Thank you for your friendship and most importantly, being a brother I could not live without. To my fiancée, Anja, thank you for allowing me to see more of the Father's heart. Thank you for always being there, for the joys and for the trials, and for pointing me back to Jesus. Your constant love and support make me excited for what is to come.

Finally, I would like to honour God, Jesus and the Holy Spirit, who without, I would not have been able to achieve any of the things I have. The grace that has been extended to me by God through the sacrifice of Jesus has showed me a love that I had never known. I pray that this thesis brings God all the glory and honour.

To my friends and family,

Trust in the Lord for His faithfulness

"O LORD, you are my God; I will exalt you and praise your name, for in perfect faithfulness you have done marvellous things, things planned long ago."

Isaiah 25:1

Table of Contents

List of Figures

Page

List of Tables

Page

List of Abbreviations

2D	Two-dimensional
3D	Three-dimensional
API	Application Programming Interface
AR	Augmented Reality
CAD	Computer-Aided Design
CPS	Cyber Physical System
CPU	Central Processing Unit
CSV	Comma-Separated Values
DT	Digital Twin
DTA	Digital Twin Aggregate
DTI	Digital Twin Instance
FM	Facilities Management
FR	Functional Requirements
GB	Gigabyte
HMD	Head Mounted Display
IoT	Internet of Things
JSON	JavaScript Object Notation
MADRG	Mechatronics, Automation and Design Research Group
MB	Megabyte
NFR	Non-Functional Requirements
PC	Personal Computer
PLM	Product Lifecycle Management

RAM	Random Access Memory
SLADT	Six Layer Architecture for Digital Twins
SLADTA	Six Layer Architecture for Digital Twins with Aggregation
SSD	Solid State Drive
TB	Terabyte
TCP/IP	Transmission Control Protocol/Internet Protocol
UI	User Interface
VR	Virtual Reality

1 Introduction

1.1 Background

Industry 4.0 is the fourth industrial revolution that is related to concepts such as the Internet of Things (IoT), Cyber Physical Systems (CPSs) and Digital Twins (DTs). In essence, a DT is an accurate digital representation, in a digital environment, of a physical model (Redelinghuys *et al.*, 2019). Along with a DT being a virtual representation of a physical model, the DT is able to support the prediction and operation of the physical model throughout its lifecycle (Malik *et al.*, 2020).

The use of DTs typically results in the recording of a vast amount of information. This information is not just recorded to have the operational history of a DT, but also to make informed decisions for the physical system relating to the DT. This results in a data driven decision-making process, where previously recorded data is used, in conjunction with previous experience and domain knowledge, to make a decision for a particular system.

The advancement of technology has also seen the development of various virtual technologies such as augmented reality (AR) and virtual reality (VR). VR places the user in a 3D environment where, with the aid of the appropriate equipment, they are able to move around and interact with elements in the 3D environment. VR is currently mostly used in the computer or console gaming industry. However, the question arises of how VR can be used to aid and advance industries outside of the gaming industry.

At the most basic level, VR can be seen as a way of visualising information. This introduces the concept of using this visualisation medium to visualise information; specifically the information obtained by DTs. Current research into this area is fairly scarce as it is a concept that is in its infancy. An unanswered question is whether using VR to visualise DT information will be more advantageous than viewing the information using conventional methods, such as a PC monitor; or whether VR will bring additional complexities and challenges that outweigh possible advantages. The answer to this question is context specific and will require careful investigation. This thesis, therefore, does not address this question. Another question is whether DTs can aid the information visualisation process in VR. This second question is considered in this thesis.

The Mechatronics, Automation, and Design Research Group (MADRG) at Stellenbosch University's Department of Mechanical and Mechatronic Engineering focusses on enabling Industry 4.0 for the South African context. The research group focuses especially on the IoT and CPS concepts of Industry 4.0. DTs form an

integral part of CPSs and the research group believes that DTs can “improve the global competitiveness of South African enterprises, while addressing important social concerns” (Kruger, n.d.). As a result, the research group has been responsible for developing DT implementations in various industries. Some of the implementations being in the manufacturing and maritime industries, as was compared by Taylor *et al.* (2020) who are also members of MADRG. The research group also developed the Six Layer Architecture for Digital Twins with Aggregation (SLADTA) (Redelinghuys *et al.*, 2019) where, using this architecture, a DT can logically and effectively be implemented in the scenario of a manufacturing cell. However, SLADTA has the potential to be used in other implementations and not just for a manufacturing cell context.

1.2 Objectives

The main objective of the research is to evaluate the integration of DTs and VR to support data driven decision-making for complex systems.

This study includes using existing DT system concepts and newly developed concepts to identify and evaluate possible opportunities that exist with the integration of DTs and VR. Integration refers to the manner in which a DT, and its information, will most effectively be interfaced with VR so that the two technologies may complement one another; such as the contextualisation of DT data in a VR space to enable using previously recorded information to make an informed decision for the future of a complex system.

The research focuses on only transferring information for visualisation to VR and does not include making changes to a physical system using VR. The research works from an existing DT architecture (SLADTA). The research’s objective of integrating VR and DTs uses the Facilities Management Division at Stellenbosch University as a case study for evaluation. This division is responsible for overseeing the operations of complex facilities, such as a university campus. The ideas and concepts generated in this research were created to be used more generally than only for a facilities management division implementation.

1.3 Motivation

VR is growing in use as a visualisation tool that enhances the data visualisation and decision-making process. However, the need to effectively transfer information into a VR environment is rising and remains largely unexplored. This research allows for an effective method of transferring information, through the use of DTs, to the VR environment to be investigated.

The DT concept, inherently, requires the ordering of large quantities of data for the purposes of making better decisions for a physical system. DTs, therefore, provide a possible means to then also be used to transfer information to VR. This DT method of transferring information to VR could rival, what could be considered, a more "conventional" method of transferring information. However, a comparison of the two methods is required to determine the extent of this rivalry.

Facilities management, with specific focus on the facilities management division at Stellenbosch University, is a suitable case study context for evaluation of integrating DTs and VR because it is a complex system, with many components and complex relationships. DTs can provide an accurate and near-real time reflection of reality which, for facilities management, is vital for decision-making. This presents an opportunity of DTs to be used in this context to accurately reflect this complex system. The opportunity of using VR for data visualisation to aid in the data driven decision-making process for facilities management is also present as large amounts of data about various facilities is recorded for decision-making purposes. These opportunities indicate the possibility and need for an effective DT and VR integrated system in the facilities management context.

The selection of SLADTA, mentioned in Section 1.2, as the internal architecture for DTs in a system is supported by the implementations of Redelinghuys (2020) and Human (2022). SLADTA has been shown to aid with logically separating a physical system into hierarchical components, whose information can then be aggregated and used for decisions. This hierarchical separation and aggregation are concepts that are applicable to the facilities management context, further supporting the selection of SLADTA as the internal architecture for the DTs.

The current study is a steppingstone to highlight the potential of using DTs to aid a VR experience for data visualisation and decision-making. During this study, VR is used purely as a visualisation tool. However, the technology has the potential to also be interactive and have bi-directional communication, where the physical system of a DT can be altered by a user interacting with the DT in a VR environment, and vice versa.

1.4 Methodology and Overview

This study investigates the opportunities for a data driven decision-making process that can be improved by the integration of DTs and VR. A literature review, provided in Chapter 2, is first conducted to gain insight into the current research of DTs and VR, and how these concepts fit into the broader context of Industry 4.0. The literature review contains information regarding DTs, the design of complex DT systems, SLADTA, the VR technology and some of the uses of VR. This literature review is used to provide valuable insight into how these concepts can relate to one another and what research has previously been conducted on these concepts.

Chapter 3 indicates various opportunities and challenges, associated with using DTs and VR, that are identified. New concepts of how to use VR and DTs together are developed and formulated based on these identified opportunities and challenges. These new concepts primarily focus on how the integration of VR and DTs can potentially be beneficial and advantageous in a data driven decision-making process. Facilities under the authorisation of the Facilities Management Division at Stellenbosch University are used in the implementation case study. The case study is selected, from the facilities management context, based on how well the integration of VR and DTs can be displayed in the case study. The selection and design of this case study is presented in Chapter 4.

The DT system for this selected case study is implemented and integrated with VR to enable a user to visualise facility information. A non-DT method for visualising the same facility information in VR is also implemented. Both implementations produce the same output and have the same functionality. The two implementations are compared and reviewed to determine the advantages, if any, of integrating DTs and VR for the data driven decision-making process. The DT and VR integration also shows the use of SLADTA as a possible internal architecture for the DT system. The implementations do not allow for changes to be made to facilities in the case study using VR. The details regarding the two implementations, as well as the VR application, are provided in Chapter 5.

The selected case study only uses the energy usage information for the various facilities in the system. This information is reliably available and constantly being updated, unlike other utility usage information. The use of only energy usage information of the facilities allows for the concept of integrating DTs and VR to be showcased, and the addition of other information for visualisation would have had very little benefit. The case study is focused only on the main campus, located in Stellenbosch, for Stellenbosch University. The selected campus is sufficient in displaying the integration of DTs and VR with various system hierarchical levels. Stellenbosch University, as a whole, is not selected because this additional level in the system hierarchy provides little benefit for the subsequently increased complexity of the system.

In Chapter 6, the case study implementations are evaluated using different metrics, such as system latency, computational resource utilisation, and system reconfigurability. The evaluation results of the two implementation methods are used to determine whether the use of a DT and VR system for the data driven decision-making process is, indeed, beneficial or not. Chapter 7 discusses the various components and outcomes of the research. Any possible areas for further work are also provided in this chapter. A conclusion, recalling the objectives of the research, is made based on these evaluation outcomes and is presented in Chapter 8. It is important to note that the research considers published work, as well as newly developed work, to achieve the objectives stated in Section 1.2.

2 Literature Review

The chapter provides information of the various aspects that are used within this research. Digital twins are discussed, followed by a discussion for an existing design framework for designing complex DT systems. The internal DT architecture SLADTA is then discussed. The VR technology and its various components and uses are then discussed. The final section of the chapter is a discussion of how the various aspects relate to one another for the purposes of this research.

2.1 Digital Twins

Industry 4.0 is the result of many years of technological advancements. Some of these technologies include the Internet of Things and Cyber Physical Systems. Using these technologies, opportunities for greater levels of productivity are seen in the concept of digitalisation in manufacturing (Uhlemann, *et al.*, 2017). Industry 4.0 technologies allow for intelligent components to become easily interconnected and the components to be integrated (Negri *et al*, 2017). The digital technologies have enabled the idea of virtual product and process planning. This results in a vast amount of data being collected, stored, and analysed to make decisions (Kritzinger, *et al.*, 2018).

The DT concept is a concept that has arisen due to the advancement of Industry 4.0 technologies. The term DT had first been used by Grieves in 2002. The term was used in a presentation of product lifecycle management (PLM) in an industry context (Grieves & Vickers, 2017). Grieves and Vickers (2017) stated that a DT is a "digital informational construct about a physical system, created as an entity on its own and linked with the physical system in question". Included in the DT should be any information within the physical system that could be obtained by viewing the system in the real world (Grieves & Vickers, 2017). The DT concept was also mentioned with regards to use for future NASA and U.S. Air Force vehicles where the definition of a DT was "an integrated multi-physics, multi-scale, probabilistic simulation of a complex product and uses the best available physical models, sensor updates, etc., to mirror the life of its corresponding [physical] twin" (Glaessgen & Stargel, 2012).

A key feature and benefit of a DT is to be able to provide data in a consistent format. DTs are not simply just data; they include algorithms that are able to reflect their physical twin and make decisions based on processed data. A definition of a DT is that it is the digital counterpart/representation/reflection of a physical system. However, it has been identified that the terms Digital Model, Digital Shadow, and DT are being used interchangeably (Kritzinger *et al.*, 2018).

The terms Digital Model, Digital Shadow, and DT can be seen to differ in the level of the data interaction between the physical twin and the DT, as illustrated in Figure 1.

A Digital Model refers to a digital representation of a physical system that does not make use of an automated data transfer between the physical system and the digital representation. A change in the physical system will have no direct impact on the digital representation or the other way around (Kritzinger *et al.*, 2018).

A Digital Shadow refers to when there is an automated flow in one direction, from the physical system to the digital representation, but the opposite direction of data flow is still manual. A change in the physical system results in an automatic change in the digital representation but not conversely (Kritzinger *et al.*, 2018).

If the data flow between the physical system and the digital representation are fully automated in both directions, the digital representation is deemed to be a DT. The digital representation could be acting as a controlling instance for the physical system. A change in the physical system will result in a change in the digital representation and the other way around (Kritzinger *et al.*, 2018).

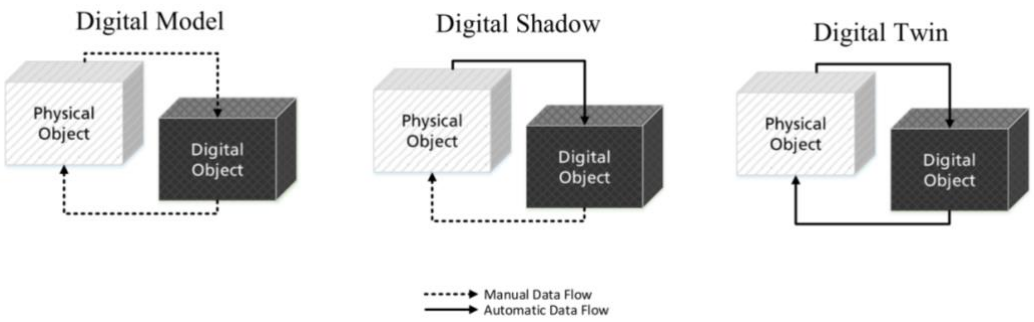


Figure 1: Digital representation term data flows (Adapted from Kritzinger *et al.*, 2018)

Due to the differences in the level of interaction between a physical system and its corresponding digital representation, the technologies needed for each implementation varies. The general technologies required are: simulation methods, communication protocols, and other technologies that form part of Industry 4.0 (IoT, Cloud Computing, Big Data, etc.) (Kritzinger *et al.*, 2018). Kritzinger *et al.* (2018) show that the concept and development of DTs is still in its infancy because most literature consists only of having concepts of DT technologies and not having any suitable case-study implementations.

Some advantages of DTs are that they can provide accurate representation or simulation for system components. DTs can act as a processing-monitoring tool for a user to be able to predict potential incidents, identify failures and can be used

to optimise the system. The realistic representation of a physical system using a DT allows for this optimisation aspect to provide realistic outcomes for the system (Havard *et al.*, 2019).

The definition of a DT in this research is a virtual representation of a physical system, where the DT is seen as an entity that can communicate with the physical system.

2.2 Complex Digital Twin System Design Framework

The "DT" of a complex system is also complex, to the extent that it can be referred to as a "complex DT system". The design of such a complex DT system requires various aspects to be taken into account for implementation in a given context with desired functionality being required. A framework was developed by Human (2022) for designing complex DT systems, and this section reviews this design framework. This is a suitable complex DT design framework as SLADTA, required in this research (Section 1.2), is already considered and used in the framework. No other alternative, yet comparable design frameworks could be found in literature.

The framework was developed to provide general principles to be used to design a DT system architecture, with some recommended implementation decisions, for managing complex physical systems. These complex systems are seen as a large network containing many components that result in complex behaviour and information processing, and require adaptation or evolution (Human, 2022). DTs, especially DTs with aggregation, are seen as a potential solution to managing these complex systems. Human (2022) states that the purpose of the design framework is "to enable systematic, effective decisions when designing a system of DTs to represent a complex physical system".

Figure 2 illustrates the design framework, which is divided into the *Problem space* and the *Solution space*. The *Problem space* receives the user's requirements and the physical system as inputs. Specified design steps are then followed, and the output of the *Problem space* is then used as input for the *Solution space*. The output of the *Problem space* is a list of system functional requirements, a list of non-functional requirements, a hierarchical physical system decomposition, and the characterisation of the data present in the system. Functional requirements are the functions in a system that will achieve the user requirements. Non-functional requirements are how or how well these functional requirements are achieved (Human, 2022).

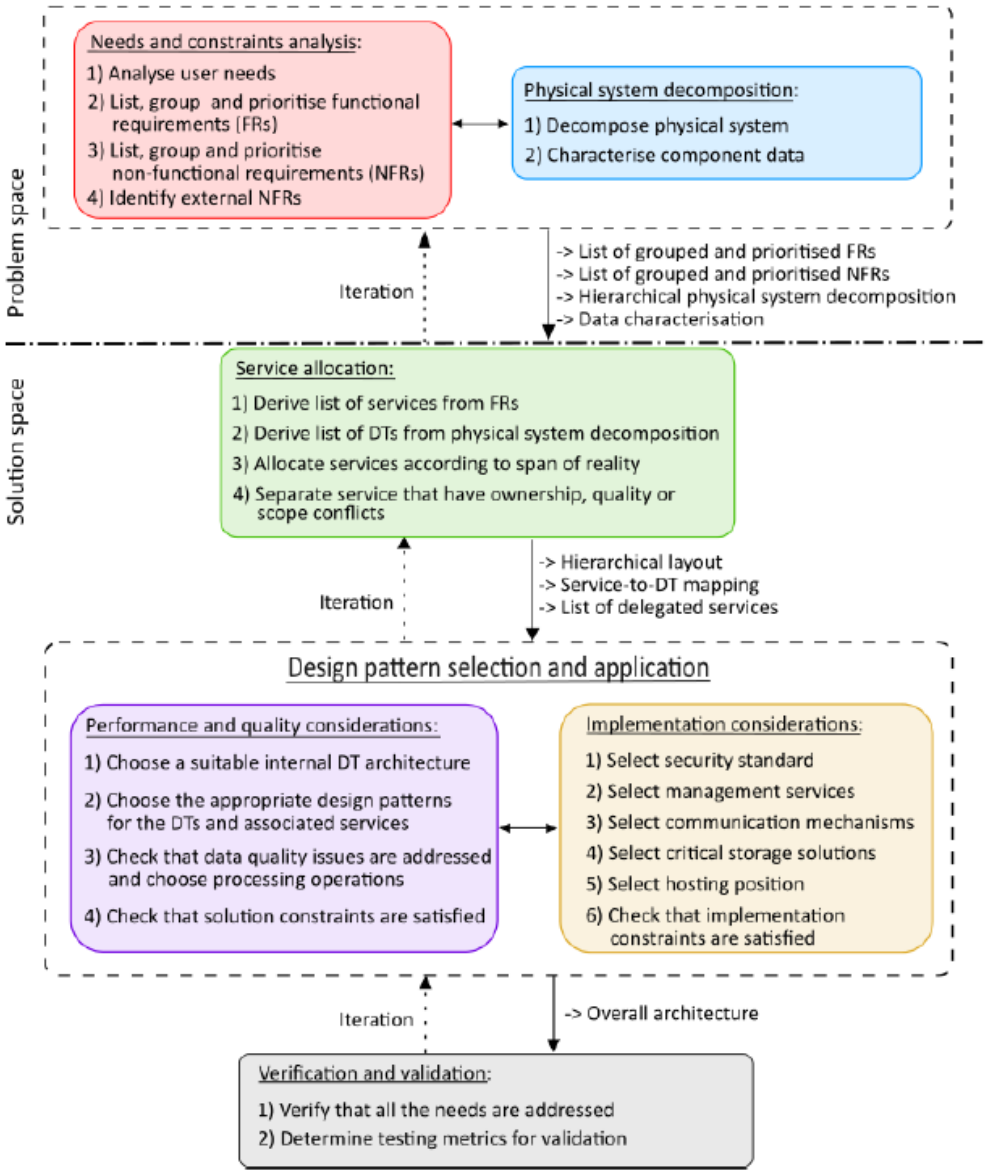


Figure 2: Complex DT design framework (Human, 2022)

The design process steps for the *Solution space* are then followed and the output for this space is the overall system architecture. Once the overall system architecture has been designed and implemented, various validation and verification tests are carried out to determine whether the user's needs are met and to what degree they have been met. The *Solution space* development begins with identifying the services required in the system to meet the functional and non-functional requirements from the *Problem space*. Using the hierarchical physical system decomposition, DTs are derived to reflect the physical reality. The identified services are then allocated to the DTs in the system or to a services

network. The Shared Services component is a set of services, including the services network, offered to, or “shared” by the DT components in the system. The Shared Services component of the DT system could contain various system management services in addition to the services network, such as a service gateway, a directory service, an orchestration service, etc. These additional services are used to aid the operational aspects of the DT system.

Human (2022) gives a number of design patterns and implementation considerations that guides the design in ensuring that the implemented system is able to achieve, firstly, the user requirements and, secondly, any additional system requirements that might not be directly stated in the user’s requirements. A design pattern is a set of architecture and implementation recommendations that prioritise certain system requirements. Human (2022) identified six design patterns for, respectively, performance efficiency, reliability, maintainability, compatibility, portability, and security.

Before implementation details are considered, an overall DT system architecture and an internal structure for the DTs must be selected. Human (2022), and this research (as stated in Section 1.2), used SLADTA as the internal structure of the DTs and the reference architecture shown in Figure 3 for the overall architecture.

After design, the overall system architecture must then be implemented, and various validation and verification evaluations should be carried out to determine if the system achieves all of the user requirements provided at the start of the design phase. This evaluation phase could also be used to determine to what degree, or how well, the requirements are achieved by the system.

It must be noted that during the design process, iteration between phases is required to ensure that the correct system is designed. The final system architecture might not be fully designed during the first iteration as some considerations and implementation selections could affect other selections made previously in the design process.

Human (2022) used the framework to design architectures for three different complex system case studies, i.e. a water distribution system, a smart city, and a heliostat field. The water distribution system and smart city case studies were more high-level case studies, while the heliostat field case study was more in-depth and showed detailed application of the framework. The architecture for the heliostat field complex system was implemented to validate the design framework. The evaluation of the heliostat field case study showed that the framework was "able to guide the design of a feasible architecture for a system of DTs” (Human, 2022). This supports the notion of the design framework being able to be applied more generally to other contexts.

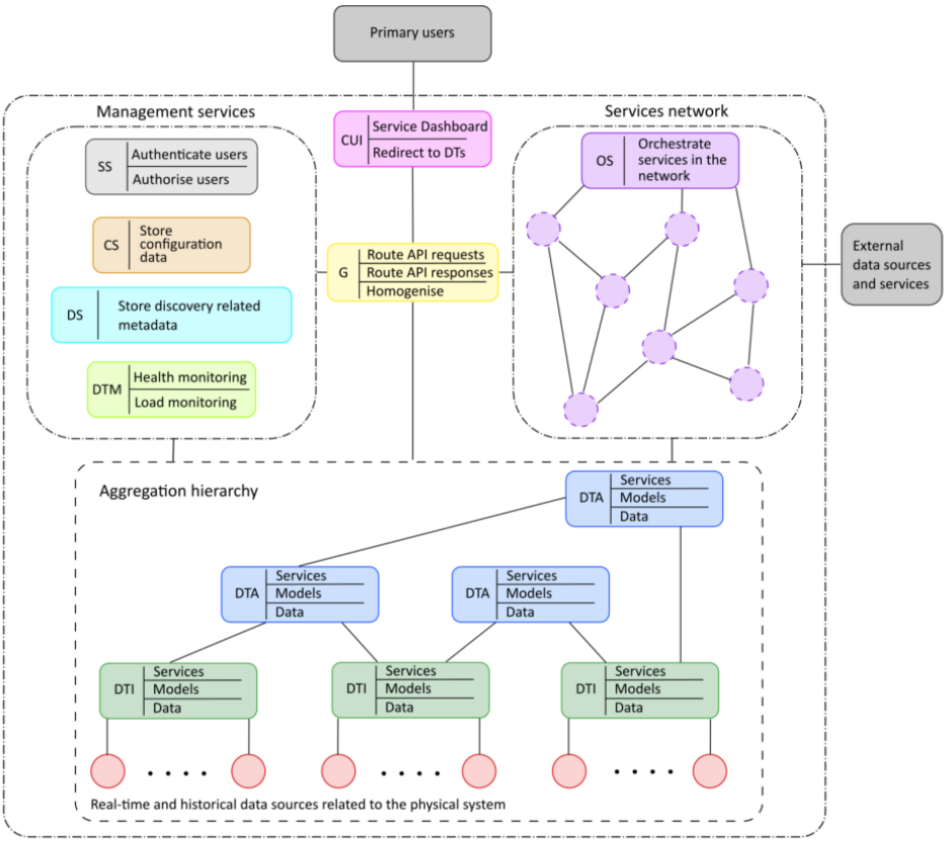


Figure 3: Design reference architecture (Human, 2022)

2.3 Six Layer Architecture for Digital Twins with Aggregation

As mentioned previously, the Six Layer Architecture for Digital Twins with Aggregation (SLADTA) is used as the internal architecture for the DTs in the system. An alternative architecture to SLADTA that could be considered is the architecture developed by Borangiu *et al.* (2020). The two architectures are briefly compared at the end of this section.

SLADTA is an adaptation of the Six Layer Architecture for Digital Twins (SLADT), which has been configured to allow for the aggregation of DTs. SLADT was initially developed for the implementation of a DT of a manufacturing cell (Redelinghuys, 2020). Figure 4 shows the basic structure of SLADT.

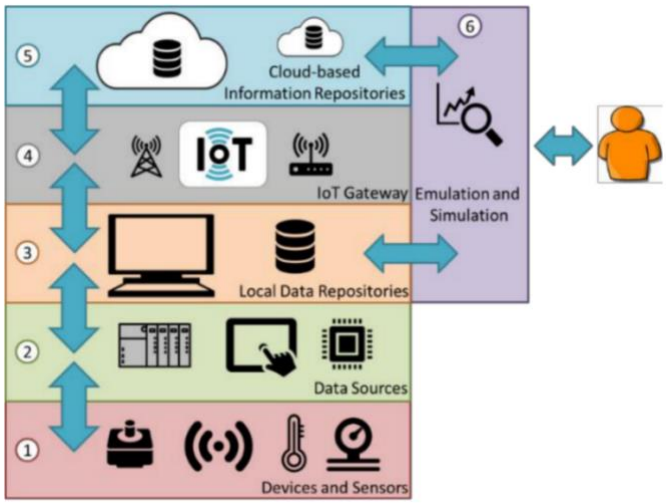


Figure 4: SLADT reference architecture layout (Redelinghuys *et al.*, 2019)

Figure 4 shows the data and information flow from a physical twin (Layer 1) to long-term storage, typically hosted in the cloud, but also locally (Layer 5). The information can also flow in the other direction from the cloud to the physical twin. Layer 4 contains functionality to convert data into information before it is sent to Layer 5. Layer 6 contains the simulation or emulation software that uses the information provided by the physical twin (Redelinghuys *et al.,* 2018). Each layer of SLADT, Layer 1 to Layer 6, is expanded on below.

Layer 1 contains the different physical devices (sensors and actuators) that can be used to provide information to the controllers in Layer 2. These devices provide information related to the physical twin and they are seen as being a part of the physical twin. Layer 2 is seen as a distinct layer because, although the controllers are connected to the physical twin, they are able to supply specific functionality to the DT. They transfer the data to Layer 3 (Redelinghuys *et al.*, 2018).

Layer 3 contains short-term local repositories of the stored data for the physical twin. These local repositories are located near the physical twin. This layer is supposed to be able to support vendor neutral integration, such as an OPC UA server that can be used to transfer and collect the data (Redelinghuys *et al.*, 2018).

Layer 4 is the gateway between the physical twin and the cyber world. This layer converts the data stored in Layer 3 into information that is stored in Layer 5 and used in Layer 6 (Redelinghuys *et al.*, 2018). The layer is custom developed software and, as mentioned previously, can be used to provide some specific functionality to the DT using the data from Layer 3 and, possibly, Layer 5.

Layer 5 includes cloud-based or local database servers that are used as a long-term information repository for the information obtained regarding the physical twin

and the DT. The information stored is generally the history of the physical twin and the physical twin's current or latest state (Redelinghuys *et al.*, 2018). This information, along with the data in Layer 3 is used in the emulation or simulation software that is present in Layer 6, as seen in Figure 4.

Layers 1 to 5 are seen as the infrastructure for the DT and the actual intelligence is provided in Layer 6. This layer should transfer information between the physical twin via the long-term cloud, or local repositories in Layer 5 or even the short-term local repositories in Layer 3 (Redelinghuys *et al.*, 2018). Layer 6 is responsible for making decisions for the DT that will impact the physical twin. The decision will then be sent back through the architecture to the physical twin.

SLADT was developed to be vendor neutral and allow for clear distinctions to be made between the roles of different layers in a DT architecture. SLADT was developed to allow for the use of off-the-shelf components without having to use a specific vendor. Redelinghuys (2020) showed that SLADT is useful for developing the DT of a physical twin in cyberspace.

SLADT was only demonstrated for a single physical twin and was, therefore, adapted to allow for multiple DTs to be connected. This adaptation is called SLADTA and has the functionality to aggregate DTs for more complex systems.

In SLADTA there are two types of DTs, i.e. digital twin instances (DTIs) and digital twin aggregates (DTAs). The two types were derived from, but differ from, other definitions (Grieves & Vickers, 2017). A DTI corresponds to the digital twin that is connected to the physical twin for the entirety of its lifespan. A DTA is the aggregation of various DTIs, as well as other DTAs. A DTI can be viewed as independent, but a DTA is dependent on other DTIs and DTAs (Grieves & Vickers, 2017). Redelinghuys (2020) identified that there was a need for the idea of a "digital twin of twins". This is the concept of a having an aggregation architecture for DTs.

Figure 5 shows the architecture layout for SLADTA. In the architecture, every physical twin has its own DTI that comprises of a SLADT implementation with all six layers of the architecture. The higher-level twins (DTAs) only contain the relevant layers of SLADT, i.e. Layer 3 to Layer 6. The connection between DTs (DTIs and DTAs) occurs through Layer 3 of each DT. Layer 4 is used to manage the interactions between DTs, but the actual information is transferred between Layer 3 of the DTs (Redelinghuys, 2020). Further developments using SLADTA (Human, Basson & Kruger, 2021), however, evaluated having DTs communicate with each other through Layer 4 which differs from the originally proposed architecture by Redelinghuys (2020). This new communication through Layer 4 approach, is more generally applicable (Human *et al.*, 2021) and should, therefore, be implemented in place of the original design by Redelinghuys (2020).

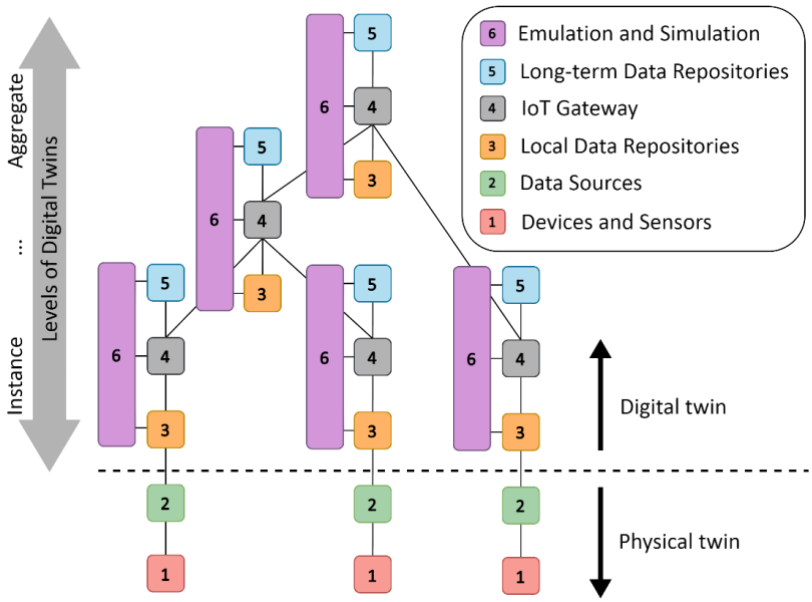


Figure 5: Architecture layout of SLADTA (Adapted from Redelinghuys *et al.*, 2020)

SLADTA is designed to be reconfigurable and flexible because large hierarchical structures can be costly to design, maintain and modify. SLADTA allows for the aggregation of information from different DTs, but also allows for the information to be segmented. Aggregating information from multiple DTs reduces the complexity by encapsulating the functionality of related information to each DT. Each DT can make its own decision and is flexible and intelligent (Redelinghuys, 2020). These are some of the advantages of using SLADTA for DTs.

A digital twin technology review (Juarez *et al.,* 2021) showed that the architecture developed by Borangiu *et al.* (2020) is the most similar to SLADTA as provision is made for aggregation, but the architecture does not have some of the other advantages that are offered by SLADTA. The architecture developed by Borangiu *et al.* (2020) makes use of the aggregation concept as data is acquired in the first layer of the architecture and is then aggregated in the second layer to develop the process models represented in the DTs. The third and fourth layers of their architecture are responsible for the analysis and decision makingt of the DT. The area where the architecture is lacking compared to SLADTA is where SLADTA offers the possibility of a “digital twin of twins”, where available information is both segmented and aggregated. This “digital twin of twins” is not possible with the architecture of Borangiu *et al.* (2020) as the data received by the sensors is aggregated in the architecture without each physical twin having their own DT and then aggregating the information to other DTs like with SLADTA. This segmenting of information allows for a better representation of the physical as each element is represented by a DT.

2.4 Virtual Reality

Virtual Reality (VR) is a newly developing technology that could be beneficial for use in the data visualisation and decision-making process. The technology has been available for a number of years, but due to recent developments, there has been an increase in the accessibility to visualisation tools like VR (Havard *et al.*, 2019). A brief overview of VR is provided next, followed by the hardware and software aspects for VR, and different uses of VR.

2.4.1 Virtual Reality Overview

VR is a digital artificial environment that a human's senses will perceive to be real. The core of VR is to be able to create a near real environment. This environment should be able to simulate or present a physical environment in real time (Liagkou *et al.*, 2019; Malik *et al.*, 2020). A user in VR is able to enter an immersive interactive environment that interprets the user's prompts to respond accordingly to the user's behaviour (Sekaran *et al.*, 2021). The main purpose of VR is to provide the user with a "multi-modal, close-to-reality experience". Considering visual senses, VR is able to showcase the highest level of graphic benefits when considering the dynamic and immersive visualisation in comparison with VR's visualisation method counterparts (Sekaran *et al.*, 2021). Users are able to experience a virtual environment with high detail, allowing them to experience a fully immersive environment that cannot be achieved using a conventional desktop computer (Andersen *et al.*, 2019).

VR is characterised by three main aspects called the "3I's" (Phoon *et al.*, 2017):

- Imagination – The user can participate in a scenario simulation which is created in a virtual environment.
- Interaction – A VR system must be able to react to some actions or behaviours carried out by the user.
- Immersion – A VR system must be both mentally and physically immersive. The mental immersion is achieved by the quality, in terms of rendering and simulation, of the environment that the user is experiencing. The physical immersion is accomplished by updating what the user is seeing in real-time according to their actions or prompts.

In contrast to the above, some researchers divide VR into two categories, namely immersive and non-immersive VR. Non-immersive VR is when users can visualise models or information on a screen, for example desktop PC screens or widescreen projectors. Immersive VR often makes use of head mounted displays (HMDs) that allow a user to virtually "enter" and interact with an environment to visualise models or information. Nowadays, when VR is mentioned, it is mostly referring to

immersive VR (Malik *et al.*, 2020). Figure 6 is an illustration of the spectrum of VR with regards to the level of immersion.

In this thesis, the focus is on visual immersion and not the other senses.

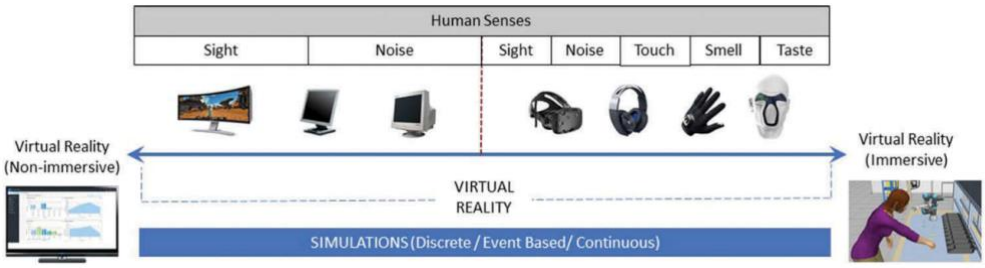


Figure 6: VR spectrum (Malik *et al.*, 2020)

2.4.2 Virtual Reality Hardware

VR is a technology that creates a real-world visual perception by using artificial computer-generated environments. This is achieved by using a combination of three effects: a total immersion experience; stereoscopic vision – where a scene is rendered for each of a user's eye at a slightly different angle; and motion capture – where a user's head and controller position are used to update the environment for the user (El Beheiry *et al.*, 2019). A user can interact with VR using a variety of methods. Those interaction methods include HMDs, console gaming controllers, keyboards, and haptic devices (Sekaran *et al.*, 2021). Other interaction hardware includes gloves, a 3D mouse, a space ball, voice recognition, biological sensors, and full-body suits (Liagkou *et al.*, 2019). VR applications require some input from the user, and this is generally achieved by using keyboards, mouses, or joysticks. However, these devices "break the illusion that users are directly interacting with the virtual world because they are non-intuitive ways to interact with virtual objects" (Erra *et al.*, 2019).

There are different types of VR implementation hardware, and many VR platforms provide multi-user collaboration in virtual environments. These include CAVEs, PowerWalls, and HMDs. The differences in these implementations are the degree of user immersion, the maintenance effort, and the cost. CAVEs and HMDs are seen as full-immersion VR devices, whereas PowerWalls or monocular head-based VR devices are semi-immersive (Kroupa *et al.*, 2018).

The quality and realism of a VR application can influence the quality of user immersion. The cost of VR is an important consideration because, as expected, more expensive equipment provides the user with a better immersive experience (Liagkou *et al.*, 2019). Previous VR technologies used complex and non-portable equipment such as CAVEs or hyperwalls (Donalek *et al.*, 2015). A VR CAVE

implementation is where images are projected onto walls and an environment is created around a user. This VR implementation is expensive and requires large amounts of space to be implemented. This setup has been implemented in various universities and data centres to enhance visualisation in areas of science and engineering (Cordeil *et al.*, 2017).

A newer VR setup is the use of the previously mentioned HMDs. The technology has been developed to "ensure visual comfort and ergonomic usage" (El Beheiry *et al.*, 2019). The technology has also become available to many consumers by the introduction of VR headsets such as HTC Vive, Oculus Rift, and Windows Mixed Reality that are now affordable (El Beheiry *et al.*, 2019). Current HMDs are more portable and affordable than creating CAVE environments or previous HMD versions. This makes them an ideal tool for exploratory visualisation (Drouhard *et al.*, 2015). As mentioned previously, VR applications require inputs from the user and some of these inputs could result in the user feeling less immersed. However, with more recent VR equipment developments, such as the HTC Vive and Oculus Rift, the user can make use of controllers to interact with the environment. The use of these controllers allows for the user to feel almost completely immersed in the environment and enhances their VR experience.

2.4.3 Virtual Reality Software

There are many software development environments that could be used to develop a VR environment, including Unity, Unreal Engine 4, Google VR for everyone, Amazon Sumerian, Blender, 3ds Max, and Maya to mention a few (Davies, n.d.). The reason for the large number of development environments is the increasing popularity and use of VR. This increased use creates the opportunity for competition to arise between different entities, resulting in a multitude of possible options to choose from.

Related research has shown that Unity and Unreal Engine are the most widely used options (Donalek *et al.*, 2015; Havard *et al.*, 2019; Kroupa *et al.*, 2018; Liagkou *et al.*, 2019). This is due to their wide use in the game development area, their large community support base, and the familiarity of the development language used. Unity uses C# and Unreal Engine uses C++. Unity and Unreal Engine are both game engines and, thus, operate very similarly. Although related research has shown to consider both software environments, a large portion make use of Unity to develop the VR environments (Donalek *et al.*, 2015; Havard *et al.*, 2019; Kroupa *et al.*, 2018; Liagkou *et al.*, 2019).

Unity makes use of two major aspects, i.e. GameObjects and the Components attached to the GameObjects (Kuts *et al.*, 2019). The GameObjects are seen as any object that is used within the developed environment (Unity, 2017a). The use of the GameObjects allows for a modular approach of development to be adopted.

The term GameObject encapsulates multiple objects, including models, geometries, or effects, used within the environment. The Components that are attached to these GameObjects differ depending on the purpose of the GameObject. The Components include custom C# scripts or built-in functionality offered by Unity that are used to control the behaviour of the GameObject (Kuts *et al.*, 2019).

Figure 7 provides an example of a GameObject used in a Unity project and the Components that are attached to the GameObject. The GameObject is the grey cube on the left side of the figure, with the Components being in the tab on the right side of the figure. A Component example is the Transform Component that is used to control the position, rotation, and scale of the GameObject within the developed environment.

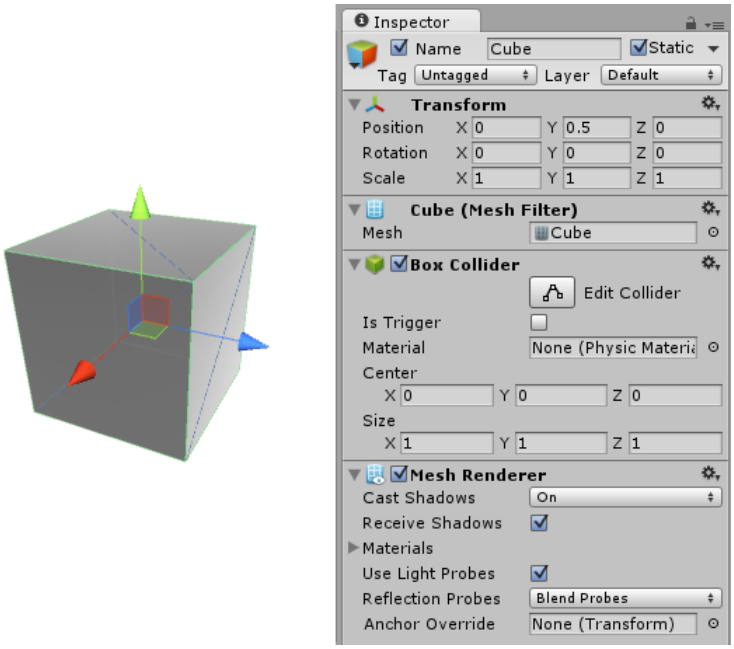


Figure 7: Unity GameObject and Components (Unity, 2017b)

GameObjects and Components are used in various combinations to develop a desired environment in Unity. Unity provides multiple built-in GameObjects and Components that aid in the development of VR environments. These are used, then, to help a developer more easily develop a VR environment with the functionality and appearance that they desire.

Unity is a powerful tool to aid in the creation of both 2D and 3D environments which makes it a suitable choice for the development of complicated environments such as VR environment.

2.4.4 Virtual Reality Implementations

VR is typically associated with use in the gaming and entertainment industry; however, it is beginning to be used in other areas and industries (Hu *et al.*, 2021). VR's use in science is still being elaborated (El Beheiry *et al.*, 2019). The data and information obtained through integrated communication channels can be visualised using VR. This data and information include operational, reporting, and monitoring data and information (Kovar *et al.*, 2017). For VR implementations in Industry 4.0, it must be adaptable and able to change to new events and situations (Liagkou *et al.*, 2019).

VR is becoming easier to implement as hardware and software is continually being advanced. The technology is being used in more activities such as product or process design, facility layout planning, training, and remote collaboration (Havard *et al.*, 2019). There is a useful potential in combining VR and DT technologies to aid in training people in a virtual environment that behaves realistically. Using correct software and the integration of these two technologies could allow for dynamic virtual environments to be created for training or designing sessions (Havard et al., 2019). This indicates the potential of integrating the two technologies. Liagkou *et al.* (2019) stated that "Industry 4.0 could benefit from using VR models to display larger and complex processes and products referring to training, simulation, maintenance and all the aspects of production line and/or at management activities". VR has been used in applications where it has been used to document the design and validation process of complex systems (Malik *et al.*, 2020).

Using VR as a tool for enhanced interaction and virtualisation allows for processes such as design, evaluation, and management improvement to be carried out in a way that is both cost and time effective. VR can be used to accelerate and streamline a task as it has the ability to represent models with a high level of detail and provides an interactive user experience (Sekaran *et al.*, 2021). Using virtual spaces as testing areas is a way to design and evaluate complex systems (Malik *et al.*, 2020).

Previous work has shown that scientists benefit from immersion for palaeontology, brain tumours, shape perception, underground cave analysis structures, MRI, organic chemistry, and physics (Donalek *et al.*, 2015). Other areas of implementation include automotive engineering, aerospace engineering, medicine and mechanical engineering (Kovar *et al.*, 2017). Another area of use is with regards to manufacturing where VR could be used for analysing products in the design stages, analysing the interaction between the final product and customers, optimising or designing manufacturing processes, and remotely monitoring/supervising processes within a system (Liagkou *et al.*, 2019). Engineers and technologists can use VR to test and experiment with a system in a way that

is close-to-reality and intuitive. This can be done before any further developments are made. Using this approach decreases the probability of system failures (Malik *et al.*, 2020).

2.5 Discussion

DTs have been shown to be useful for implementation for complex systems and accurately reflecting their physical systems. DTs provide the benefit of adding value to recorded data and allow for the data available in complex systems to be ordered, in a consistent format, and accessible. This functionality of ordering information and allowing it to be accessible makes DTs a suitable method of transferring information to various applications, like VR.

The design of a DT system for a complex system has currently no consensus due to the infancy of the concept and/or the diversity of applications. However, the design framework developed by Human (2022) offers a solution to the challenge of designing a complex DT system. Included in this design process is the need to select an internal DT architecture. SLADTA is a promising internal DT architecture that considers the interactions between DTs at different levels within a complex system. SLADTA also provides a logical separation between the different components that are contained within a DT. These different components ensure that the desired functionality of a DT is achieved. For complex systems, with many elements, there is value in a hierarchical separation of elements within the physical system, but for information from these various elements to still be aggregated and accessible. SLADTA makes provision for the aggregation of different DTs at various hierarchical levels. This further supports its use for the internal architecture of DTs of a complex system.

The VR technology is shown to have various hardware and software components needing to be considered. The literature review indicates that a choice of Unity as a software development environment is supported by various implementations in scientific VR applications. The VR implementations provided show how VR can be used to visualise information and enhance the visualisation process. However, in VR implementations the information must be transferred to the VR application in some manner that is not always trivial.

VR can also be used to visualise information from real-world complex systems, but these complex systems require an effective method for transferring information to VR that can cope with the complexities of the system. A well-designed DT system for a complex system could be a possible method for achieving this effective information transfer. Literature has mentioned the potential of integrating DTs and VR for data visualisation. However, some of the mentioned VR implementations, for more complex systems, did not make use of DTs for information transfer, and as such, this is an integration opportunity worth

exploring. VR also has a spatial component that could be well supported by DTs reflecting a physical system. This is because a physical system 3D model could form part of a DT, and like with transferring other information, this physical system 3D model could also be transferred to VR and be part of the visualisation.

From the literature, it is therefore apparent that, although the DT and VR technologies can be used individually in respective applications, they can also be integrated to enhance and support the data driven decision-making process for complex systems. It is important in such a system that: the purpose of the DTs is understood clearly; the DT system architecture is designed by following a credible design process with a suitable internal DT architecture; and that a VR application, using DTs, is appropriately developed to allow for this enhanced data driven decision-making process to be realised.

3 Integration Opportunities and Challenges

This chapter discusses various opportunities and challenges that are associated with VR as a visualisation technology, and then also the integration of the VR technology with DTs. The opportunities of using VR for data visualisation are first mentioned, followed by the opportunities for the integration of VR and DTs. After this, the challenges of VR for data visualisation are provided, followed by the challenges of integrating VR and DTs. It must be noted that the lists of opportunities and challenges provided below are not exhaustive. A brief discussion regarding the various opportunities and challenges is then provided. The information provided in this chapter is developed from that of Da Silva *et al.* (2022), unless another reference is given.

3.1 Data Visualisation Opportunities

The use of VR presents several opportunities for the purposes of data visualisation:

- Users are able to move around through data more easily and navigate the data in a manner that is more intuitive for people. This intuitive data navigation enhances and aids the pattern recognition process (El Beheiry *et al.*, 2019; Erra *et al.*, 2019). As a VR environment is a 3D space, the user is able to interpret and visualise the data presented to them in a way that is more familiar. Humans perceive and function in a 3D physical world around them; having data presented in the same way creates more of a familiarity between the user and the data than there would be if the user is not in a VR environment. In the VR environment they are then able to move (in the virtual world) through the data in a similar, more intuitive, way that they would in the real world. Humans already have a useful pattern recognition process (Donalek *et al.*, 2015), and VR allowing for better pattern recognition, enables the user to then obtain more useful insight when they are analysing data. This, therefore, allows better decision-making in comparison to conventional methods, like using a PC monitor for visualisation.

- The current VR technology developments allow for a user to visualise data with a high level of detail. Data being presented with a high level of detail allows a user to go nearer to the data points without losing visual quality. Along with users being able to view the data close up, they are also able to visualise data in its entirety by viewing it from a distance. A high level of detail refers to the detail of the visualisation and not to the granularity of the data to be visualised.

- Presenting data in VR allows for the data to be visualised more realistically in terms of the distances between data points as well as the relations between data points. This more realistic and accurate visualisation will bring the perception of users closer to reality than with the use of conventional methods. This allows the user to better understand the relations within the data which aids the first mentioned opportunity.

- VR allows for collaboration during the data visualisation process. Multiple users are able to visualise and interact with the same data together. This collaboration could be in the same VR environment where users are able to use multiple VR systems to enter the same environment or they can be in different VR environments, using different VR system, but visualising the same data.

- VR aids the data visualisation process as it enables users to draw conclusions, and complete desired data visualisation tasks in a shorter time than compared to conventional methods (Filho *et al.*, 2018). VR has also been shown to decrease the time needed during verification and validation processes (Akpan & Shanker, 2019). Along with reducing the amount of time taken to complete tasks, the use of VR also results in less errors being made by a user (Raja *et al.*, 2004).

3.2 Integration Opportunities

Along with some of the data visualisation opportunities with the use of VR, outlined in the previous section, there are several opportunities for the integration of VR and DTs:

- DTs could potentially include spatial Computer-Aided Design (CAD) models of the physical system they are “twinning”. VR allows users to have an immersive interaction with the DT and have a visualisation that is a more accurate representation of the physical system. This provides users with a better sense of the spatial mapping of a physical system represented by a DTs.

- The use of VR to visualise DT data has the possibility of better converting the DT data to information by supplementing the data with more context. This contextualisation refers to the possibility of being able to overlay a spatial representation, as mentioned above, with information that is non-spatial. An example of this contextualisation of non-spatial information is the colour of a motor in a virtual model can be changed to draw the user’s attention to it if the temperature of the motor exceeds a certain limit.

- DTs usually have the inherent functionality to update their models in near real time. This integration with VR will, thus, have the added benefit of updating these virtual models in near real time and allow users to monitor changes with the enhanced perception provided by VR. VR also has the ability to allow for multidimensional data to be visualised, annotated and changed with this near real time updating.

- VR has the potential to aid the design phase of a DT by allowing for accurate and easier visualisation of the physical system. This would reduce design time as well as costs (Havard *et al.*, 2019; Kovar *et al.*, 2017; Sekaran *et al.*, 2021). This visualisation of the physical system allows for DTs to be developed in parallel with or prior to the construction of the physical system.

- VR facilitates a virtual more "hands-on" interaction with a DT because a user, as mentioned in the data visualisation opportunities, is able to intuitively interact with a DT's data and make good decisions based on what they have visualised.

- Another opportunity is in the standardisation of the integration procedures between DTs and VR. This standardising will enable a better synergy between DT developers and VR developers. These two teams will have better means of collaborating with each other to achieve the desired needs of the DT system and presenting the information to a user using VR.

3.3 Data Visualisation Challenges

Along with opportunities associated with using VR to visualise data, a number of challenges are also present:

- VR has the risk of information overloading (Erra *et al.*, 2019) because, when large amounts of data is presented to a user, they could become overwhelmed. This large amount of data being presented to a user could negatively impact their ability to make decisions (Sekaran *et al.*, 2021).

- Data navigation is another challenge with using VR to visualise data (Filho *et al.*, 2018; Gracia *et al.*, 2016). Unlike with navigating data displayed on 2D monitors, there is no wide consensus about data navigation user interfaces (UIs) in VR. As such, users may find it challenging to navigate through the data as they desire and this could affect the visualisation process.

- Developing a VR environment presents the challenge of how to best present the data to a user so that they are able to most effectively interpret

the data. Developing a VR environment could also require more time and effort to develop compared to creating visualisations using conventional methods (Akpan & Shanker, 2019).

- Perspective distortion and occlusion are some visual challenges when wanting to visualise data in VR (Filho *et al.*, 2018; Gracia *et al.*, 2016). Occlusion is where an object, in a 3D space, is, from the user's perspective, hidden behind another object and thus not visible. In VR it is possible that a user's perspective is distorted due to the fisheye lenses that are used in a VR HMD to give the immersive feel of VR. This distortion, however, most likely only affects the pixels closer to the outer edge of the lens.

- Drouhard *et al.* (2015) mentions that other challenges include the safety and comfort of a VR system user. VR system users could experience a number of symptoms including motion sickness, disorientation, nausea, sweating, and headaches when using the system for prolonged periods of time (Liagkou *et al.*, 2019).

- Another challenge is the VR system cost. VR systems are computationally expensive and, therefore, require relatively expensive computer processing hardware as it is still a technology that is only recently becoming more easily available. However, it is expected that a larger usage of VR will result in lower hardware costs.

3.4 Integration Challenges

There are several challenges with the integration of DT and VR technologies:

- A challenge, much like a data visualisation challenge mentioned above, is to determine what DT information is to be displayed to the user in VR. There are a number of factors associated with this challenges, namely: the variety and amount of data, the possibility of information overloading, and the lack of experience industry members have had with using VR systems.

- DTs and VR are currently technologies that are developing at relatively fast rates. These "moving targets" make it challenging to create a stable interface between these two technologies that will not be impacted as these technologies develop further.

- The integration of the two technologies also presents the challenges of computational power utilisation. As mentioned previously, VR is already computationally expensive, and the addition of a DT system will require even more computational power resulting in higher hardware costs.

3.5 Discussion

It is evident that there are many opportunities and challenges associated with the use of VR as a data visualisation tool, and the integration of VR with DT technology. It must be noted, again, that the list of opportunities and challenges is not exhaustive and that there is a possibility that there are other opportunities and challenges that have not been mentioned here.

The opportunities and challenges mentioned for using VR as a data visualisation tool are also applicable in a system created by the integration of VR and DTs. These VR data visualisation opportunities and challenges are not meant to only be viewed in isolation, but also for an integration between the two technologies. It might not be possible that all opportunities or challenges will be realised in a single VR and DT implementation, and that different implementations could encounter their own challenges and opportunities. The above discussion looks at VR and DT systems in a general sense, but each specific system will require its own research to be conducted on the possible opportunities and challenges encountered in the system.

Although there are both challenges and opportunities, the potential opportunities will often outweigh the potential challenges associated with the creation of a general VR and DT system. These opportunities and challenges must be evaluated in practice to determine the effectiveness of using VR and DTs together.

Therefore, a system implementation is required to realise these potential opportunities, as well as identify any encountered challenges. The proposed implementation will be a DT system that makes use of VR to visualise the information obtained by the DT system for the decision-making process. It will be beneficial to compare this VR and DT system implementation to an implementation that does not make use of DTs. After comparing these two implementations, a more supported and credible conclusion can be made with regards to the integration of VR and DTs.

4 Facilities Management Division Case Study

This chapter describes the context of the case study that is used to evaluate the integration of DTs and VR, i.e. using the Facilities Management Division at Stellenbosch University. The case study description is first provided, followed by the system architecture design process.

4.1 Case Study Description

The case study consists of two implementations: The one implementation is a system that contains and integrates VR and DTs (the *DT implementation)* and the other implementation (the *Non-DT implementation*) does not make use of DTs for importing information into a VR environment for data visualisation. In the two implementations, the VR environment and the presentation of the information to the user is identical. The aspect where the two implementations differ is with regards to how the information is retrieved for the VR environment.

The Facilities Management (FM) Division at Stellenbosch University is responsible for overseeing the operational management of the various facilities that form part of Stellenbosch University. These facilities include faculty buildings, university residences, and private housing owned by the university for the purpose of student accommodation. Stellenbosch University has five different campuses: the main campus in Stellenbosch, the medical and health sciences campus in Tygerberg, the military science campus in Saldanha, the business school campus in Bellville, and another medical and health science campus in Worcester.

FM is responsible for overseeing the many facilities at each of these campuses. Managing such a multitude of facilities involves a complex system containing many different elements (in this case, facilities). The overseeing of operational functions includes facility utility usage monitoring, maintenance management tasks, and campus construction planning among others.

It is evident that such aspects will result in the vast amount of data being recorded with the purpose of decision-making. The visualisation of this vast amount of data could be challenging for the members of FM. This creates an opportunity where a VR and DT system could ease this data driven decision-making process.

A DT system could be used to aid the data acquisition, ordering, and storing processes that are required for FM. The VR system is then useful as the visualisation medium for the purpose of data visualisation. These two technologies would be used collaboratively to provide the user with the most optimal solution to ease the already complex data driven decision-making process for FM.

4.2 System Architecture Design

A VR and DT system is a suitable system for FM. This system first needs to be designed before it can be implemented and evaluated. This section describes the process used to design the architecture of the system. This design process uses the design framework developed Human (2022) that is discussed in Section 2.2 and illustrated in Figure 2.

4.2.1 User Needs

The first step in the system design process is to determine what the user's needs/requirements are for the system. For a system that uses VR and DTs, there are several stakeholders, but the following three types of users are identified:

- End user: This user enters the VR application to visualise information obtained by the DTs in the system.

- Configurator: This user configures the VR application or DT system for its use case, without changing any source code. A VR configurator determines what elements or information can be accessed by an end user in the VR application, but is not responsible for creating the elements needed to transfer or manage the information in the VR application. A DT configurator is responsible for ensuring that the DT system is configured correctly with various system components, but is not responsible for developing the various DT components.

- VR developer and DT developer: These users are responsible for developing the VR application and DT system, respectively. The DT developers focus on creating the DT system elements allowing for the integration of the DT system with VR. The VR developer are required to develop the VR application that is able to make requests to the DT system for information. The two developments are separate from each other but must be integrated with one another to achieve the end user's needs.

The system is designed considering the different requirements for the identified stakeholders. The point of interest is how information is transferred to the VR application using the different implementation methods. However, if both implementations produce the same outputs for the same input, the end user does not need knowledge of how this information is retrieved for use in the VR application. Therefore, the requirements of the end users, referred to now as the user, are of a higher priority in the design process of this system than the other stakeholders' requirements.

After some consultation with FM, the user requirements (URs) presented in Table 1 were identified. Although FM is responsible for a number of functions, only the monitoring of energy usage information for the main Stellenbosch campus is selected as focus here. The reason for this is provided in Section 1.4.

Table 1: User requirements

UR ID	Description
UR1	Visualise energy usage information for facilities within Stellenbosch University.
UR2	User must be able to select what energy data to visualise.
UR3	Have access to all university facilities' information from a central point.
UR4	Information available must be specific for a facility or group of facilities.
UR5	Allow for addition, removal, or modification of elements in the physical system (e.g. adding/removing buildings).
UR6	Allow for system functions to be added to in the future if desired.

4.2.2 System Requirements

From the URs, the system requirements must be identified. These system requirements are separated into system functional requirements (FRs) and system non-functional requirements (NFRs). FRs are the functions that a system must perform to achieve the user's requirements, while NFRs describe how, or how well, the system must perform the FRs (Human, 2022).

4.2.2.1 Functional Requirements

The system FRs that are derived from the above URs are provided in Table 2. These FRs are deemed to be the most fundamental functions that are used for a number of other functions within the system.

Table 2: Functional requirements

FR ID	High-level Requirement	Description
FR1	VR visualisation	A VR environment is required to visualise the energy usage information for the different facilities within the university.
FR2	Remote monitoring	Users should be able to visualise any energy information in the VR without needing to be at the facility.

Table 2 (continued): Functional requirements

FR ID	High-level Requirement	Description
FR3	Exploratory analysis	A user should be able to visualise and analyse different facility information such as energy usage trends, or energy usage comparisons between facilities.
FR4	Derived information	To support some decisions, the user should be able to visualise information that is derived from sensor data (e.g. trends) and, potentially, analyses of the physical system (e.g. carbon footprint).

4.2.2.2 Non-functional Requirements

The NFRs most applicable to an FM context are provided in Table 3. The rationale for the NFRs in the FM context is also provided. The NFRs identified in Table 3 form part of various NFR groupings identified by Human (2022). These NFR groupings are used to identify the possible implications of the NFRs for the system. These implications result in suggestions to aid the design of the system architecture.

For the identified NFRs in Table 3, four system architecture suggestions are made based on the NFR groupings. The first is the use of an aggregation hierarchy in the system. The remaining three are recommended design patterns that are used to design the architecture. Design patterns are discussed in Section 2.2. The design patterns used in the architecture design based on the identified NFR and subsequent NFR groupings are the performance efficiency, compatibility, and maintainability design patterns in Human (2022).

Table 3: Non-functional requirements

NFR ID	NFR	Rationale for NFR
NFR1	Allow for retrofitting of new and existing technology and information systems	A FM division must handle different maturity levels for various technologies. There is a possibility that many of the facilities still make use of legacy systems which must be integrated or “retrofitted” with newer technologies. The implemented system should not interfere with the current system.

Table 3 (continued): Non-functional requirements

NFR ID	NFR	Rationale for NFR
NFR2	Allow for efficient system reconfiguration	Facilities management can be viewed as a fairly static environment with not many aspects changing in short periods of time. However, if newer technologies, such as new energy meters or water sensors, are to be introduced, no matter how often, it is still an important requirement that the system is able to reconfigure efficiently. This NFR also includes the possible introduction of other additional services to the system requested by the user.
NFR3	Provide a fault tolerant system	The system is dependent on receiving readings from various meters over a wide geographical area. It is desired that if some of these meters become faulty and do not return readings, that the other aspects of the system are not affected, and the system is still able to function as required.
NFR4	Facilitate heterogeneous data handling	The facilities within a university will record large amounts of data and it is a highly likely that this data will be of different types. It is, therefore, necessary to ensure that this heterogeneous data is handled in a manner that the heterogeneity of the data will not negatively impact the data driven decision-making process.
NFR5	Provide for large amounts of data	A FM division will record large amounts of different data, but the system must still be able to function with low latencies.
NFR6	Avoid physical resource contention amongst software components	The system might be hosted on a single machine that will be responsible for hosting all elements of the system, including the VR application. Therefore, resource contention could potentially become a concern for the system and should be avoided.

4.2.3 Physical System Decomposition

This section considers the decomposition of the physical system for the FM context. Physical system decomposition is part of the definition of the *Problem space*, as illustrated in Figure 2. The section is divided into two parts, namely, a hierarchically decomposed physical system diagram and the system data characterisation.

4.2.3.1 Physical System Diagram

Within this section, the various physical system components present within a FM division's supervision are identified leading to a hierarchically decomposed physical system diagram. As mentioned in Section 1.4, the focus here is on energy usage data.

The university contains the following components:

- University – The main component of the system and is the highest level of the hierarchy. All other components are contained within the university.
- Campuses – The university comprises of multiple campuses with each campus being comprised of the following components.
- Precincts – A campus is comprised of multiple precincts. A precinct, in this thesis, is considered to be a geographical area in a campus that contains various system elements within that area. A precinct can also have a designated energy meter that records the overall energy usage for that precinct.
- Buildings – A precinct contains buildings that, in this thesis, are considered to be facilities being supervised by FM. Buildings contain various energy meters that are used to record energy usages within a building.
- Energy meter network – Precincts also contain energy meter networks. Some energy meters do not form part of a building but are still within a precinct area. These meters still represent some part of a precinct, and are, therefore, included as a system component.
- Energy meters – Energy meters are a type of utility usage measuring device that is used to record the electrical energy usage for a specified area within a precinct, building, or energy meter network.

The list above contains the various physical system components that form part of the physical system decomposition. Using these system components, a hierarchical physical decomposition diagram (Figure 8**)** is created.

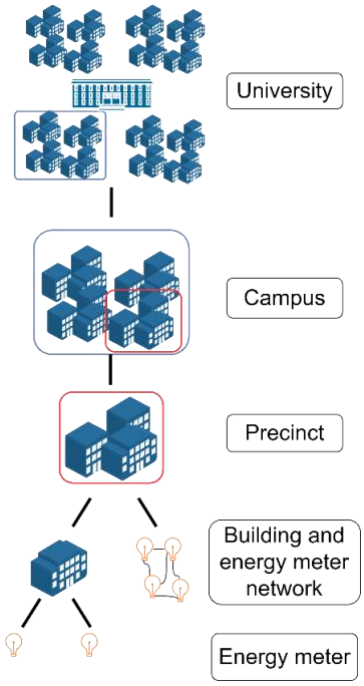


Figure 8: Physical system decomposition

4.2.3.2 Data Characterisation

This section uses Figure 8 to obtain the data characterisation information for the different components within the system. The energy meter is the only system component where data is received from the physical world. The architecture assumes that all energy meters are IoT devices that transmit their readings periodically, typically at five-minute intervals, to a service provider's servers. An application programming interface (API) is used to access the readings and timestamps through the internet. The other system components only receive information that is derived based on this energy meter data.

4.2.4 Service Identification and Allocation

This section is the start of defining the S*olution space* (as shown in Figure 2) for the identified FM system. Various services are identified in this section and allocated to either DTs or the Services Network, following the process detailed by Human (2022).

4.2.4.1 Services Identification

The following services are required to achieve the system requirements:

Mirror service

The Mirror service allows for the energy usage of various components (buildings, energy networks, precincts, and campuses) within, and including, the university to be presented to the user. The user must be able to request specific information for a component(s) and the service will be responsible for retrieving that information. The Mirror service fulfils the remote monitoring requirement (FR2).

For the building and energy meter network components the energy data for an energy meter is received using the API. This energy meter data is aggregated with data from other energy meters that form part of the component. The energy data for an energy meter is typically received at five-minute intervals, after which, it is then aggregated. For the precinct, campus, and university components energy information is received from the components lower in the physical system hierarchy. Similarly, this energy information received from a lower component is aggregated with information from other lower components. This frequency of information is dependent on the frequency of data received by the energy meters.

This service is invoked periodically as requests are made by the user. A consideration for the implementation of this service is that the number of components in the university could result in a data communication bottleneck depending on how much information is requested by the user.

Context generation service

The Context generation service must generate new information once new data is received by an energy meter. This generated information is derived from the energy data and is specific to the needs of FM. For example, FM require the average daily energy usage for a facility. This average usage is derived from the raw energy data recorded by an energy meter. This service fulfils the requirement for derived information (FR4).

The components in the system all require this service as components at different levels require for different information to be generated. For example, the information generated for a precinct is different to that generated for a building. For this service to be carried out for the building and energy meter network components, the data that is required is the energy data provided by the energy meters. This service at a precinct, campus, or university component level requires the aggregated energy information from the components lower in the physical system hierarchy.

This service is used as new energy data is recorded. If no new energy data is available, the service will not operate, and no information will need to be generated. The service need not be invoked by a user, but is constantly requesting for new energy data.

Exploratory analytics service

The Exploratory analytics service allows the user to compare information from various components within the system to identify any trends with regards to energy usages. The user is able to select what information they would like to compare from the components in the system. This service fulfils the requirement for exploratory analysis (FR3).

The service requires access to be able to make requests to all system components or to a component that can make requests to other components. For example, the university component has access to other system components, either directly or indirectly. This access ensures the service is able to compare the energy usage information for any component. The information required in the service is the derived information generated using the Context generation service.

The service is invoked when requests are made by the user. The service requires access to a network to access the requested information. Similarly to the Mirror service, the number of components in the university could result in a data communication bottleneck depending on how much information is requested by the user. A suitable communication method is required to transfer or collect the information. Alternatively, an information limit could be introduced allowing the user to view segments of the data if the amount of data requested is above a specified limit.

4.2.4.2 Digital Twin Identification

This section gives the DTs that are identified based on the physical system components in Section 4.2.3. The system components that are best suited for a DT implementation is given in Table 4 with the DT types, as well as the rationale. The energy meters are not included as DTs in the system as an individual meter will provide little useful information for FM as compared to a group of meters that form part of a building or energy meter network.

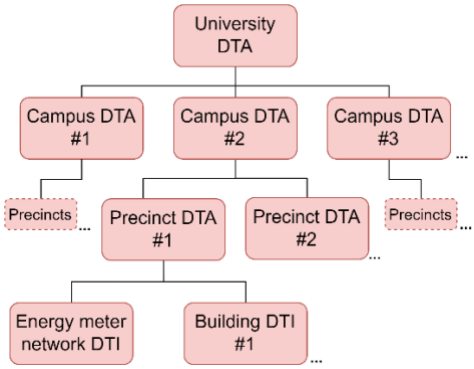


Figure 9: Aggregation hierarchy

The aggregation hierarchy of the DTs identified in the system is illustrated in Figure 9, with the DTIs at the lowest level and information being aggregated upwards in the aggregation hierarchy.

Table 4: System digital twins

Digital Twin	Digital Twin Type	Rationale
Building	DTI	A building and energy meter network are the lowest level of the DT hierarchy because the case study focuses on the overall functioning of a facility, and not the functioning of specific meters within the facility as mentioned in Section 1.4.
Energy meter network		
Precinct	DTA	The FM division would find it useful to analyse information for a specific precinct, campus, or university and not only the individual components that make up a precinct, campus, or university.
Campus		
University		

4.2.4.3 Services to Digital Twin Allocation

Table 5 shows the potential allocation of the identified services to either the identified DTs or the Services Network.

The Mirror service and Context generation service are required for each DT because the information received by the energy usage meters must be used to derive new information for a DT. The Mirror service will not provide the user with the raw data received by the DT. Rather, it will provide the user with the derived information based on the received raw data. The DTIs will receive raw data from the meters, while the DTAs will receive derived information from DTIs or other DTAs, using the Mirror service, below it in the hierarchy. The DTs will use this derived information to derive new information for the DTA. It is possible to allocate these two services to the Services Network; however, because they are persistent services and require the raw or generated data obtained by the DTs, it is more logical to host them in each DT.

The University DTA can host the Exploratory analytics service as it has access to all the information through its connection with the DTs below it in the hierarchy. However, as this is a periodically requested service and is a more general service and not specific to a DT, the service will reside in the Services Network and only be invoked when the user makes a request. The Services Network can also make use of external services or databases that are not part of the system. The Services Network may require these external elements to fulfil desired requests, and provision for this has been made.

Table 5: Potential services hosts

Service	Building DTI	Energy Meter Network DTI	Precinct DTA	Campus DTA	University DTA	Services Network
Mirror service	X	X	X	X	X	X
Context generation service	X	X	X	X	X	X
Exploratory analytics service					X	X

4.2.5 Digital Twin Internal Architecture and Design Pattern Application

The internal architecture to be used for the DTs in the system, SLADTA, was determined in Section 1.2 as an objective for the thesis. SLADTA is discussed in greater detail in Section 2.3.

Figure 10 illustrates the internal structure of the DTs in the system. The energy meters' data is already stored and accessed using an API, as mentioned in the Data Characterisation section (4.2.3.2). Therefore, the relevant DTs need not implement a Layer 1 or Layer 2. These layers are, however, still illustrated to indicate that they have been implemented in some form but not in the scope of this system. Layer 4 is responsible for data acquisition using the provided API for the relevant energy meter. The API therefore fulfils the role of Layer 3 in this implementation. Only the DTIs and Precinct DTAs contain a Layer 3 because only they have associated meters and need to request information from the API.

Having selected the internal DT architecture, the design patterns formulated by Human (2022) can be selected and applied to the system components. The three design patterns required for the design of the system are for maintainability, compatibility, and performance efficiency as identified in Section 4.2.2.2. These design patterns are used, in combination, to design the implementation architecture for the system.

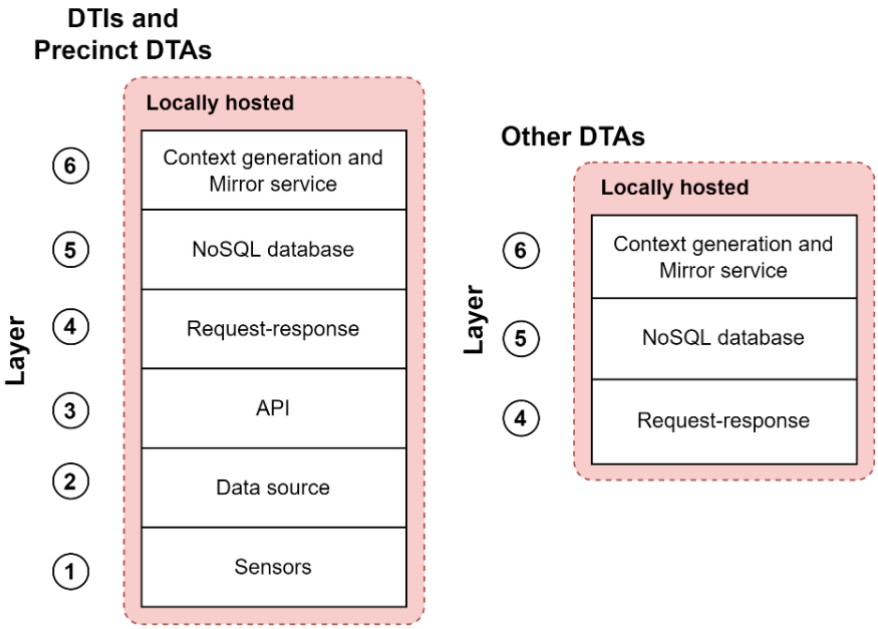


Figure 10: DT internal architectures

The system makes use of pre-storage aggregation as is recommended by the compatibility and performance efficiency design patterns. Therefore, the data that is obtained by Layer 3, using the API, is first aggregated and then stored for the relevant DTs (i.e. Buildings, Energy meter networks, and Precincts). Pre-storage aggregation is also implemented for the other DTs in the system. This pre-storage aggregation is a local aggregation (as opposed to cloud-based) as per the performance efficiency design pattern. Layer 4 uses a request-response communication between DTs to transfer information required for aggregation as recommended by the performance efficient design pattern.

Layer 5 makes use of a local NoSQL database to store the aggregated DT information as recommended in the performance efficiency and compatibility design patterns. The reason for this choice is that a NoSQL database is more scalable, can handle heterogeneous data, supports high throughput, and has a lower latency than SQL databases (Human, 2022). A VR application for data visualisation purposes requires a quick responsiveness to adapt to the information requests by the user. The locally hosted databases aid in this regard due to the low latency they offer.

The Mirror and Context generation services are hosted locally in Layer 6 for the DT. The reason for local hosting is that the system requires a low latency for the high throughput of information that is to be transferred between system components. The system components are all connected to the same network. Therefore, the system can make use of a local network communication protocol,

such as TCP/IP, for communicating between the various components. A local communication protocol also allows for lower latencies and high throughputs compared to cloud communication protocols.

Figure 11, which illustrates the overall system architecture, shows the internal architecture of the Shared Services component. The Service Network hosts the Exploratory analytics service as mentioned in Section 4.2.4.3. The Shared Services component also contains a Service Gateway and DT directory service (which forms part of the Management services). A central user interface (CUI) is advised in the maintainability and compatibility design patterns. The CUI, in this implementation is the UI in the VR application that is used to request information from the DT system.

Figure 11 illustrates how all of the system components are in communication with one another. The “VR application” aspect of Figure 11 is a custom designed application. The design of this application is implementation specific, and the internal architecture of the application is not shown in this diagram. The “Aggregation hierarchy” is the hierarchy shown in Figure 9. It must be noted that DTs are able to access external databases, as indicated in Figure 11, that a DT may require to be able to carry out a service.

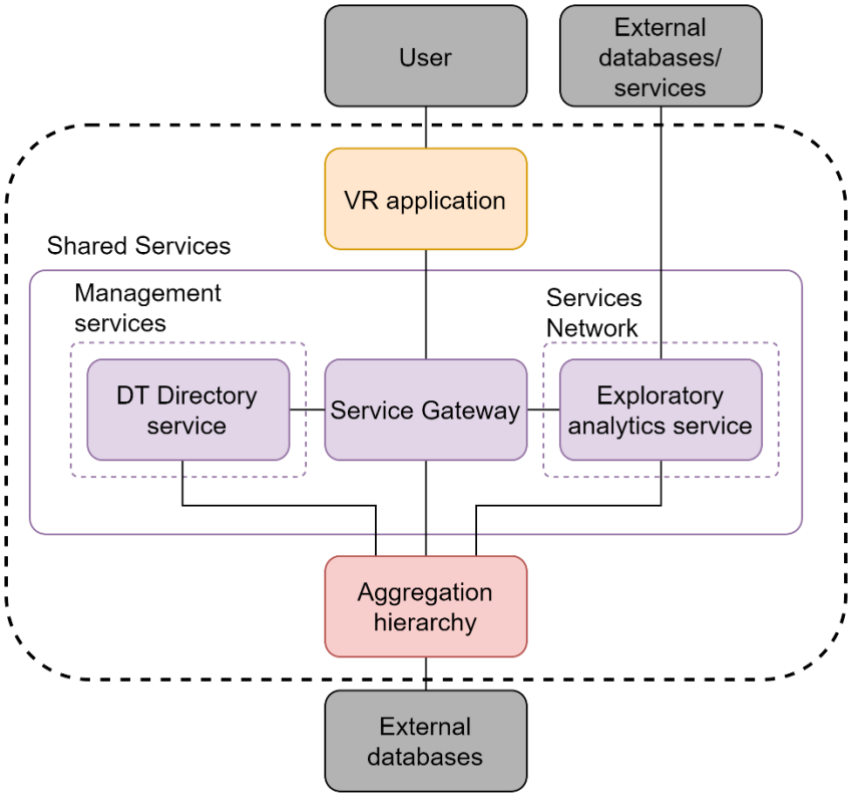


Figure 11: Overall system architecture

5 Case Study Implementations

This chapter discusses the details of the two implementations used in the case study. The objectives of the case study are first discussed to determine how the implementations will be used to aid the research. The *Non-DT implementation* details are then discussed, followed by the *DT implementation*. The VR applications used in both implementations are then discussed and the chapter concludes with a comparison of the implementations.

5.1 Case Study Objectives

The main objective of the case study is to present two implementations, a *Non-DT implementation* and a *DT implementation*, that are compared to one another to determine the value of DTs for VR data visualisation purposes. The two implementations use different methods for transferring information into VR for a user to visualise, but are intended to provide the same output. The case study, therefore, highlights any differences or similarities in the implementation methods. These differences or similarities are then used, in conjunction with evaluations, to determine whether DTs are beneficial for use in a system that uses VR.

As mentioned in Section 1.4, the implementations make use of the available energy meter data from FM. The implementations also focus on the main campus in Stellenbosch and not on the whole of Stellenbosch University with the various campuses. The implementations also do not include the energy meter network component and only focus on providing energy usage information for the buildings and precincts of the campus. A typical use case entails that a user is able to request and visualise energy information for the various components in the campus using VR.

5.2 Non-DT Implementation

This section discusses the *Non-DT implementation*, starting with the implementation architecture and followed by the implementation details.

5.2.1 Implementation Architecture

The development of the *Non-DT implementation* VR application first requires an implementation architecture to be developed. There is currently no consensus in literature on how this architecture should be structured. Therefore, the architecture is designed here specifically for this implementation case study. In this implementation, all the functionalities required for visualising data obtained from the external data source is carried out within the developed VR application.

The system does not make use of independently operating components, in contrast to the *DT implementation*.

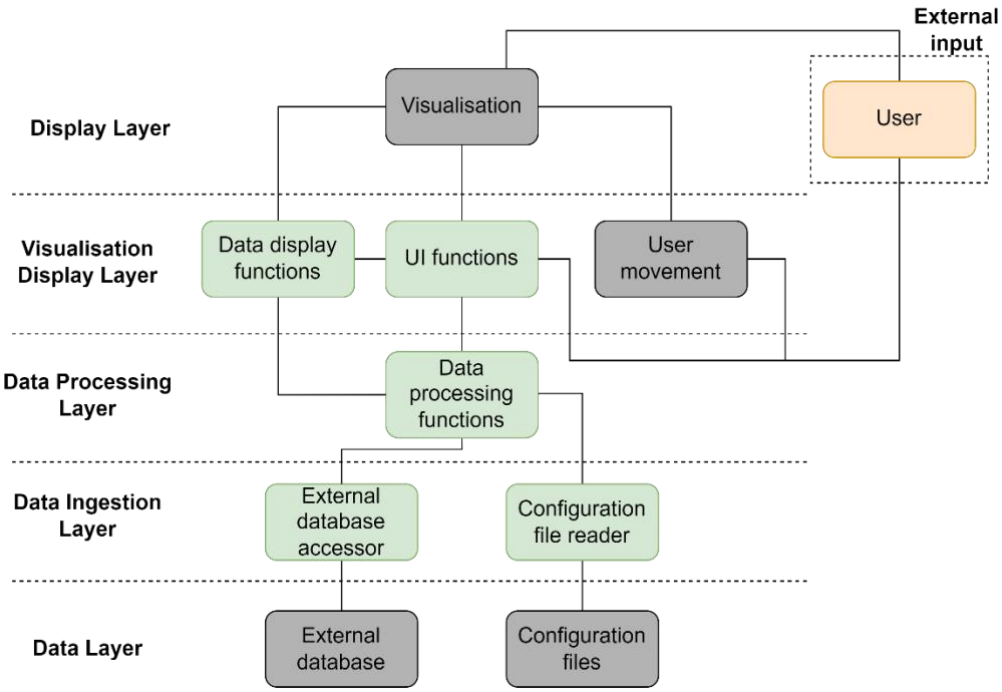


Figure 12: Non-DT implementation architecture

Figure 12 shows that the selected implementation architecture consists of five layers, with each layer being responsible for certain functions for the VR application. The layers are the Display Layer, the Visualisation Display Layer, the Data Processing Layer, the Data Ingestion Layer, and the Data Layer.

The Display Layer contains the visualisation that the VR application user is immersed into and can interact with. This layer receives information from the layers below it to display the correct VR environment to the user. The Visualisation Display Layer contains various functionalities that are used to create the correct VR environment for the user. This layer receives information from the Data Processing Layer and input from the user to update the visualisation as required. A user is able to make a request to visualise desired energy usage information in the VR application. This request is sent from the Visualisation Display Layer to the Data Processing layer. The Data Processing Layer is responsible for requesting and receiving information from the Data Ingestion Layer and performing any necessary calculations and processing of the data for use in the visualisation. Information is requested from the Data Ingestion Layer by the Data Processing Layer based on the request received from the user.

The Data Ingestion Layer contains the functionality to access the external database or to access the system specific data stored in the system configuration file. System specific data is contained in the configuration file that indicates which elements are present in the system, such as buildings, precincts, and the campus. A reason for having a specific layer for data ingestion is for the case where data for a new data type is to be added to the system. The functions in this layer will need to be changed to be able to make provision for this new data type. Having this separate Data Ingestion Layer with designated functionality will reduce the complexity to reconfigure this layer without needing to reconfigure functions that are part of other layers if there was no specific Data Ingestion Layer.

The Data Ingestion Layer uses the request from the Data Processing Layer to obtain the necessary data from the Data Layer. The Data Layer is where any information available for visualisation, or system specific data, is stored. The Data Layer receives a request from the Data Processing Layer, then responds with the desired information. The information is then sent to the Visualisation Display Layer, and then to the Display Layer where the user can visualise the requested information.

A VR application for the purposes of data visualisation can have two methods for accessing information. The first is by storing all the data in the VR application during initialisation. The data can then be accessed within the application. The second is by using an external database to access the information as requested. This second method does not require the data to be stored in the VR application as the data is available in an external database. Kroupa *et al.* (2018) makes use of the external database method in their implementation. On request, the external database is accessed using a client/server connection.

Both methods provide the same outcome and the decision as to which method is implemented is dependent on the case study. In FM, the raw energy usage data is accessed using an API. As the information has already been stored and can be accessed externally, the *Non-DT implementation* architecture, therefore, makes use of the external database method for the implementation.

5.2.2 Implementation Details

This section provides the details regarding the implementation of the *Non-DT implementation* using the architecture in Figure 12. The architecture is implemented using Unity (which is discussed in Section 2.4.3). Unity is selected due to its popularity, reliability, and support. Unity also has VR functionalities that allow for a VR application to be created fairly easily.

5.2.2.1 Data Layer

The configuration file in the system is a comma-separated values (CSV) file and contains information about the hierarchy of various elements (buildings and precincts) in the campus. This information includes the name of the element, the type of element, the physical coordinates of the element, and the relation of that element to the other elements in the hierarchy. A CSV file is used because FM uses a similar hierarchy CSV file in their operations. A sample of the configuration file used is provided in Appendix B.

As mentioned previously, the raw energy data is stored in an external database that is accessed using an API. The API requires a URL and provides energy data based on the specified meter ID and the time period of the data requested. Requests are made to the API when the user makes a request in the VR application to visualise specified energy information. The data is received from the API in JavaScript Object Notation (JSON) format.

5.2.2.2 Data Ingestion Layer

The Data Ingestion Layer consists of an External database accessor and a Configuration file reader. The External database accessor is a set of functions that are used to access the energy data using the API mentioned previously. These functions receive the requested energy data and transfer the data to the Data Processing Layer. The Configuration file reader is responsible for obtaining system information from the system configuration file. This information can then be used by other system elements. For example, the information from the configuration file is used to populate menu items, such as a list of buildings, in the UI.

5.2.2.3 Data Processing Layer

This layer contains various data processing functions that are used to convert the raw energy data, received from the Data Ingestion Layer, to information for visualisation. These functions include calculating the daily, monthly, or yearly average energy usage for a system element. As mentioned in Section 4.2.3.2, the energy meter data records new data typically at five-minute intervals. This five-minute interval energy data is processed to obtain the desired average energy usage information. A request can also be made for the latest energy usage for a system element. The Data Processing Layer then processes the energy meter data to obtain the latest energy usage for a system component.

The energy usage information for the building components is derived from the raw energy data for the energy meters in that building. The derived energy usage information for the buildings, in a precinct, is then used to derive the energy usage information for that precinct. Similarly, the derived energy usage for the precincts in the campus are used to derive the energy usage information for the campus. An

example of this process is when the daily average energy usage for the entire campus over a specified period of time is requested. This process is illustrated in Figure 13. The campus requires information from the precincts which requires information from the buildings that receive the energy data from the energy meters. The configuration file is used to determine how information is distributed amongst the various system components.

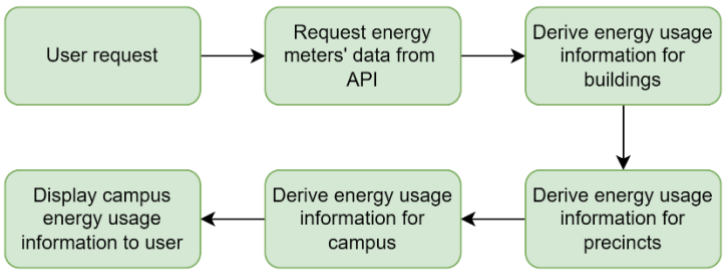


Figure 13: Requesting campus information process flow

Figure 13 shows that after a user makes a request, the relevant energy meter data is requested using the API. The data is processed to derive the energy usage for the buildings in the various precincts in the campus. The buildings' energy usage information is used to derive the various precincts' energy usage information that is then used to derive the campus' energy usage information. This campus energy usage information is then displayed to the user for visualisation. The user is then able to make another request after they have visualised the information.

5.2.2.4 Visualisation Display Layer

The Visualisation Display Layer consists of three aspects: Data display functions, UI functions, and User movement. These aspects together are used to ensure the correct VR environment is displayed to the user. The Data display functions receive the processed information from the Data Processing Layer. The information is then interpreted to populate the information in the VR environment for visualisation. The UI functions interpret the requests that are made by the user when interacting with the UI. The user's request is then transferred to the Data Processing Layer by the UI functions. The User movement functions are a combination of built-in functionality available in Unity and custom developed functions for the user to navigate the VR environment. Using VR controllers the user is able to navigate through the VR environment. The VR application adjusts the visualisation according to this inputted user movement. The populated information in the VR environment is not altered during this adjustment, rather only the user's viewing perspective is changed depending on their movement. These Visualisation Display Layer functions are similar to those for the *DT implementation* and are discussed in Section 5.4 because of this commonality between implementations.

5.2.2.5 Display layer

This layer, as mentioned previously, contains the actual VR environment that the user will be immersed in. This VR environment uses the discussed layers above to display the environment with the correct energy usage information to the user. Using the VR equipment discussed in Section 5.4.1 the user can interact with and visualise the information in this VR environment.

5.3 DT Implementation

This section presents the *DT implementation*. The implementation is also used to visualise energy usage information for the Stellenbosch Campus in VR, like the *Non-DT implementation*. The implementation architecture is first discussed, followed by details regarding the DT components, the Shared Services component, and, finally, the operation of the system.

5.3.1 Implementation Architecture

The architecture for the *DT implementation* is developed in Chapter 4, as shown in Figure 11, with the internal architecture of the DT components to be used in the implementation shown in Figure 10. The aggregation hierarchy of the implementation differs slightly from the hierarchy provided in Figure 9 in that the University DTA component and the energy meter network DTI have been omitted because they would not add much value to the case study.

The Shared Services component contains a Service Gateway, a DT Directory service, and the Exploratory analytics service, with the VR application being the CUI, as discussed in Section 4.2.5. The overall architecture for the *DT implementation* is shown in Figure 14. The VR application architecture is discussed in Section 5.4 as the details of the application are similar to the Visualisation Display Layer of the *Non-DT implementation* in Section 5.2.2.4.

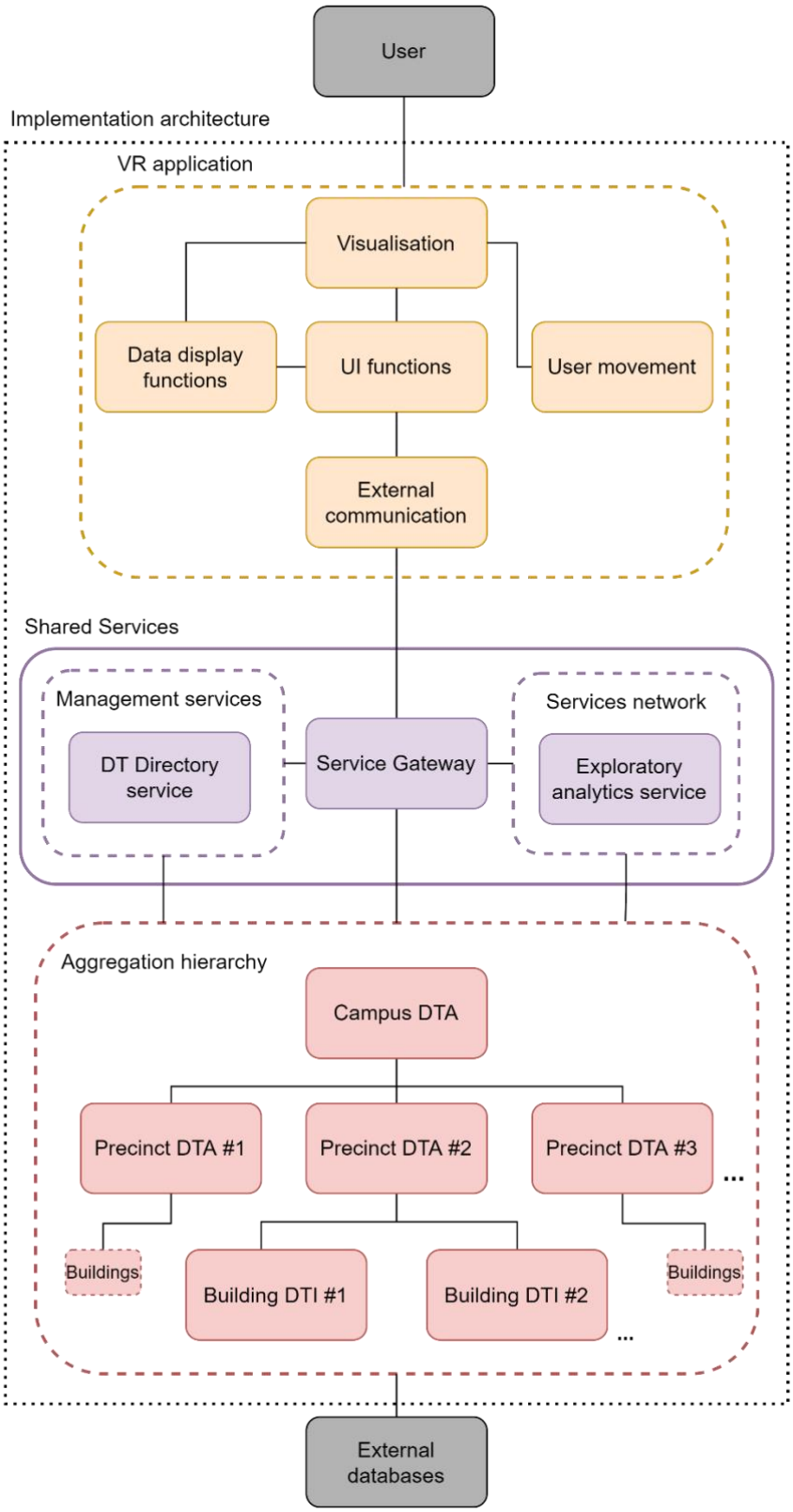


Figure 14: Overall DT implementation architecture

5.3.2 Digital Twin Components

The DT components are those of the aggregation hierarchy in Figure 14. The DT components are developed using the C# language as the VR application is developed in Unity that makes use of C#. The DT types (Building DTI, Precinct DTA, and Campus DTA) are similar, but there are differences in how each operates. The DTs consist of several aspects: the use of SLADTA for implementation, the communication and aggregation between DTs, and the services offered by the DTs.

5.3.2.1 SLADTA

SLADTA is selected as the internal architecture of the DTs in the system. As detailed in Section 2.3, SLADTA consists of six layers. Figure 10 indicates the details for each layer for a DTI or a DTA.

5.3.2.1.1 Layers 1, 2, and 3

Section 4.2.5 indicates that the energy usage data use in the implementation is already recorded, stored and is available through an API. The DTIs and Precinct DTAs in this implementation, therefore, have no Layers 1, 2, or 3. The API fulfils the role of Layer 3 in the implementation of the DTIs and Precinct DTAs, that is the same API used in the *Non-DT implementation* to access the raw energy data. The API requires a URL with specified API key, and other values like the desired meter ID, and the time period of the requested data. The API responds with the desired data in JSON format.

5.3.2.1.2 Layer 4

Layer 4 is a set of custom developed functions with the purpose of obtaining data from Layer 3, for DTIs, or information from other DTs, in the case of DTAs. This data or information is then processed and transferred to Layer 5 for long-term storage. As per the design selections made in Section 4.2.5, Layer 4 makes use of a request-response communication structure. When the functions for a DT's Layer 4 are to be carried out, a request is made to the necessary components for the desired information. The Layer 4 of each DT type is similar with slight differences in where the data or information is requested from.

Building DTIs

The Building DTIs are the lowest level of the aggregation hierarchy. Layer 4 of a Building DTI makes requests to the API of Layer 3 to retrieve energy data for all the energy meters in a building that, together, reflect the energy usage of the building. Layer 4 then aggregates the energy data before it is used by the Context

generation service. This pre-storage aggregation is a design decision made in Section 4.2.5.

Precinct DTAs

Layer 4 for a Precinct DTA is like that of a Building DTI, and also like that of a Campus DTA. A Precinct DTA has Building DTIs of buildings that form part of the precinct, but the precinct also has a main energy meter that is used to record the raw energy data for the precinct as a whole, like a building. As such, Layer 4 can make requests to the API in Layer 3 for the raw energy usage data for the precinct, like a Building DTI would, but because a precinct only contains one energy meter, it does not require for raw energy meter data to be aggregated like with the Building DTI.

In addition to obtaining data from an energy meter, the Precinct DTA requests energy usage information from the various Building DTIs within that Precinct DTA. If a Building DTI has the necessary information, the information is then transferred to the Precinct DTA and aggregated with the other Building DTI energy information before it is used by the Context generation service.

The aggregation of the Building DTI energy data ideally should coincide with the energy data from the precinct's energy meter. Two data sources providing data for the same component allows for a possible anomaly detection service to be implemented. This service is not implemented in this case study, but the two methods of data retrieval are mentioned to indicate that provision has been made for this type of service to be implemented.

Campus DTA

Layer 4 for a Campus DTA is like that of a Precinct DTA's Layer 4. The campus does not have a designated energy meter that provides energy data about the campus like a precinct. The Campus DTA must retrieve the energy meter information from the Precinct DTAs below it in the aggregation hierarchy. The energy information of the Precinct DTAs is aggregated to obtain the energy usage information for the campus. This aggregated energy information is then used by the Context generation service.

5.3.2.1.3 Layer 5

For Layer 5 a local NoSQL database is used for long-term storage as shown in Figure 10. The reason for the locally hosted database is for a low latency system; this is a recommendation of the performance efficiency design pattern in Section 4.2.5. MongoDB is selected to implement these local databases.

The raw energy data available from the API is not duplicated and stored in the local database of a DT because the API data is under the control of FM and remains available in the long-term. Therefore, only some aggregated information, like the latest energy usage, the information that may be required by the Context generation service, like the average energy usage, and other more static data are stored in the local databases

The static information stored for a DT in the database includes the name of the DT, the location coordinates of the DT, and the subordinate DTs or energy meters of the DT. A DT does not store any information about DTs higher up in the aggregation hierarchy as this does not form part of the span of reality of that DT. A DTA only stores information that is part of its span of reality, without duplicating information, like energy usage, for the span of reality of its subordinate DTIs or DTAs.

5.3.2.1.4 Layer 6

Layer 6 contains the services that are offered by the DTs in the system. For the case study, two services are offered by a DT, namely the Mirror service and the Context generation service, as mentioned in Section 4.2.4.

Mirror Service

The Mirror service provides the user with requested information about a DT or group of DTs. The service collects information from a DT or group of DTs based on the desired information that is requested by a user. A DT receives a message with what information is requested. For energy usage information, the implementation makes provision for requests for either the average energy usage or the latest energy usage. A DTA could also receive a request for the names of its subordinate DTIs and DTAs. The Mirror service is used to obtain this subordinate DT information.

The Mirror service for a DTA can call for the Mirror service of any of its subordinate DTs. An example of this request is illustrated in Figure 15 where a request is made to the Campus DTA for the latest energy usage of all components in the campus, including the buildings, precincts, and the campus itself. The numbered lines in the figure show the order of information flow in processing this type of request.

Once the request has been received by the Campus DTA, a request is made to the list of Precinct DTAs that form part of the campus for the same type of information. Each Precinct DTA makes requests to the Building DTIs that form part of the Precinct DTA for the same information. The Building DTIs respond to the Precinct DTA with the information. The Precinct DTA also retrieves its own latest energy usage information and, along with the information from the Building DTIs transfers

this information to the Campus DTA. This process occurs for all Precinct DTAs and Building DTIs. The Campus DTA then retrieves its own latest energy usage information from the NoSQL database. The reply to the external component contains the latest energy usage for the campus, all precincts, and all buildings in the DT system.

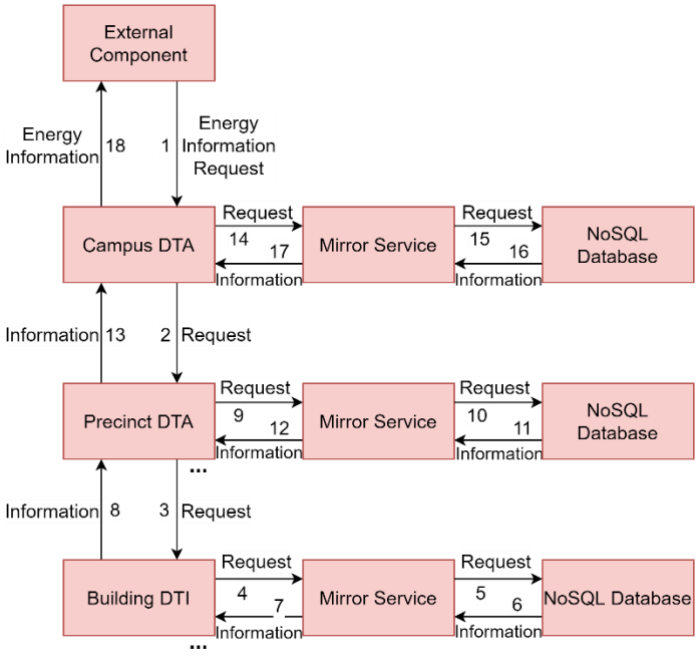


Figure 15: Mirror service example

Context Generation Service

The Context generation service, as described in Section 4.2.4, derives new energy information for a DT. For a Building DTI, information is derived from the raw energy data from energy meters, while for a Precinct DTA and Campus DTA, the information is derived based on information from subordinate DTs. Precinct DTAs, however, can also derive information from raw data from its single energy meter. Every DT offers the Context generation service.

Figure 16 illustrates this service being carried out when new energy data is available from the energy meters. The service is only invoked when new energy data becomes available for a DT. The functions in Layer 4 periodically make requests to the API in Layer 3 or other DTs, if applicable, to determine whether new energy data is available. If new data is available and has not been processed yet, the service uses this newly available data, as well as the already derived information in the NoSQL database for a DT to derive new information. This process begins at the Building DTI level, then moves to the Precinct DTA level, and finally to the Campus DTA level.

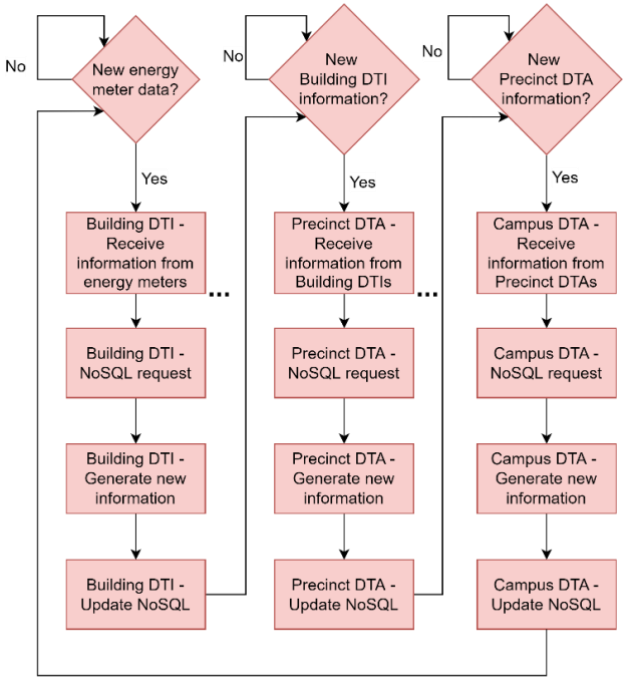


Figure 16: Context generation service process

The service is fully executed first for the Building DTIs, then the Precinct DTAs, and finally the Campus DTA. The service is not executed fully for a DT until all the subordinate components have new information available. The reason for this idle period is that the service requires the complete aggregation of subordinate data to correctly generate the energy usage information for a DT.

5.3.2.2 Communication

The DTs in the system are independently operating instances of the various developed DT type classes and require a communication mechanism between the DTs. Socket communication, specifically Transmission Control Protocol/Internet Protocol (TCP/IP), is selected for this communication. All system components, including the DTs, Shared Services, and the VR application are hosted on the same network, and TCP/IP offers a low latency form of communication between the various components. Each DT contains a TCP/IP server for receiving requests from other components, and all also contain a TCP/IP client for making requests to other system components. This communication uses JSON strings for the message payload.

The use of this socket communication presents an opportunity for a distributed hosting of system components. This allows for management of computational resource utilisation that could be a challenge of a DT and VR system, as mentioned in Section 3.4.

5.3.2.3 Digital Twin Component Operation

The operation of the DT system consists of two aspects: the initialisation of the DT component and the operational functions of the DT component. More detailed information for these two aspects and some of the associated complexities of the DT system is provided in Appendix A. Figure 14 provides general names for the various DTs in the system. More specific names for some of the system DTs are provided in Appendix B which contains a sample of the configuration file used.

5.3.3 Shared Services Component

The other aspect of the DT system is the Shared Services component. This aspect is responsible for offering services in the system that are not offered by DTs. The general architecture of the Shared Services component internal structure is shown in Figure 11 and, as implemented in the case study, in Figure 14. This internal structure indicates that the Shared Services component consists of a Service Gateway, Management Services, and a Services Network. The services in this Shared Services component are discussed below. The Shared Services component is also developed using C#. The Shared Services component is separate from the DT hierarchy and is developed irrespective of the DT aggregation hierarchy.

5.3.3.1 Service Gateway

The Service Gateway receives requests from a component, which for the case study relates DT information or to the energy usage information from the DTs, formatted as a JSON string. When a request is received by the Service Gateway, the Service Gateway decodes the JSON string, interprets the request, and transfers the request to the correct service component. This service component then carries out the service and responds with the correct information, and the Service Gateway transfers this information to the component that made the original request.

The Shared Services component makes use of TCP/IP socket communication like the DT aggregation hierarchy for communication between different system components to be possible. A TCP/IP server is hosted by the Service Gateway, allowing external requests to be received. The Service Gateway is then also able to use a TCP/IP client to make requests either to other service components (Management Services or Services Network) or to the DT aggregation hierarchy.

5.3.3.2 Management Services

The Management Services are services that are used to oversee the functioning of the system. In this implementation, the only management service used is a DT Directory service. The role of the DT Directory service is to contain information

about the DTs in the system, such as the names of the DTs, the communication addresses of the DTs, and the subordinate DTIs or DTAs for a DTA.

A request is made to the DT Directory service when information about a DT is required. Any service in the Services Network can make a request to the DT Directory service. Services offered by the DT Directory service are: providing the communication address for a specific DT and the structure of the aggregation hierarchy, i.e. the various DTs in the system and their subordinate DTs.

The DT Directory service is a single component that is responsible for storing information for the various DTs, as well as the structure of the aggregation hierarchy. Although this information can be obtained by making requests to the aggregation hierarchy directly, this method introduces complexities as a request will need to be made to every DT in the system every time this information is required. The DT Directory Service allows other services in the Shared Services component to have access to this information without needing to make requests to the aggregation hierarchy directly.

5.3.3.3 Services Network

The Services Network generally contains components that can offer various services for the system. In this implementation, the only service that is part of the Services Network is the Exploratory Analytics service discussed in Section 4.2.4.

The Exploratory analytics service receives a request from the Service Gateway for the desired energy information for specified DTs in the aggregation hierarchy. The service interprets this request to determine for which DT the request for information must be made to. A request is then made to the DT Directory Service for the communication address of the DT. Using the communication address received from the DT Directory service the Exploratory Analytics service then makes a request to the relevant DT to obtain the desired energy usage information. Once the energy information is received from the DT, the Exploratory Analytics service then responds to the original request from the Service Gateway with this energy information. The Exploratory Analytics service can interact directly with DTs in the aggregation hierarchy, and with other services in the Service Component.

5.3.3.4 Shared Services Component Operation

The operation of the DT system consists of two aspects: the initialisation of the Shared Services component, and the operational functions of the Shared Services component. More detailed information and some implementation software code for these aspects is provided in Appendix A.

5.3.4 System Operation

The DT system enters an operational stage after all the components, DT components and Shared Services components, have been initialised. In this implementation there are three operations that are carried out during this operational stage, i.e. the operation of the DT aggregation hierarchy, fulfilling the request for information about the DT aggregation hierarchy, and fulfilling the request for energy usage information.

5.3.4.1 Digital Twin Aggregation Hierarchy Operation

The operation for the DT aggregation hierarchy includes the updating and processing of new energy data that becomes available. When an energy meter records new energy data the Building DTI carries out the Context generation service for this new data. This propagates upwards in the aggregation hierarchy to the Precinct DTAs, and the Campus DTA. This operation for the aggregation hierarchy ensures that the latest energy usage information is available from the DT aggregation hierarchy. The operation of the DT aggregation hierarchy also includes the use of the Mirror service for a DT when a request is received. More information regarding these DT aggregation hierarchy operations is provided in Appendix A.

5.3.4.2 Digital Twin Aggregation Hierarchy Request

The DT system allows for information to be requested about the DT aggregation hierarchy with its DTs and how these DTs relate to one another. The handling of such a request is shown in Figure 17. This process is identical for all requests for information about the aggregation hierarchy. An example of this type of request is the user making a request to receive a list of the subordinate DTs that form part of the Campus DTA. The result of this request is a list of the names of the subordinate Precinct DTAs that are part of the Campus DTA. Figure 17 shows that handling this request for information is fairly simple as the information can be provided directly by a service in the Shared Services component, the DT Directory service, through interaction with the Service Gateway.

Figure 17: DT aggregation hierarchy structure request

5.3.4.3 Digital Twin Energy Information Request

The DT system also fulfils requests for the energy usage information for the DTs. An example of this request is a request for the latest energy usage for only the

campus, and buildings in the campus. In this example the information at the Precinct DTA level in the aggregation hierarchy has not been requested. The process of this request is illustrated in Figure 18. Information is requested from the aggregation hierarchy by using the Mirror service of the DTs that is shown in Figure 15. For the sake of brevity the full procedure for the Mirror service of the DTs is not shown here. Rather, just an indication of the different components that are part of the request for energy information is provided.

Figure 18 shows that although the precinct level information is not requested, the Precinct DTAs are still required to request information from their Building DTIs. Figure 18 shows how the DT aggregation hierarchy is used to obtain information from various DTs in the system. The result of this process is the latest energy usage for the campus as a whole, as well as the latest energy usage for all of the buildings in the campus.

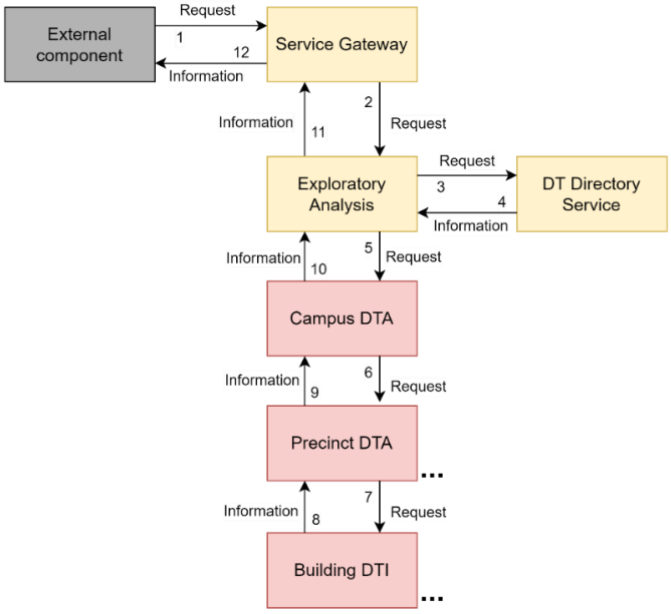


Figure 18: Exploratory analysis request example

5.4 Virtual Reality Application

The *Non-DT implementation* and the *DT implementation* each have their own VR application, but the functioning of the two VR applications, in terms of user interaction and displaying information, is similar. For this reason the VR application implementations are discussed together below. The VR application consists of four aspects: the hardware and software used, the virtual environment for the user experience, the user interaction with UI, and the displaying of

information to the user. The architectures used for the VR application implementations is shown in Figure 12 and Figure 14.

5.4.1 Hardware and Software

The VR hardware used in the implementation is the HTC Vive Pro (Figure 19), including an HMD, two VR controllers, and two infrared sensors. The HMD is used to display the developed VR environment to the user. The user uses the VR controllers to provide input to the VR application for the application then to adjust accordingly. The infrared sensors track the movement of the HMD and controllers to update the visualisation from the user's perspective through the HMD.

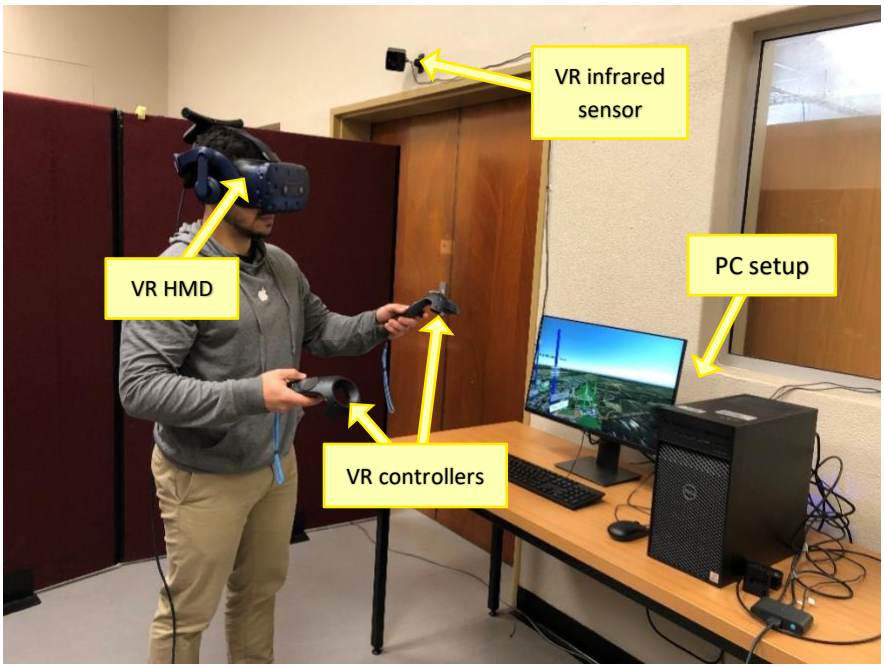


Figure 19: VR equipment setup

A desktop PC is used to develop the *DT implementation* components, the *Non-DT implementation* components, and the VR applications. The same PC is used to host the various components for the implementations including the developed VR application. The VR equipment is connected to this PC which allows the user to visualise the VR environment on the PC, through the HMD. The PC used has 64 GB of RAM, an Intel Core i7 10th Gen CPU, 1 TB SSD, and an NVIDIA GeForce RTX 2060 Super graphics card. The PC has high specifications as running a VR application is computationally intensive.

The Unity game engine, discussed in Section 2.4.3, is the selected software to develop the VR application. The Unity game engine makes use of the SteamVR software to allow the user to enter the VR environment and interact with it.

Appendix C.1 provides some additional information about the Unity environment that was used during development.

It should be noted that other software, such as ABB RobotStudio, is used in industry for the visualisation of 3D robotic systems in a virtual world and that such software could also be considered for this application. VR software provided by automation vendors was not used in this thesis because it is often expensive to maintain, vendor specific and not as expandable as software such as Unity.

5.4.2 Virtual Environment

Using Unity and its functionality, the user can be immersed in the VR environment and interact with it. The environment includes several elements to provide the user with a satisfactory experience when using the VR application to visualise energy information. These elements include the UI and the displaying of information to the user aspects which are discussed in Section 5.4.3 and 5.4.4, respectively. Other elements include the objects in the VR environment and the movement mechanics of the user.

The case study is focussed on visualising energy usage information for the Stellenbosch University campus located in Stellenbosch. Therefore, a major aspect of the virtual environment is a map of Stellenbosch that is displayed to the user. This map is shown in the visualisation examples provided in Section 5.4.4 and Appendix C.3.

An opportunity mentioned in Section 3.2 is the ability to overlay a physical model with DT information. The map of Stellenbosch allows for this opportunity to be realised. The energy usage information for the different components in the campus can be overlayed onto the map at the exact position of the coordinates of the component in the real world. This gives the user a better understanding of the information presented to them as more context is given to the information. An example of this overlaying of information is provided in Section 5.4.4.

Along with this map being displayed to the user, the user is also able to navigate the map as they desire. This forms the User movement aspect of the VR application architecture shown in Figure 12 and Figure 14. The user is able to navigate the environment in several ways. The first is by simply moving around their physical location which will result in the same movement in the virtual environment. The infrared sensors register the movements of the HMD and adjust the VR environment accordingly. As the user moves in the physical world, the user's GameObject is moved in the virtual world. Another mechanism of movement is using the trackpad on the controllers. A user can use the trackpad to either move forward, backward, or side-to-side for more precise movement. In addition to this, they are also able to turn in the VR environment using the trackpad instead of having to physically turn around. The last movement mechanic

is the ability to “fly”. Using the controller, the user is able to point in a direction, and when pressing the trackpad, they are able to fly towards that direction.

The controllers that the user is holding are also represented in the VR environment. In this application, they are represented as a set of gloves to give the user a more natural feel when interacting with the VR environment. The reason for showing the user the controllers is that the user can see what position the controller is at compared to their view. This allows the user to have a better sense of their movements and interactions. When interacting with the UI, the gloves, in VR, emit a laser to allow the user to see on which element of the UI they are hovering over or selecting.

5.4.3 User Interface Interaction

The purpose of the UI in the VR application is to enable the user to make a selection as to what energy information they would like to visualise. The UI makes use of cascading menus that the user follows, and in the end, a message is generated containing the information selected using the UI by the user. This message is the request for the desired information that is then transferred to the necessary component to receive the information. In the *DT implementation* this component is the Service Gateway, and with the *Non-DT implementation* the component is the Data processing functions component. The UI used in the VR application is discussed in Appendix C.2. The functionality of creating the information request message based on the selections by the user using the UI is encapsulated in the UI Functions component of the architecture. This component also contains the functionality for displaying the correct menu after one another as the user works through the UI. Figure 20 provides an example of the VR UI where a user is selecting the span of reality to visualise.

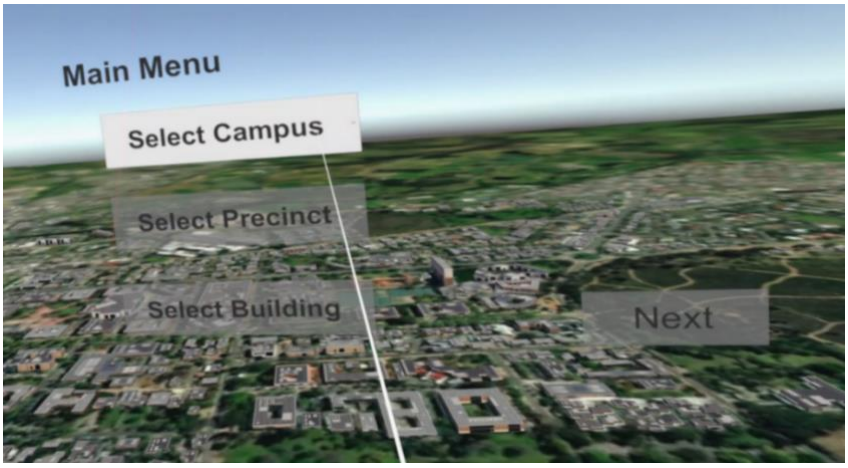


Figure 20: VR UI example

5.4.4 Displaying Information

The displaying of the selected information to the user is a vital component of the VR application. This displaying process consists of two aspects: receiving and interpreting the information, and displaying the information. An example of the information displayed in the virtual environment from the user's perspective with the visualisation UI is shown in Figure 21.

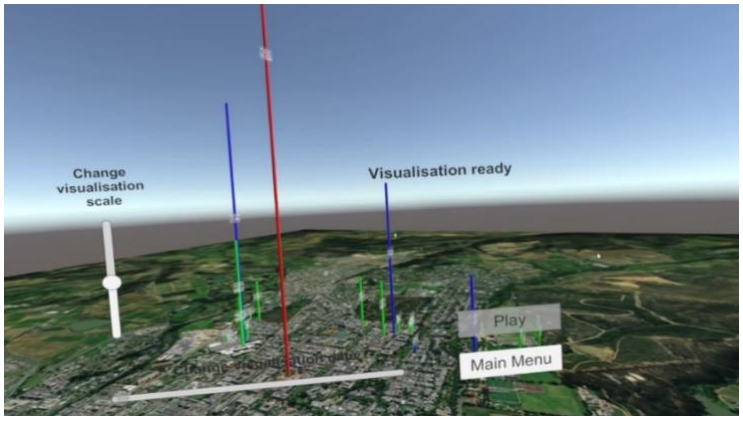


Figure 21: Information overlay in VR

The first aspect is the receiving of information. In the *DT implementation* the information is received in JSON format, whereas with the *Non-DT implementation* the information is simply transferred from one class to the other as a variable. This is possible because the *Non-DT implementation* contains all of the various layers and components in the same Unity project. Although the information received in the implementations are in different formats, the information received contains the same parameters.

In the case of the *DT implementation*, the information is first converted from a JSON string to a usable variable format, which is the same variable format used in the *Non-DT implementation*. The type of information contained in the variable is the name of the element, the coordinates of the element, the value for the energy usage, the timestamp of the energy usage value, and the element type (building, precinct or campus). This variable contains values for these parameters for every data point that will be displayed to the user. The Displaying Functions, Figure 12 and Figure 14, makes use of these parameters to display the correct information in the environment to the user.

The last aspect of the displaying process is the actual displaying of information to the user. The received and interpreted information is used to instantiate GameObjects in the virtual environment. Unity GameObjects are discussed in Section 2.4.3. Using the coordinates provided in the list of information from the variable, a GameObject can be instantiated at a location on the map of

Stellenbosch in the position that corresponds with that system element. The energy usage value in the information is then used to set the height of this GameObject.

The element type is then used to set the colour of the GameObject. In the VR application, buildings are represented by green columns, precincts by blue columns, and the campus by a red column. The GameObject also contains a panel with information about the data point. This panel indicates the name of the element, the energy usage value and the timestamp of the data point. An example of this panel with the information for a data point is shown in Figure 22.

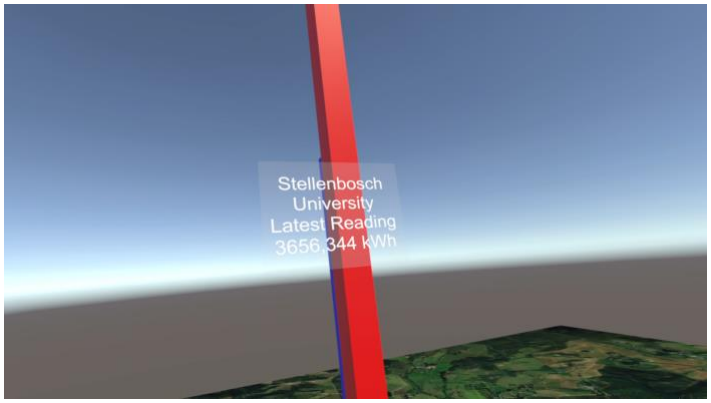


Figure 22: Information panel in VR

Once all of the GameObjects have been instantiated correctly, the user is then notified on the visualisation UI that the visualisation is ready. The user is then able to begin navigating through the information. After visualisation is complete, the user can then request new information using the UI. The process of obtaining and displaying this information is then followed again.

More examples of the operation of the VR application are provided in Appendix C.3. These examples contain some instances of a user requesting or visualising information, or adjusting the current visualisation as they desire.

5.5 Comparison of Implementations

This section compares the *DT implementation* and the *Non-DT implementation*. Various similarities are identified, as well as differences between the two implementations. The reason for this comparison is to indicate that although the two implementations have some similarities and produce the same outcome, the outcome is achieved by different means. It must be noted that the URs in Table 1 are all achieved by both the *DT implementation* and the *Non-DT implementation,* showing that the implementations are similar to one another.

The most obvious difference between the two implementations is that the *DT implementation* makes use of several components that are not part of the same software application and can operate independently of one another. This is seen with the DTs in the DT aggregation hierarchy, the services in the Shared Services component, and the VR application; all of which are independent of each other but can operate concurrently. This is in contrast with the *Non-DT implementation* that makes use of the same software application in Unity to carry out its functionality. The different layers in the *Non-DT implementation*, including the VR application environment, are, therefore, dependent on each other to continue operation.

The independently operating components allow for the *DT implementation* to have a distributed operation capability, as mentioned in Section 5.3.2.2, where different components are able to be hosted on various hardware devices on the same network. This distributed operation was tested for the *DT implementation*. Various DTs, as well as Shared Services service components were hosted on different hardware devices on the same network with the VR application being hosted on the PC mentioned in Section 5.4.1. The *DT implementation* was able to operate normally as if all the components were hosted on the same computer. In addition to this distributed operation, a second VR setup was used to be able to enter a duplicate of the developed *DT implementation* VR application, with the original VR application simultaneously, to visualise the energy usage information from the DT aggregation hierarchy. This second VR application was operated concurrently with the original VR application. Two VR applications were able to access the information from a single DT system's DT aggregation hierarchy. This distributed operation is an aspect that the *Non-DT implementation* is not capable of.

Another aspect that is similar but has slight differences for each implementation is with regards to data accessing and processing. Both implementations access information using the same API. Both implementations process the information received by the API similarly. In both implementations, the energy information for the lowest element (a building) must be aggregated upwards in the hierarchy to obtain the information for the other elements, such as a precinct or a campus. The difference between the two implementations is when this data processing occurs and what happens to the information afterwards.

In the *Non-DT implementation*, the data is processed on-demand, as the user makes requests. The processed data is then cleared to allow for a new request to be processed. Therefore, this data processing must be carried out every time the user makes a request for information. Calls to the API must also be made whenever the user requests information. The *DT implementation* operates differently in that the API is automatically periodically queried for new data. Once new data is received, it is processed, and the new information is stored in a local

database. When a user makes a request, the already processed information, stored in a local database, is retrieved and displayed to the user. The data is not required to be processed every time a user makes the request, like with the *Non-DT implementation*. This locally stored, already processed information is expected to result in a very responsive system for the *DT implementation*.

The final aspect of comparison between the two implementations is with regards to the complexities of the implementations, as well as the development challenges associated with each implementation. The *Non-DT implementation* is the simpler implementation. The reason for this is that the *Non-DT implementation* does not contain as many independently operating components as the *DT implementation*. The components are part of the same software application and, therefore, are not required to make or receive any external communication. The *Non-DT implementation* is also not required to have any long-term storing functionality like with the *DT implementation*. There is other functionality, in addition to the two mentioned, that the *DT implementation* requires that the *Non-DT implementation* does not. This added complexity and functionality results in the *DT implementation* being more complicated, time consuming, and challenging to implement.

However, the added complexity and functionality allow for the *DT implementation* to be more reconfigurable for the addition of other features. Due to the highly modular nature of the *DT implementation*, additional features can be added to the implementation without having to greatly change the core components of the system. This is because of the individually and independently operating components in the system. For example, if the energy information for the campus is to be viewed using a web application instead of a VR application, the *DT implementation* would easily be able to accommodate this. The web application would be able to connect to the Service Gateway and make requests in a similar manner to the VR application.

The same cannot be said for the *Non-DT implementation*. For a web browser, or other application, to have access to the information available in the VR application for the *Non-DT implementation*, many changes to the core components of the implementation would be required. This shows that the *DT implementation* has the potential to be more easily used for situations outside of the use only of VR for data visualisation.

The comparisons above are not exhaustive and there are several other aspects of the two implementations that could be compared. The ones mentioned above are provided to highlight some of the differences between and similarities of the two implementations.

6 Case Study Evaluation

This chapter describes the evaluation that was conducted on the case study implementations. The section begins with discussing the objectives of the evaluation, as well as what evaluation experiments were conducted. The method of these evaluation experiments is then discussed, followed by the results of these evaluation experiments. A brief discussion of the results is then provided at the end of the chapter.

6.1 Objective

The objective of the evaluation is to compare the *DT implementation* to the *Non-DT implementation*. The purpose of this comparison is to determine their relative merits for the use of VR in data visualisation.

The implementations are evaluated using three different experiments to measure: the latency in the system, the computational resource utilisation of the system, and the reconfigurability of the system.

Latency is considered as an evaluation experiment because VR is intended to aid a user in navigating and visualising information. The responsiveness of the system when a user requests new information is, therefore, an important factor to consider when implementing a VR system. This evaluation is aimed at measuring the responsiveness of the two implementations. The computational resource utilisation of each implementation is evaluated as the computational cost of such a system should remain as low as possible. This is so that no additional latency is added to the system that is the fault of insufficient computational power available. The reconfigurability of the system is also an important factor to consider for a system in the FM case study context. It is highly likely that, from time to time, the VR implementation will have to be altered and features added. The reconfigurability of the two implementations is therefore evaluated.

Although there are many other factors that are of importance to the functioning of a VR system for the purposes of data visualisation, the three aspects selected to be evaluated are deemed important fundamental aspects of such a system.

6.2 Method

This section discusses the method that was followed in the three experiments of the evaluation to show that the results obtained from the experiments are credible and that an accurate conclusion can be drawn from the results.

6.2.1 Latency and Computational Resource Utilisation

The latency and computational resource utilisation experiments are very much hardware dependent; however latency is also dependent on the network used. As such, the experiments were conducted on the same computer and network. Each experiment was conducted with multiple scenarios for added value to be extracted from the results.

A latency experiment for each implementation was conducted to determine the latency associated with the implementations. The experiment involved a user making a request for specific energy information. The time taken from when the user requests the information to when the information is displayed to the user in VR is the latency measured for the system. This experiment includes varying system scenarios as well as varying information being requested. The different system scenarios refer to the number of elements (buildings, precincts, and the campus) that are present in the system. The varying of requested information refers to the number of data points that are requested from each element, as well as which element the information is requested from. Each scenario was repeated three times to reduce the possibility of outliers, and to obtain more reliable results. In some cases, the test for the scenario was conducted more than three times depending on the scenario.

The results of this experiment were the latencies for different system scenarios and different amounts of information being requested. For each scenario in the system, there is at least one building, once precinct, and the campus. All building elements only contain one energy meter. The reason for this was to obtain an accurate comparison between the two implementation latencies that was not skewed by a building having multiple energy meters and the system having to access the API multiple times, in the case of the *Non-DT implementation*.

The computational resource utilisation experiment was tightly coupled to the latency experiment because the latency of the system is measured as the system operates and the computational resource utilisation experiment obtained results during the same system operation period. As such, the computational resource utilisation experiment was conducted at the same time as the latency experiment. During this experiment, the RAM usage and CPU usage of the system were measured for each implementation. The RAM and CPU usage recorded was the total usage for the components in the implementation. For the *DT implementation* this included the DT aggregation hierarchy, the Shared Services component, and the VR application. For the *Non-DT implementation*, this was only the usage for the VR application which contains all the system functionalities. The experiment was conducted for the same scenarios as mentioned for the latency experiment. Functionality was added to the system source code to access information from the operating system to record the RAM and CPU usages of the various components

while the system was operating. The experiment was conducted while the application was in the debug configuration, as this allowed for more accurate results for the system's overall CPU and RAM usage to be recorded.

6.2.2 Reconfiguration

The reconfiguration experiment was used to evaluate the reconfigurability of each of the implementations. In this experiment, a new feature/service was added to the system. This reconfigurability was measured in terms of how long it took to add the new feature, the lines of source code changed/added for this feature, the lines of the configuration file that was changed/added for this feature, and the percentage of code reused in adding this feature. These metrics are used to give an indication of how reconfigurable a system is with regards to time required and additional coding effort required. The experiment was conducted for three scenarios for each implementation. In each scenario, a new feature is added to the system. The first is a "Max" feature to be added that returns the maximum energy usage for a system element for a day, month, or year. The second is a "Total" feature that provides the total energy usage for a system element for a day, month, or year. The final feature is a "Cost" feature that uses the value from the "Total" feature to calculate the cost of energy usage for a day, month, or year for a system element.

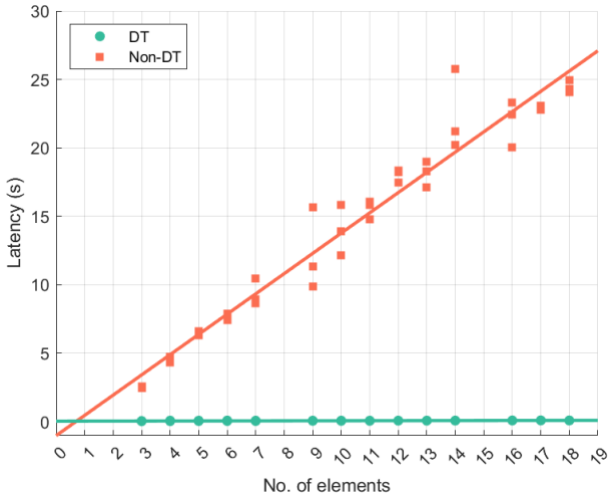
6.3 Results

This section provides the results from the three different experiments. For the latency and computational resource utilisation experiments, a brief mention of the various scenarios used in the experiments is given followed by the results of these scenarios. The results for the reconfiguration experiment are also provided for each feature added to the system.

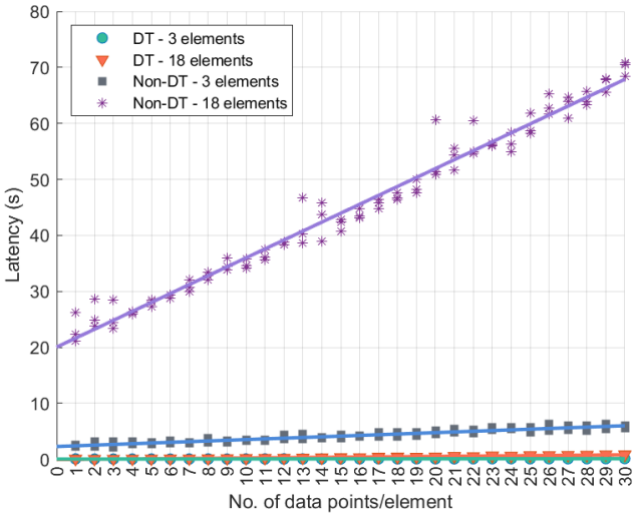
6.3.1 Latency and Computational Resource Utilisation

This experiment consisted of four different testing scenarios where the latency and computational resource utilisation of the two implementations were measured. For all the tested scenarios, the scenario began with the system containing three elements, a building, a precinct, and a campus. This is because a system like this in the FM context will at least have these three elements. The number of elements in each of the scenario was then increased to 18 elements comprised of a combination of buildings and precincts, but still with a single campus. Only a few results of the various test scenarios are provided in this section. The other recorded results are provided in Appendix D. The trend lines shown in the latency, RAM usage, and CPU usage results indicate linear trends in the measurements.

Scenario 1 and Scenario 2 are where energy usage information was requested for all the elements in the system. This includes energy information about the campus, all the precincts, and all the buildings. The difference between the two scenarios is that in the Scenario 1 only a single energy usage value was requested for all the system elements, and in Scenario 2, a range of the number of energy usage values was requested, ranging from a single requested value to requesting 30 values. The top end of this range was selected as 30 values to simulate a request being made to visualise the daily average energy usage for every day in a month. Figure 23 shows the results of the latency experiment for the two scenarios.



(a)



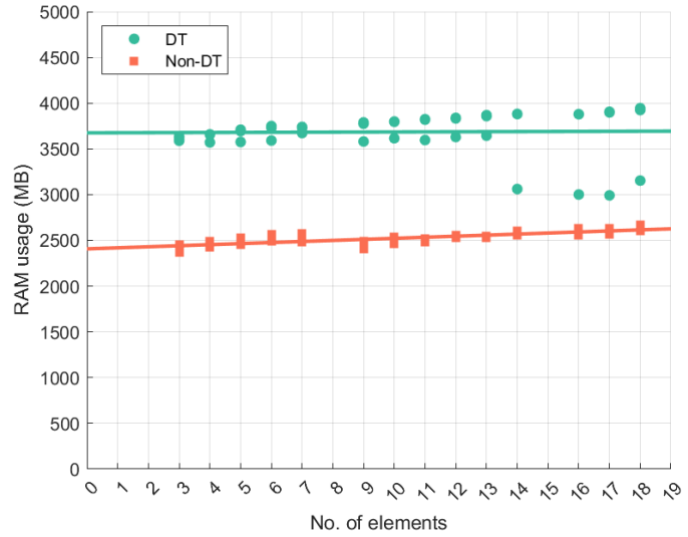
(b)

Figure 23: Scenario 1 (a) and Scenario 2 (b) latency results

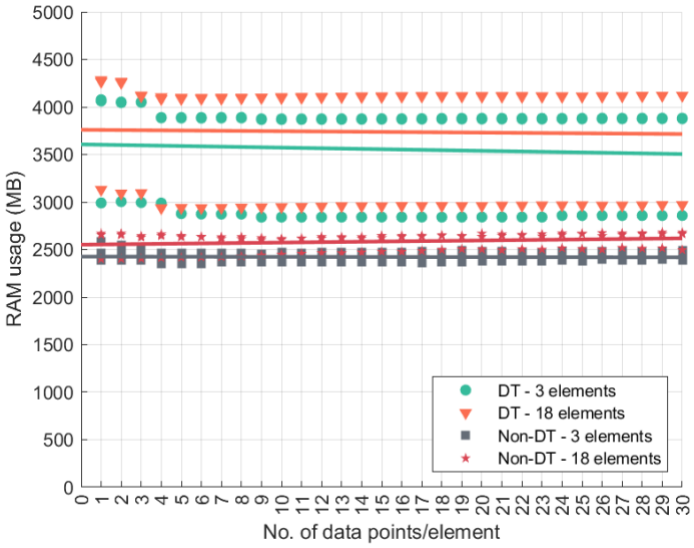
From Figure 23 it is evident that the latency for the *DT implementation* was significantly lower than the latency for the *Non-DT implementation*. The latency for both implementations increased with more elements present in the system, as well as the number of data points requested for each element. The latency of the *DT implementation* does not seem to increase in the figure, but the reason for this is the scale of the figure. The *DT implementation* had latencies in the order of tens of milliseconds, whereas the *Non-DT implementation* had latencies in the order of seconds.

Figure 24 provides the results of the RAM usage experiment for Scenario 1 and Scenario 2. The results show that the difference in RAM usage between the two implementations was modest, i.e. approximately 1000-1200 MB, which is a relatively low difference between systems that make use of a VR application. The *DT implementation* used more RAM than the *Non-DT implementation*, which is an expected outcome as the *DT implementation* had more components that require RAM, compared to the *Non-DT implementation* which only had the VR application. The VR application, in both implementations was responsible for most of the RAM usage in this system. This is expected as the VR application, developed in a game engine, is graphics heavy and requires sufficient RAM to operate and provide a satisfactory VR experience.

Figure 24b shows a large difference between the individual RAM usage results for the DT system with 3 elements and 18 elements. A reason for this could be due to the development software making use of a Garbage Collector and the system operating in the debug configuration. The Garbage Collector manages the allocation and release of memory. This Garbage Collector forms part of the system and was, therefore, included in the RAM usage results. The Garbage Collector could have allocated memory differently for the testing of the DT system and resulted in a wide range of results for the individual RAM usage measurements between tests.



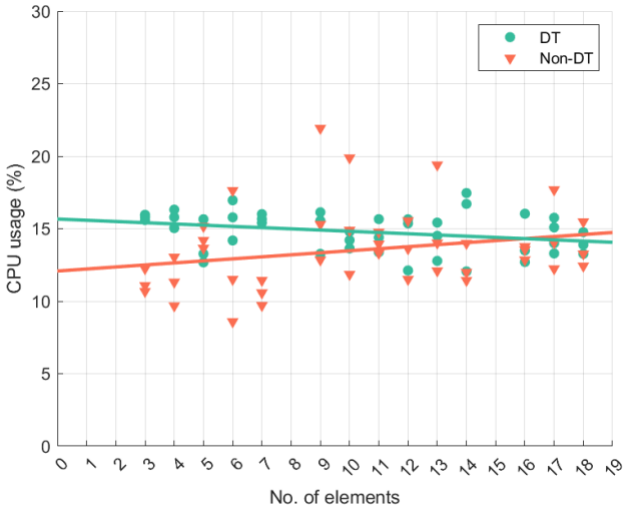
(a)



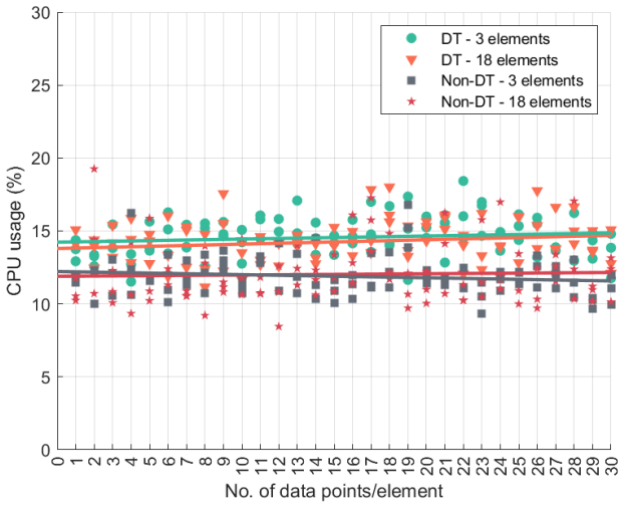
(b)

Figure 24: Scenario 1 (a) and Scenario 2 (b) RAM usage results

The results of the CPU usage of the two implementations for Scenario 1 and Scenario 2 are provided in Figure 25. The results indicate that the CPU usage for both implementations in both scenarios was very similar. The CPU usage for the two implementations was typically between 10% and 20% during operation with the *DT implementation* using slightly more.



(a)



(b)

Figure 25: Scenario 1 (a) and Scenario 2 (b) CPU usage results

The results presented in Figure 23 to Figure 25 indicate that the most glaring difference in performance between the two implementations was with regards to the latency in the system. A possible reason for this discrepancy in latencies is the *Non-DT implementation* having to reactively make requests to the API and having, then, to also process the data as necessary. This contrasts with the *DT implementation* where API requests are done proactively so that information is available in a local database and little processing of information is required when a request is received. The RAM and CPU usages were similar for both implementations and no significant differences are noted in this respect.

Test Scenario 3 and Scenario 4 were used to determine the latency and computer resource utilisation associated with requesting information for a single system component. In these scenarios, only information for a single building element was requested. In Scenario 3, like Scenario 1, only one energy value was requested, and in Scenario 4, like Scenario 2, a range of the number of values was requested. As only a single building's information was requested in these scenarios, only one API call was made for the *Non-DT implementation* because the building only had one energy meter. This was in contrast with the other scenarios with multiple components which resulted in multiple API calls needing to be made in the *Non-DT implementation.* These scenarios were used to determine whether a large portion of the latency or computer resource utilisation was due to the API calls being made or if it was simply due to the selected architecture. The latency results for Scenario 4 are shown in Figure 26.

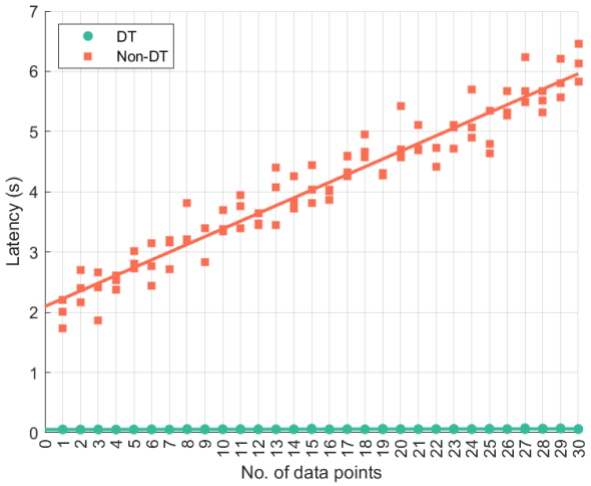


Figure 26: Scenario 4 latency results

A similar trend to Scenario 1 and 2 is present in the latency results in Figure 26. The *Non-DT implementation* had a higher latency than the *DT implementation*, even if only information for a single element was requested. Figure 26 shows the results for a single data point request for a building element. This request required minimal processing, and as such, shows that the latency of receiving data from the API is the major contributing factor to the overall latency for the *Non-DT implementation*.

The RAM usage results for Scenario 4 are shown in Figure 27. The results indicated, as expected due to the results of the previous scenarios, that the RAM usage for both implementations in these scenarios was very similar with an average difference of approximately 750-1000 MB between the implementations. The *DT implementation* was shown to use slightly more RAM than the *Non-DT*

implementation, and this is expected due to the *DT implementation* having more components operating concurrently.

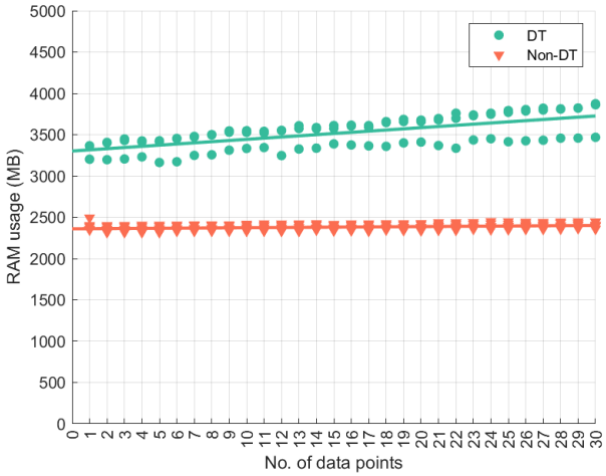


Figure 27: Scenario 4 RAM usage results

The results shown in Figure 28 indicate CPU usages for both implementations that are very similar to the results obtained in Scenarios 1 and 2. The results indicate that both the *DT implementation* and the *Non-DT implementation* used between 10% to 20% of the CPU with the *DT implementation* using slightly more. The results in Figure 28, along with the RAM usage results in Figure 27, further emphasise the conclusion that the type of implementation (either DT or Non-DT) has little difference in terms of the computer resource utilisation of the system.

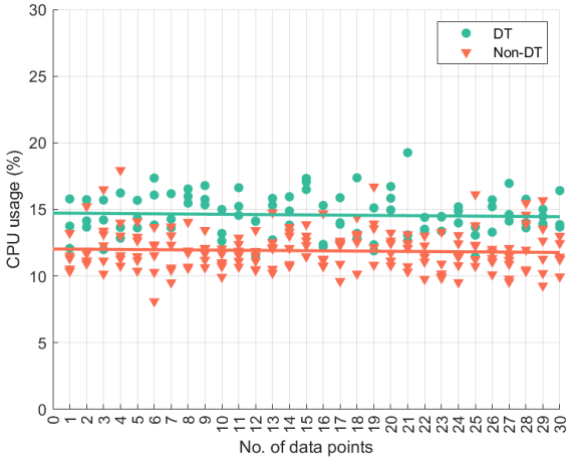
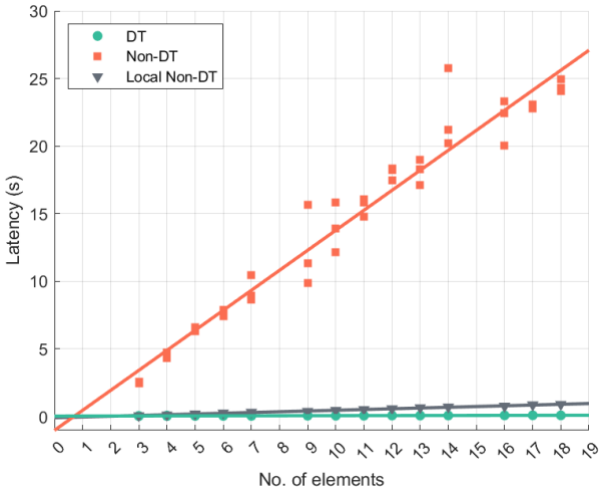


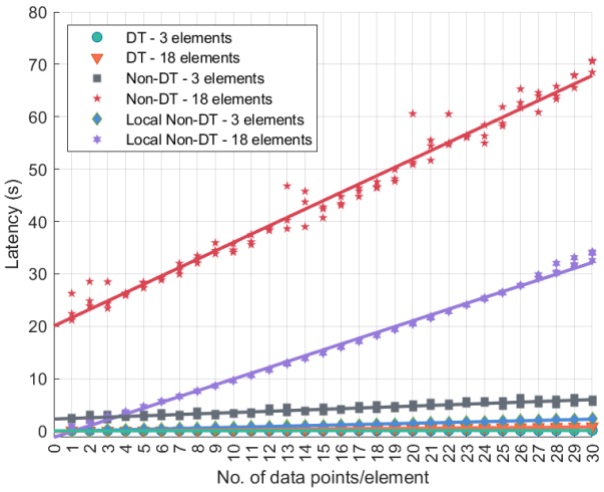
Figure 28: Scenario 4 CPU usage results

The comparison between latencies of the two different implementations show that the *Non-DT implementation* had much larger latencies than the *DT implementation*. A reason for this is that the *Non-DT implementation* is required

to make requests to the API when information is needed. This request to the API takes a large amount of time to be fulfilled, compared to the *DT implementation* that makes requests to a local NoSQL storage database when fulfilling a request. To evaluate this hypothesis, the test scenarios conducted previously were carried out again for the *Non-DT implementation* that had a slight modification. This modification was that instead of requests being made to the API, requests were made to a local NoSQL storage database, like with the *DT implementation*. This local storage database contained the same information that was available directly from the API, this being the raw energy data. The data from this database still had to be processed, as done originally for the *Non-DT implementation*, to return the correct energy usage information that was being requested. Some of the latency results for Scenario 1 and Scenario 2 with this change are shown in Figure 29.



(a)



(b)

Figure 29: Scenario 1 (a) and Scenario 2 (b) local database latency results

Figure 29 shows that the use of a local database reduced the latency experienced in the *Non-DT implementation* significantly. However, this latency was still higher than for the *DT implementation*. A possible reason for this was due to the processing of information that was still required in the *Non-DT implementation* to fulfil the request. As such, Scenarios 3 and 4 were tested again with this use of a local database. The results for the Scenario 4 test are provided in Figure 30.

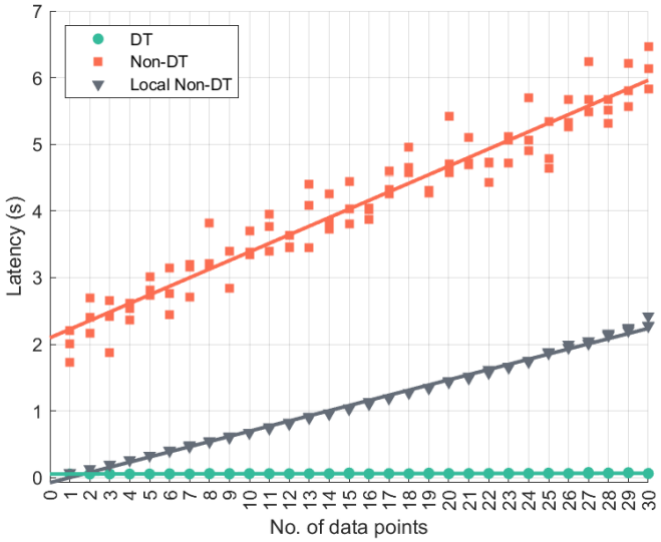


Figure 30: Scenario 4 local database latency results

Figure 30 shows that even with the use of a local database, the latency of the *Non-DT implementation* was still significantly higher than with the *DT implementation*. The expected reason for this is thought to be the time required to process the necessary data to fulfil the request. In the *DT implementation* the data had already been processed by the Context generation service and this already processed information was then used to fulfil the request. The computer resource utilisation results for this modification to the *Non-DT implementation* are provided in Appendix D. The results indicate a similar trend that was present in the other computer resource utilisation test scenarios conducted previously.

6.3.2 Reconfiguration

This section provides the results of the reconfiguration experiments for the *DT implementation* and the *Non-DT implementation.* As mentioned previously, the experiment consisted of adding three new features to the already existing systems.

Both implementations required changes to the source code, and possibly to the configuration file, for the features to be added. For the *DT implementation*, there is a possibility to alter the source code or configuration file for the DT aggregation

hierarchy, the Shared Services component, or the VR application. The details of which components were altered are provided in the results. For the *Non-DT implementation*, the system functionality was developed in the VR application, as such there was only one component whose source code or configuration file could be changed, that being the Unity code, unlike with the *DT implementation.*

6.3.2.1 Max Feature

The addition of the Max feature required a change to the UI in the VR application for both implementations. The functionality was added to allow for the system to return the maximum energy usage for a selected element and time period. The reconfiguration results of adding this Max feature for both implementations are shown in Table 6 and Table 7, respectively.

Table 6: DT implementation Max feature reconfiguration results

Lines of code added	**Unity code**	**Shared Services Component code**	**DT Component code**	**Configuration file**	**Total lines added**
	9	0	690	0	699
Lines of code reused	**Unity code**	**Shared Services Component code**	**DT Component code**	**Configuration file**	**Total lines reused**
	6	0	469	0	475
Percentage of code reused	**Unity code (%)**	**Shared Services Component code (%)**	**DT Component code (%)**	**Configuration file (%)**	**Total lines reused (%)**
	66.67	0	67.97	0	67.95
Time taken (hrs:min)	02:45				

Table 6 indicates that only changes to the DT aggregation hierarchy components and the Unity code were required. This is expected as the Max feature is a new service that is offered by the DT. This added service also then required an update in the Unity code in the form of adding a button and subsequent functionality to make use of the service. Table 6 also shows that almost three hours were required to add this feature to the *DT implementation*. A reason for this is that the test was conducted once both implementations were complete. The *DT implementation* was completed first, as such, a certain degree of "relearning" the specifics in terms of code layout and functions was required. This "relearning" aspect took place during testing, unbeknownst to the developer, and gives an incorrect impression

of the actual time that was required to implement the new feature. The Total feature test was conducted afterwards to determine whether the time taken result for the Max feature was skewed by the "relearning" aspect of the test. These results are further discussed in Section 6.3.2.2.

Table 7: Non-DT implementation Max feature reconfiguration results

Lines of code added	**Unity code**	**Configuration file**	**Total lines added**
	406	0	406
Lines of code reused	**Unity code**	**Configuration file**	**Total lines reused**
	341	0	341
Percentage of code reused	**Unity code (%)**	**Configuration file (%)**	**Total lines reused (%)**
	84	0	84
Time taken (hrs:min)	00:21		

From Table 6 and Table 7 it is evident that more total lines of code were required to add the Max feature to the *DT implementation* compared to the *Non-DT implementation*. The reason for this is that the *DT implementation* has complexities that are not present in the *Non-DT implementation*. An example of these complexities is that a DT in the *DT implementation* is required to process and store the information for this Max feature. This processing and storing process resulted in more code being required, and some of this added code not being from reused code. This is the reason for the higher percentage of code reused for the *Non-DT implementation* compared to the *DT implementation*. The time taken to add this Max feature to the *Non-DT implementation* is also significantly shorter than with the *DT implementation*.

6.3.2.2 Total Feature

The addition of the Total feature was similar to the addition of the Max feature and was used to determine if the "relearning" aspect played a significant role in the Max feature experiment. Table 8 and Table 9 provide the reconfigurability results for the addition of the Total feature to the *DT implementation* and the *Non-DT implementation*, respectively.

Table 8 shows that the only code changes required were with the DT aggregation hierarchy code and the Unity code. The Total feature was an additional service offered by the DTs. The changes to the Unity code were the addition of a new UI button as well as the functionality for the user's selection of this new service. Table 8 indicates that the time required to add the Total feature to the *DT implementation* was significantly shorter than the time required for the Max feature in Table 6. This shows that the addition of the Max feature required time

for "relearning" of the *DT implementation* which resulted in a longer implementation time. As this "relearning" was already complete, the time required to add the Total feature is only based on the addition of the feature and is a better representation of the time required to add a new feature to the *DT implementation*.

Table 8: DT implementation Total feature reconfiguration results

Lines of code added	**Unity code**	**Shared Services Component code**	**DT Component code**	**Configuration file**	**Total lines added**
	8	0	605	0	613
Lines of code reused	**Unity code**	**Shared Services Component code**	**DT Component code**	**Configuration file**	**Total lines reused**
	4	0	432	0	436
Percentage of code reused	**Unity code (%)**	**Shared Services Component code (%)**	**DT Component code (%)**	**Configuration file (%)**	**Total lines reused (%)**
	50	0	71.4	0	71.13
Time taken (hrs:min)	00:40				

Table 9: Non-DT implementation Total feature reconfiguration results

Lines of code added	**Unity code**	**Configuration file**	**Total lines added**
	396	0	396
Lines of code reused	**Unity code**	**Configuration file**	**Total lines reused**
	335	0	335
Percentage of code reused	**Unity code (%)**	**Configuration file (%)**	**Total lines reused (%)**
	84.6	0	84.6
Time taken (hrs:min)	00:15		

Table 8 and Table 9 show that, like with the addition of the Max feature, the *DT implementation* required more additional lines of code than the *Non-DT implementation* to add the Total feature. However, like with the Max feature, the reason for more lines of code being required is due to added complexities associated with the *DT implementation* that have been mentioned previously. The

Non-DT implementation also had a better percentage of code reused than the *DT implementation*. The *Non-DT implementation* also required less time to add the Total feature than the *DT implementation*. The reason for this better percentage code reused and less time required is, again, linked to the added complexities with the *DT implementation*.

6.3.2.3 Cost Feature

The final reconfiguration test was the addition of the Cost feature for the VR application. This would use the value returned by the Total feature and perform a calculation for the cost of the total energy used for a system element for the selected time period. This cost calculation was based on the electricity rate in South Africa of R 1.209/kWh (GlobalPetrolPrices.com, 2021). Table 10 and Table 11 provide the reconfigurability results for the addition of the Cost feature to the *DT implementation* and the *Non-DT implementation*, respectively.

Table 10: DT implementation Cost feature reconfiguration results

Lines of code added	**Unity code**	**Shared Services Component code**	**DT Component code**	**Configuration file**	**Total lines added**
	17	105	0	1	123
Lines of code reused	**Unity code**	**Shared Services Component code**	**DT Component code**	**Configuration file**	**Total lines reused**
	6	81	0	0	87
Percentage of code reused	**Unity code (%)**	**Shared Services Component code (%)**	**DT Component code (%)**	**Configuration file (%)**	**Total lines reused (%)**
	35.3	0	0	0	70.73
Time taken (hrs:min)	00:11				

Table 10 shows that in total 123 lines of code were added to the *DT implementation* for the addition of the "Cost" feature. The majority of this added code was in the Shared Services component. The other lines of code added were in the Unity code and the configuration file. Unlike with the other reconfiguration experiments where the added feature was a service offered by a DT, this added feature was a service that was offered by the Services network. The percentage of code reused was similar to the other reconfiguration experiments. The amount of time required to add this feature was much lower in comparison to adding the

other features. The reason for this is that adding a service to the Services Network does not have the added complexities that are encountered when adding a service to a DT. This reduction in complexity is also the reason why fewer lines of code were required to add the feature.

Table 11: Non-DT implementation Cost feature reconfiguration results

Lines of code added	**Unity code**	**Configuration file**	**Total lines added**
	51	0	51
Lines of code reused	**Unity code**	**Configuration file**	**Total lines reused**
	34	0	34
Percentage of code reused	**Unity code (%)**	**Configuration file (%)**	**Total lines reused (%)**
	66.67	0	66.67
Time taken (hrs:min)	00:12		

Table 11 shows that fewer lines of code were required to add the "Cost" feature to the *Non-DT implementation* compared to with the *DT implementation* in Table 10. However, although fewer lines of code were required, the percentage of code reused was less than for the *DT implementation*. Although the time required to add the feature was a minute longer than for the *DT implementation*, this is not significant enough to indicate that there was a great time difference between the two implementations for the addition of this Cost feature.

6.4 Discussion

The results presented in Section 6.3 show that in terms of latency, computer resource utilisation, and reconfigurability, latency was the biggest difference between the *DT implementation* and the *Non-DT implementation*. The *DT implementation* had latencies, with regards to obtaining and displaying information, in the order of milliseconds, whereas the *Non-DT implementation* had latencies for the same process in the order of seconds or even tens of seconds. Even with the use of a local database for both implementations, the *DT implementation* still had lower latencies.

These results indicate that the *DT implementation* provided better responsiveness, which is desired for a VR application, than the *Non-DT implementation*. Although this is evident, there is a reason for this large discrepancy in latencies between the two implementations. As mentioned in Section 5.5, both implementations access the energy usage data from an API. The *DT implementation* processes this data and stores the processed information in a local database that is then later queried for information when requested. The *Non-*

DT implementation on the other hand, carries out the same process of accessing and processing information like with the *DT implementation*. The difference between the two implementations is that the accessing and processing is carried out on user request for the *Non-DT implementation* but has already been completed for the *DT implementation* and the information must just be requested from the local database.

The latency experiment, therefore, did compare the implementations to one another, but a nuance in this comparison was that the comparison could be seen as a comparison between pre-processed results and on-demand processed results. The *Non-DT implementation* could also make use of this pre-processing functionality, and the latencies would, in theory, then be comparable to those with the *DT implementation*. However, the reason this functionality was not implemented was due to the nature of the energy usage information available for FM, and what has been observed in other uses of VR for data visualisation. With new data being constantly recorded, the *Non-DT implementation* would have to be constantly updating the processed information, which is an inherent functionality of DTs. This implementation would then begin encroaching on the philosophies of DTs and possibly no longer be a *Non-DT implementation*, and rather a *DT implementation*. Based on this reasoning, the comparison of latencies in Section 6.3 between the two implementations, considering the pre-processing versus on-processing calculation argument, is, therefore, a fair comparison to make. The conclusion of this latency comparison is then that the *DT implementation* was a better choice for a more responsive VR application for data visualisation.

With regards to the computer resource utilisation of each implementation, the two implementations required similar computational resources in the tested scenarios. The *DT implementation* required more RAM than the *Non-DT implementation*. This was expected due to the *DT implementation* having more components operating. Although the *DT implementation* required more RAM, the increased amount of RAM required (between 750-1200 MB) was not a significant enough value to indicate the *Non-DT implementation* being the better choice. The CPU usage of both implementations was also fairly similar for the tested scenarios. In terms of computer resource utilisation, the two implementations performed similarly showing that the choice of implementation did not greatly affect computer resource utilisation of the systems.

The final aspect of the two implementations tested was the reconfigurability of the implementation. The *Non-DT implementation* was shown to perform better with regards to reconfigurability in comparison to the *DT implementation*. The *Non-DT implementation* generally required less time and lines of code to add the same features than the *DT implementation*. The *Non-DT implementation* also had a slightly better percentage of code reused in two of the reconfigurability tests.

The reason for the *DT implementation* requiring more code and time to add a feature was because of the additional complexity associated with the implementation. Additional functionality was required for processing, storing, and accessing of information in the *DT implementation*, that was not required in the *Non-DT implementation*. This additional functionality was only present when a service was added to a DT. If a service was added to the Services Network, the reconfigurability results for the *DT implementation* were similar to those of the *Non-DT implementation* indicating that the added functionality required by a DT was the reason for more code and time being required in the *DT implementation*. However, regardless of the additional functionality required in the *DT implementation,* the results of the reconfiguration experiment indicated that the *Non-DT implementation* was better in terms of reconfigurability than the *DT implementation* for a VR application for data visualisation.

A reason for the similar and, slightly, better reconfigurability for the *Non-DT implementation* could be due to the order in which the two implementations were developed. The *DT implementation* was developed first and then the *Non-DT implementation*. Because of this order, some of the principles used to develop a reconfigurable DT system could have then been unintentionally applied and influenced the *Non-DT implementation*. If the order of development was reversed, with the *Non-DT implementation* being developed first and then the *DT implementation*, the results for the reconfigurability experiment could have been different as some of the *Non-DT implementation* development might have been influenced by the development of the *DT implementation*.

It must be noted that the reconfigurability experiments were relatively simple and were used to showcase, primarily, how the integration of DTs and VR is affected by the addition of new features. More complex reconfigurations could be encountered on either side of the integrated system, i.e. the DT side or VR side. The complexity of the reconfigurations could yield differing results to what was obtained in the reconfiguration experiment with adding simple features. Therefore, the conclusions drawn from the reconfigurability experiment are limited in that they are very dependent on what feature was added.

7 Discussion and Further Work

This chapter discusses the integration of a DT and VR system for the purpose of data visualisation. In the chapter, a discussion of the two implementations, the *DT implementation* and *Non-DT implementation*, is provided to determine if the use of DTs is more beneficial than the non-DT counterpart. The implications from the case study, with regards to DTs and VR, are provided, followed by the overall implications for the integration of DTs and VR. The final section focuses on the possible areas of further work than can be explored in this field.

7.1 Case Study Implications

The results of the case study evaluation presented in Chapter 6 show that the *DT implementation* and the *Non-DT implementation* perform similarly with regards to computer resource utilisation. The areas where they differ are with the latency of the system and, slightly, with the reconfigurability of the system. The *DT implementation* is shown to be more responsive than the *Non-DT implementation*; whereas, the *Non-DT implementation* is reconfigured more easily than the *DT implementation*. These results show that there is, overall, marginal differences, apart from latency, between the two implementations and that either implementation could be used to allow for data visualisation using VR. The latencies experienced in the *Non-DT implementation* are, however, significantly higher than with the *DT implementation* and this will be noticed by a user. Even with the modification and evaluation of using local storage databases for the *Non-DT implementation*, the latencies encountered are still significantly higher in comparison to the *DT implementation*.

As both implementations can be used, the question arises then as to which implementation should rather be selected. The answer to this is dependent on the intended use of such a system. For the cases used in the thesis, it can be concluded that if the intended use is mostly for visualising information in a VR environment, then the Non-DT implementation, with some modifications for lower latency, should be selected. The reason is that this implementation is less complex than the *DT implementation*, requiring less development time and effort as was discussed in Section 5.5. The *Non-DT implementation* does not contain multiple individually operating components like the *DT implementation*. This results in a lower complexity as the system need not be designed with consideration of the communication between and operation of these individually operating components. This Non-DT implemented system will still achieve the same outcome as the *DT implementation*.

However, if the intended use of the system, present or possibly even future, is not only for visualising information in a VR environment, then the *DT implementation*,

with its various benefits, is the more favourable choice. The *DT implementation* is not limited to only being used for a VR application, unlike the *Non-DT implementation*. The *DT implementation* can be expanded to make provision for other applications used in the data driven decision-making process, like a web application. This is due to the modularity offered by the system with the individually and independently operating components that can communicate with one another. In addition to this, the use of DTs also has some other advantages for complex systems discussed in Section 2.1.

Chapter 3 provides various opportunities and challenges that might be present with the integration of DTs and VR. The *DT implementation* in the case study provided further insight to some of the challenges. A challenge that was encountered with the case study was deciding on what information to make available to the user, and how to best display this information. This challenge led to the subsequent encountered challenge of information overloading sometimes experienced by the user. The comfort of the user was another encountered challenge. When using the VR application, the user sometimes experienced instances of discomfort, in the form of motion sickness.

Although some of the challenges identified in Chapter 3 were encountered in the *DT implementation*, some identified opportunities were also realised. One such opportunity was being able to supplement the DT information with more context. This contextualisation was in the form of overlaying a map of Stellenbosch with information obtained from the DTs, which allowed the user to better understand the information they were visualising. Another realised opportunity was the DTs in the *DT implementation* allowing for the system to be able to respond in as near real-time as possible. A user always had access to the latest energy usage information because of this benefit of DTs. An opportunity that is not mentioned in Chapter 3, but was identified during the implementation of the case study, was with the possibility of distributed operation. The *DT implementation* was shown to be able to have different components hosted on multiple hardware devices connected to the same network. This distributed operation could aid in reducing the computational resource utilisation of the system that is imposed on individual hardware components. Another realised opportunity is the ability for the *DT implementation* to allow for multiple VR applications to have access to the same information.

7.2 Overall Implications

The implications of the case study, although specific to the case study, can be applied more generally. The case study focused on implementing a DT and VR system in the FM context. The implications of this system indicated that a DT and VR system can be developed for an application outside of the FM context. The

identified and realised opportunities and challenges of Chapter 3 and the case study, respectively, are not context specific and indicate that the use of VR and DTs does have the potential to enhance the data driven decision-making process in other areas.

DTs are shown to be a useful technology for the data driven decision-making process regardless of the visualisation method being used, due to the possibility of integrating multiple data visualisation tools. DTs allow the various elements of a complex system to be encapsulated in a manageable and value-providing system. The complex DT system design framework, discussed in Section 2.2, and SLADTA have been shown to be useful tools in creating an FM DT system. This shows how these tools that could potentially also be applied to other contexts.

Both implementations required the use of an API to request energy usage data. It is sometimes the case that data is not as easily available as with the use of this API and a system may need to be developed to be able to access information directly from sensors, e.g. energy meters. In that case, the advantages of using DTs instead of a non-DT method would be much more apparent as DTs can be used for this full data pipeline to transfer data from a sensor to a VR environment for visualisation. This data pipeline is difficult to emulate using non-DT methods.

7.3 Further Work

The current research has several areas where further work could be carried out to obtain more insight for the integration of DTs and VR. With regards to the FM context, the case study only focused on the use of one utility – energy usage. More utilities can be included such as water usage, solar power generation and usage, and building occupancy – all of which are areas of interest in FM. The case study does not consider information within the building and focuses on an entire building as the lowest element in the system. Further work could be conducted into focusing on the information within the building, such as the maintenance tasks within the building or utility usage within various parts of the building.

As mentioned previously, the concepts generated in this thesis can be applied more generally, to contexts outside of FM. Some of these other possible contexts include the manufacturing industry, the healthcare industry, and the agricultural industry to name a few. These contexts, like FM, contain large amounts of recorded information for the data driven decision-making process. These contexts are also complex systems with many components. The use of a DT and VR system for these complex contexts would allow for further evaluation of the concept of integrating DTs and VR. Further work can also be done in using VR as a data visualisation medium and determining its effectiveness.

The case study makes use of Unity as VR development software. Further work could evaluate the use of 3D visualisation development software currently used in industry, such as ABB RobotStudio, instead of making use of a game engine software like Unity. This evaluation will indicate the merits of using existing and already adopted software to allow for the integration of DTs and VR in industry.

New integration techniques, for DTs and VR, could also be identified to determine the best method to integrate the two technologies. Regardless, further investigation will provide valuable information regarding the use of DTs and VR, and if this is a useful combination of developing Industry 4.0 technologies.

Further work could also be carried out to evaluate the scalability of a system using the integration approach proposed in this thesis. The scalability should be evaluated for a VR system that makes use of DTs, and also a VR system that does not use DTs. This will provide more insight into which approach is more scalable. An evaluation should also be conducted on determining whether the use of other development tools will have an impact on the performance of the developed system. This will provide insight into whether the advantages and disadvantages of the two approaches, DT or non-DT, are inherent to the fundamental technology or dependent on the development tools used for implementing the approaches.

8 Conclusion

The research evaluates the integration of two developing technologies in Industry 4.0: DTs and VR. A DT and VR integrated system was developed for use in the FM context for the FM division at Stellenbosch University. The system was used to visualise the energy usage information for various facilities within the university. A system architecture was first designed from the identified user needs, and then implemented. This implemented system showed that a DT and VR system for data visualisation is possible. In conjunction with this DT and VR system, a VR application that does not make use of DTs was also developed. This non-DT system was designed to produce the same output as the DT system, but using different methods. The two implementation methods were then compared to evaluate the integration of DTs and VR.

The two implementations were evaluated with regards to latency, computer resource utilisation (in terms of RAM and CPU usage), and reconfigurability. It was evident, from the results of these evaluations, that the DT system was more responsive and had lower latencies than the non-DT system. Both systems had also been shown to have a similar level of computation resource utilisation. The reconfigurability tests conducted on both systems indicated that the non-DT system was more reconfigurable than the DT system. This was, however, expected as the DT system had higher complexity and more functionality associated with it, which was not present or required in the non-DT system.

The evaluations indicated that either method is a suitable implementation choice for using VR to visualise data to make decisions. If, however, there is a possibility of expanding the system outside of the use of only VR or accommodating multiple concurrent, but independent, VR setups, the DT method is a better implementation choice. The use of DTs has various benefits that support its use for complex systems. The non-DT method cannot, in most cases, provide the same benefits, and as such, is not a suitable implementation choice for such a system.

The integration of DTs and VR has many opportunities and challenges that are associated with it. The implemented DT system allowed for some of these opportunities and challenges to be realised in a real-world context. The use of these two technologies were shown to allow a user to use VR, with the support of DTs, for the data driven decision-making process for complex systems.

There is still much work that can be conducted to evaluate the integration of the two technologies, but this research provides valuable insight into the use of DTs and VR; two technologies that are constantly developing and growing in use in Industry 4.0. These two technologies could further be used for other aspects of Industry 4.0 that could aid various other contexts and industries in adopting the concepts associated with Industry 4.0.

9 References

Akpan, I.J. & Shanker, M. 2019. A comparative evaluation of the effectiveness of virtual reality, 3D visualization and 2D visual interactive simulation: an exploratory meta-analysis. *Simulation*. 95(2):145–170.

Andersen, B.J.H., Davis, A.T.A., Weber, G. & Wunsche, B.C. 2019. Immersion or diversion: Does virtual reality make data visualisation more effective? in *ICEIC 2019 - International Conference on Electronics, Information, and Communication.* Auckland, New Zealand.

Borangiu, T., Oltean, E., Răileanu, S., Anton, F., Anton, S. & Iacob, I. 2020. Embedded Digital Twin for ARTI-Type Control of Semi-continuous Production Processes. *Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future. SOHOMA 2019. Studies in Computational Intelligence*, Vol 853 Springer, Cham. 113–133. Valencia, Spain

Cordeil, M., Dwyer, T., Klein, K., Laha, B., Marriott, K. & Thomas, B.H. 2017. Immersive Collaborative Analysis of Network Connectivity: CAVE-style or Head-Mounted Display? *IEEE Transactions on Visualization and Computer Graphics*. 23(1):441–450.

Da Silva, G.S., Kruger, K. & Basson, A.H. 2022. Opportunities for Visualising Complex Data by Integrating Virtual Reality and Digital Twins. in *International Conference on Competitive Manufacturing, COMA '22.* Stellenbosch, South Africa.

Davies, A. n.d. *10 Great Tools for VR Development I DevTeam.Space*. [Online], Available: https://www.devteam.space/blog/10-great-tools-for-vr-development/ [2022, August 03].

Donalek, C., Djorgovski, S.G., Cioc, A., Wang, A., Zhang, J., Lawler, E., Yeh, S., Mahabal, A., Graham, M., Drake, A., Davidoff, S., Norris, J.S., Longo, G. 2015. Immersive and collaborative data visualization using virtual reality platforms. in *2014 IEEE International Conference on Big Data, IEEE Big Data 2014.* Washington, USA: IEEE. 609–614.

Drouhard, M., Steed, C.A., Hahn, S., Proffen, T., Daniel, J. & Matheson, M. 2015. Immersive visualization for materials science data analysis using the Oculus Rift. in *2015 IEEE International Conference on Big Data, IEEE Big Data 2015.* Santa Clara, USA: IEEE. 2453–2461.

El Beheiry, M., Doutreligne, S., Caporal, C., Ostertag, C., Dahan, M. & Masson, J.B. 2019. Virtual Reality: Beyond Visualization. *Journal of Molecular Biology*.

431(7):1315–1321.

Erra, U., Malandrino, D. & Pepe, L. 2019. Virtual Reality Interfaces for Interacting with Three-Dimensional Graphs. *International Journal of Human-Computer Interaction*. 35(1):75–88.

Filho, J.A.W., Rey, M.F., Freitas, C.M.D.S. & Nedel, L. 2018. Immersive Visualization of Abstract Information: An Evaluation on Dimensionally-Reduced Data Scatterplots. in *25th IEEE Conference on Virtual Reality and 3D User Interfaces, VR 2018 - Proceedings.* Tuebingen/Reutlingen, Germany: IEEE. 483–490.

Glaessgen, E.H. & Stargel, D.S. 2012. The digital twin paradigm for future NASA and U.S. Air force vehicles. in *53rd Structures, Structural Dynamics, and Materials Conference.* Honolulu, USA.

GlobalPetrolPrices.com. 2021. *South Africa electricity prices*. [Online], Available: https://www.globalpetrolprices.com/South-Africa/electricity_prices/ [2022, July 27].

Gracia, A., González, S., Robles, V., Menasalvas, E. & Von Landesberger, T. 2016. New insights into the suitability of the third dimension for visualizing multivariate/multidimensional data: A study based on loss of quality quantification. *Information Visualization*. 15(1):3–30.

Grieves, M. & Vickers, J. 2017. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. *Transdisciplinary Perspectives on Complex Systems*. 85–113.

Havard, V., Jeanne, B., Lacomblez, M. & Baudry, D. 2019. Digital twin and virtual reality: a co-simulation environment for design and assessment of industrial workstations. *Production and Manufacturing Research*. 7(1):472–489.

Hu, M., Luo, X., Chen, J., Lee, Y.C., Zhou, Y. & Wu, D. 2021. Virtual reality: A survey of enabling technologies and its applications in IoT. *Journal of Network and Computer Applications*. 178.

Human, C. 2022. *A Design Framework for Aggregation in a System of Digital Twins*. PhD (Eng) Dissertation. University of Stellenbosch. Stellenbosch.

Human, C., Basson, A.H. & Kruger, K., 2021. Digital twin data pipeline based on SLADTA and MQTT. *Service Oriented, Holonic and Multi-Agent Manufacturing Systems for Industry of the Future: Proceedings of SOHOMA 2020*. Studies in Computational Intelligence, 952:111-122. Paris, France.

Juarez, M.G., Botti, V.J. & Giret, A.S. 2021. Digital twins: Review and challenges. *Journal of Computing and Information Science in Engineering*. 21(3).

Kovar, J., Mouralova, K., Ksica, F., Kroupa, J., Andrs, O. & Hadas, Z. 2017. Virtual reality in context of Industry 4.0. in *2016 17th International Conference on Mechatronics - Mechatronika (ME).* Prague, Czech Republic: Czech Technical University in Prague.

Kritzinger, W., Karner, M., Traar, G., Henjes, J. & Sihn, W. 2018. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*. 51(11):1016–1022.

Kroupa, J., Tumova, E., Tuma, Z., Kovar, J. & Singule, V. 2018. Processing and visualization of microclimatic data by using virtual reality. *MM Science Journal*. 2018(December):2621–2624.

Kruger, K. n.d. *MAD Research Group - Research*. [Online], Available: https://sites.google.com/view/mad-research-group/research?authuser=0 [2021, July 27].

Kuts, V., Otto, T., Tähemaa, T. & Bondarenko, Y. 2019. Digital twin based synchronised control and simulation of the industrial robotic cell using virtual reality. *Journal of Machine Engineering*. 19(1):128–144.

Liagkou, V., Salmas, D. & Stylios, C. 2019. Realizing Virtual Reality Learning Environment for Industry 4.0. *Procedia CIRP*. 79:712–717.

Malik, A.A., Masood, T. & Bilberg, A. 2020. Virtual reality in manufacturing: immersive and collaborative artificial-reality in design of human-robot workspace. *International Journal of Computer Integrated Manufacturing*. 33(1):22–37.

Negri, E., Fumagalli, L. & Macchi, M. 2017. A Review of the Roles of Digital Twin in CPS-based Production Systems. *Procedia Manufacturing*. 11:939–948.

Phoon, S.Y., Yap, H.J., Taha, Z. & Pai, Y.S. 2017. Interactive solution approach for loop layout problem using virtual reality technology. *International Journal of Advanced Manufacturing Technology*. 89(5–8):2375–2385.

Raja, D., Bowman, D.A., Lucas, J. & North, C. 2004. Exploring the Benefits of Immersion in Abstract Information Visualization. in *Proceedings of the 8th Immersive Projection Technology Workshop*. 61–69.

Redelinghuys, A.J.H. 2020. *An Architecture for the Digital Twin of a Manufacturing Cell*. PhD (Eng) Dissertation. University of Stellenbosch. Stellenbosch.

Redelinghuys, A.J.H., Basson, A.H. & Kruger, K. 2018. A Six-Layer Digital Twin Architecture for a Manufacturing Cell. in *Service Orientation in Holonic and Multi-Agent Manufacturing, SOHOMA 2018. Studies in Computational Intelligence,* Vol. 803. Bergamo, Italy. 412–423.

Redelinghuys, A.J.H., Kruger, K. & Basson, A.H. 2019. A six-layer architecture for digital twins with aggregation. in *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing, SOHOMA 2019. Studies in Computational Intelligence,* Vol. 853. Valencia, Spain. 171–182.

Sekaran, S.C., Yap, H.J., Musa, S.N., Liew, K.E., Tan, C.H. & Aman, A. 2021. The implementation of virtual reality in digital factory—a comprehensive review. *International Journal of Advanced Manufacturing Technology*. 115(5–6):1349–1366.

Taylor, N., Human, C., Kruger, K., Bekker, A. & Basson, A. 2020. Comparison of digital twin development in manufacturing and maritime domains. in *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing, SOHOMA 2019. Studies in Computational Intelligence,* Vol. 853. Valencia, Spain. 158–170.

Uhlemann, T.H.J., Schock, C., Lehmann, C., Freiberger, S. & Steinhilper, R. 2017. The Digital Twin: Demonstrating the Potential of Real Time Data Acquisition in Production Systems. *Procedia Manufacturing*. 9:113–120.

Unity. 2017a. *Unity - Manual: GameObjects*. [Online], Available: https://docs.unity3d.com/560/Documentation/Manual/GameObjects.html [2022, August 03].

Unity. 2017b. *Unity - Manual: GameObject*. [Online], Available: https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html [2022, August 03].

Appendix A. DT System Operation

This section provides information for the initialisation and operation stages for the DT component and Shared Services component in the implemented case study DT system. First, the initialisation and operation stage for the DT component is provided, followed by the initialisation and operation phase for the Shared Services component.

A.1 Digital Twin Component Initialisation

The initialisation aspect is comprised of various processes that are followed when the DT component is to begin operating. The initialisation of the system makes use of a configuration file. This configuration file is similar to the one provided in Appendix B. The configuration file is a CSV format file that contains information for the various DTs in the system as well as the DT aggregation hierarchy structure. The configuration file contains information for the DTs including the name of the DT, the type of DT (building, precinct, or campus), the energy meters forming part of the DT (if it is a Building DTI or Precinct DTA), the location coordinates of the component, and the IP Address and the port which is used for the socket communication. On start-up, each DT is initialised with these various parameters stored in the configuration file.

The initialisation process for each DT in the system occurs concurrently as each DT runs on its own thread in the program. However, the initialisation process of some DTs is dependent on the completion of the initialisation process for subordinate DTs. For example, a Precinct DTA cannot fully initialise until all its Building DTIs have been fully initialised, because the Precinct DTA requires information from the Building DTIs below it to initialise correctly. The initialisation of a DT consists of four steps that must be followed in sequence for a DT and DT system to be initialised correctly. These being: the storing of subordinate component information (either energy meters or other DTs), a request for the subordinate component initialisation status, the carrying out of the Context generation service, and a notification that the DT has been initialised. The initialisation process is illustrated in Figure 31.

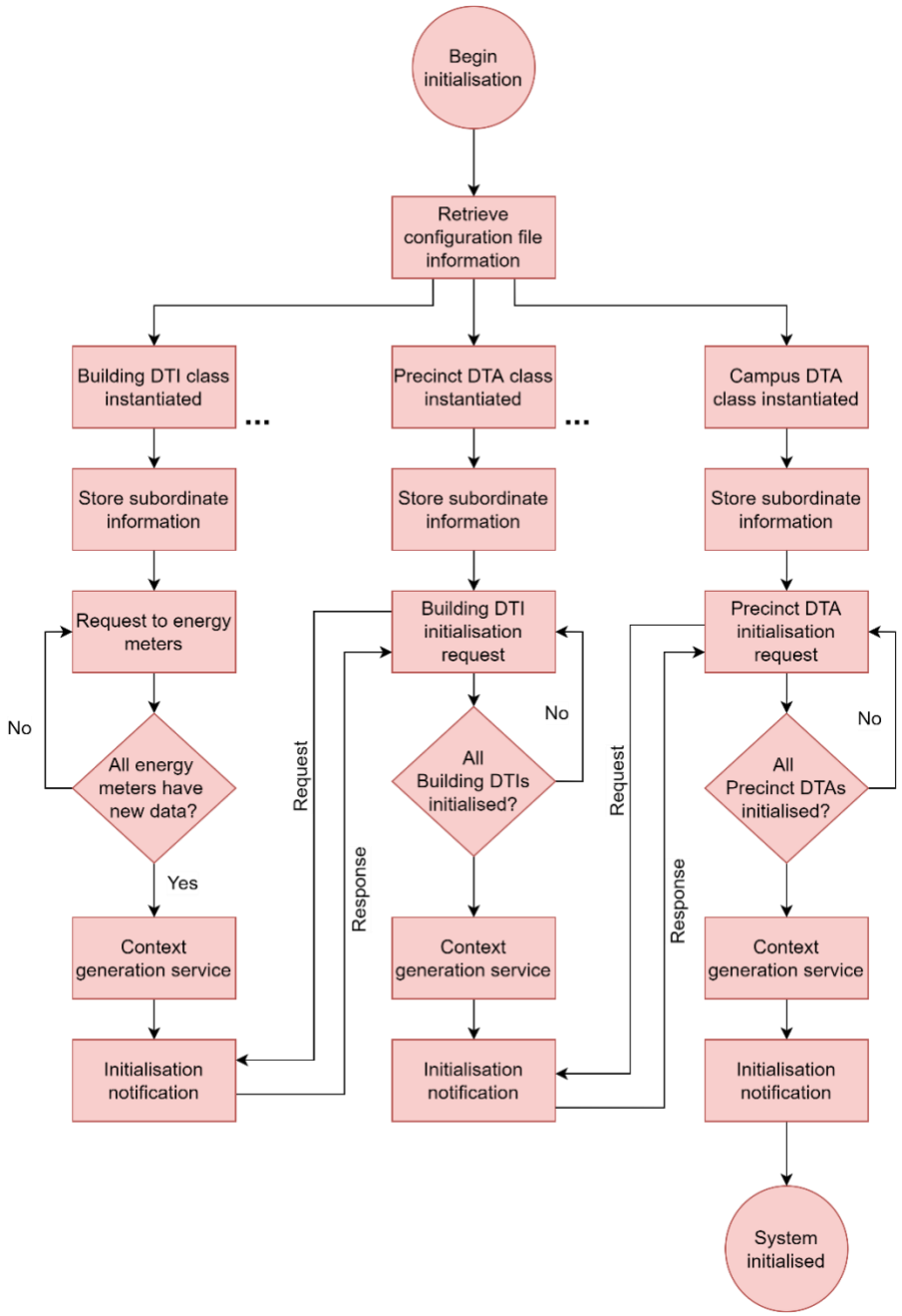


Figure 31: DT initialisation process

As mentioned previously, the configuration file contains information regarding the aggregation hierarchy of the DTs. This includes which energy meters form part of which buildings, buildings for which precincts, and precincts for the campus. This information is used to ensure that DTs communicate with the correct system components. The first aspect for initialisation is the storing of subordinate component information in the DT. For example, the Building DTIs and Precinct

DTAs contain information for which energy meter IDs must be used when accessing the API in Layer 3 for energy data. The Precinct DTAs stores the IP address and port number for the Building DTIs that they can communicate with. The Campus DTA stores the IP address and port number of the Precinct DTAs it can communicate with. This subordinate identification process is important because a DT remains idle while its subordinates initialise before it, itself, can initialise fully. Knowing which components to communicate with ensures that the system initialises in the correct order.

The second aspect of the process is the request made to a subordinate component for the status of initialisation for that component. This is required because of the dependencies some DTs have on the initialisation status of its components. Although the initialisation process for components occurs concurrently, the sequence of the initialisation process is that the Building DTIs are first fully initialised, followed by the Precinct DTAs, and finally the Campus DTA. Once a DT has completed the first aspect of storing subordinate DT information, it will then periodically make requests to its subordinate components as to whether the component is fully initialised or not. Once all the subordinate components respond to the request stating that they are initialised, the DT can then proceed to the third step.

If all subordinate components of a DT are initialised, the DT can then receive information from these components to carry out its own initialisation process. The DT must then use the Context generation service, described in Section 5.3.2.1.4, to further the initialisation process. This service receives the necessary information from the subordinate components to ensure that the correct information is generated and stored by the service. As no information has been stored for the DT, any information available from the subordinate components is deemed new information, and thus, all this new information is processed. An arbitrary start date is selected for the time frame of the historic information the DT must process. This time frame is from the selected start date to the real-world current date and time of starting the initialisation process. The start date is the same for all DT components to not result in errors. The chosen initialisation start date for the system is 01-01-2022 and is selected arbitrarily.

After the Context generation service has been carried out for the selected time frame of historic information, the DT is then initialised. The final step is for the DT to provide a notification that it is fully initialised. This occurs when the higher-level DT of a subordinate DT is completing the second stage of the initialisation process. When this request for the initialisation status is received by the subordinate DT component, the DT responds with indicating that it is fully initialised. The higher-level DT is then able to continue with the remaining steps of the initialisation process. The total time taken for the initialisation process to be completed depends on the number of DTs in the system and the selected starting date. Once

all DTs have carried out the initialisation process, the system is then correctly initialised and can enter the Operating phase.

The sample code provided below is the source code of the Bootstrap class that is used to initialise the various DTs in the DT system. The functions in the class use information from the CSV configuration file to instantiate the various DT classes in separate threads.

```
class Bootstrap
{
    // Declare list variables to store names of the DTs in the system
    private static LoadExcel excel = new LoadExcel();
    private static List<Building_DT.Building> buildingList = new
    List<Building_DT.Building>();
    private static List<Precinct_DT.Precinct> precinctList = new
    List<Precinct_DT.Precinct>();
    private static List<Campus_DT.Campus> campusList = new
    List<Campus_DT.Campus>();
    public static void Main(String[] args){
        // Invoke the DT initialisation function
        InitialiseDigitalTwins();
    }
    private static void InitialiseDigitalTwins()
    {
        // Load a list of the buildings from the configuration file
        buildingList = excel.LoadBuildingData();
        // Loop through the building list to start each Building DTI on a
        separate thread
        foreach(var build in buildingList)
        {
            Thread buildingThread = new Thread(build.InitialiseBuilding);
            buildingThread.Start();
        }
        // Load a list of the precincts from the configuration file
        precinctList = excel.LoadPrecinctData();
        // Loop through the precinct list to start each Precinct DTA on a
        separate thread
        foreach (var prec in precinctList)
        {
            Thread precinctThread = new Thread(prec.InitialisePrecinct);
            precinctThread.Start();
        }
        // Load a list of the campuses from the configuration file
        campusList = excel.LoadCampusData();
        // Loop through the building list to start each Campus DTA on a
        separate thread
        foreach (var camp in campusList)
        {
            Thread campusThread = new Thread(camp.InitialiseCampus);
            campusThread.Start();
        }
    }

}
```

A.2 Digital Twin Component Operation

After the DT component has been fully initialised, it enters the Operating phase. This Operating phase includes the processing of new information becoming available in the system, and the processing of requests made for the use of DT services. These operation functions are carried out periodically or as requested. The processing of new information functions are called every minute, and the processing of requests for DT services used are carried out as the requests are received. These requests are received by a DT's TCP/IP server.

Much like for the Initialisation phase of the system, the propagation of new information becoming available occurs at the Building DTI level first and propagates upwards in the DT aggregation hierarchy to the Campus DTA. Every minute, a Building DTI makes a request to the API in Layer 3 for data from its energy meters. If new energy data is available, then the Building DTI receives this new data. This availability of new data is identified by comparing the timestamp of the last received data to the current timestamp of the received data. If the timestamps are different, it means that new energy data is available. The Building DTI waits until all energy meters have new information available, after which, the DT carries out the Context generation service to update the DT with new energy data. A similar process occurs at a Precinct DTA level, where the Precinct DTA requests for any newly available energy information from its Building DTIs. A similar process, like with the Precinct DTA, occurs for the Campus DTA. The Campus DTA requires all the Precinct DTAs to have new information, and all the Precinct DTAs requires all the Building DTIs to have new information, which require all the energy meters to have new energy data. Using this process, as new information becomes available, the Context generation service is used at the different levels to ensure all components of the system are kept up to date.

The reason for checking for updated information every minute is that, although it is stated that energy meters typically record new data every five minutes, not all energy meters record data at the same time. If an energy meter does not take exactly five minutes, maybe six or seven minutes, other energy meters might have new data available already, and this could result in some energy meter data not being processed because of the frequency of the update functions being carried out. One minute is chosen arbitrarily as any faster would be unnecessary and any slower could result in some components not updating with the correct information which would mean that the DT system is not a correct reflection of reality.

The other aspect of the Operating phase is the processing of requests made for use of the DT services. In this case, the only service offered externally by a DT is the Mirror Service. This service is discussed in greater detail in Section 5.3.2.1.4. Unlike the updating functions of the Operating phase, this service processing

function occurs as the request is received by a DT from another system component. As mentioned previously, this request is received by a DTs TCP/IP server which is constantly awaiting communication from other components.

The software code provided below contains an example of the functions that are used to carry out the Mirror service for the Building DTI. Various requests can be made to the Mirror service, such as a request for energy usage information, the initialisation status of the Building DTI, and the subordinate DTs (which are none) for the Building DTI.

```
// Function to carry out different services offered by a Building DTI
public async Task<string> ServiceHandlerAsync(MessageModel message)
        {

            // Check if energy service is requested
            if (message.DataType == "Energy")
            {

                // Check if current energy usage is requested
                if (message.MessageType == "CurrentData")
                {

                    // Retrieve latest energy usage
                    var tempEnergy = await
                    ReturnLatestBuildingEnergyAsync();

                    // Create message with latest energy usage information
                    EnergyMeterModel temp = new
                    EnergyMeterModel(Building_name, 0, Latitude, Longitude,
                    tempEnergy, "Latest Reading");
                    List<EnergyMeterModel> tempEnergyList = new
                    List<EnergyMeterModel>();
                    tempEnergyList.Add(temp);
                    List<InformationModel> tempList =
                    GenerateInformationList(tempEnergyList);

                    // Convert message to a JSON string
                    var tempMess = JsonConvert.SerializeObject(tempList);
                    return tempMess;
                }

                // Check if average energy usage is requested
                else if (message.MessageType == "Averages")
                {

                    // Retrieve time period from request message
                    message.startDate =
                    utilities.ChangeDateFormat(message.startDate);
                    message.endDate =
                    utilities.ChangeDateFormat(message.endDate);

                    // Retrieve average energy usage information for time
                    period
```

```
            var temp = await
            ReturnBuildingEnergyAveragesAsync(utilities.GenerateDat
            eList(message.startDate, message.endDate,
            message.timePeriod));

            // Create message with average energy usage information
            and convert to JSON string
            var infoModelList = GenerateInformationList(temp);
            var response =
            JsonConvert.SerializeObject(infoModelList);
            return response;
        }
    }

    // Check if subordinate DT list is requested
    else if (message.DataType == "DigitalTwins")
    {
        if (message.MessageType == "ChildDTList")
        {

            // Retrieve list of subordinate DTs of Building DTI. A
            Building DTI has no subordinate DTs and thus the list is
            populated with “None”. Convert list message to JSON
            string
            ChildDTModel temp = new ChildDTModel("None", "None");
            var tempList = new List<ChildDTModel>();
            tempList.Add(temp);
            var tempMess = JsonConvert.SerializeObject(tempList);
            return tempMess;
        }
    }

    // Check if initialisation status is requested
    else if (message.DataType == "Initialisation")
    {
        if (message.MessageType == "Status")
        {

            // Return message with current initialisation status
            return Initialised.ToString();
        }
        else if (message.MessageType == "LatestEnergy")
        {

            // Return message with latest energy usage information
            var energy = await ReturnLatestBuildingEnergyAsync();
            return energy.ToString();
        }
        else if (message.MessageType == "Averages")
        {

            // Return message with average energy usage information
            for initialisation

            var energy = await
            GetTotalEnergyAsync(message.startDate);
            return energy.ToString();
```

```
            }
        }

        // Check if a DT operation value status is requested
        else if (message.DataType == "Operations")
        {
            if(message.MessageType == "LatestTimeStamp")
            {
                // Return message with latest updated timestamp of
                energy data
                return EnergyMeters[0].latest_timestamp;
            }
            else if (message.MessageType == "LatestEnergy")
            {
                // Return message with latest energy usage
                var energy = await ReturnLatestBuildingEnergyAsync();
                return energy.ToString();
            }
            else if (message.MessageType == "NewEnergyDataStatus")
            {
                // Return message indicatin new energy data is
                available
                return NewEnergyDataAvailable.ToString();
            }
            else if (message.MessageType == "ResetNewDataAvailable")
            {
                // Reset the “New Data Available” flag to check if new
                energy data is available.
                ResetDataAvailable();
                return "Complete";
            }
        }
                return "";

    }
```

A.3 Shared Services Component Initialisation

The initialisation phase contains a set of processes that are to be followed for the Shared Services component to be initialised correctly before it can enter the Operating phase. During this phase, the three service components mentioned in Section 5.3.3 (Service Gateway, DT Directory service, and Exploratory analytics service) must each be initialised in their respective manner. A bootstrap C# class, like with the DTs, is used to start the initialisation procedure for each of the service components. The bootstrap program makes use of a CSV configuration file for initialising the service components with the correct parameters. The bootstrap program provides each Shared Services component class with the correct information from the configuration file when being instantiated.

The initialisation process for the Service Gateway and Exploratory analytics is fairly simple. Based on the information received from the bootstrap program, the component will create a TCP/IP server socket to allow for communication to be

received from other components. With this, the service is also initialised with a list of information of the other services in the Shared Services component provided in the configuration file. This information is the name of the service component, and its communication address. The service component is able to access this list if they are required to communicate with another service.

The initialisation process for the DT Directory service has an added step of initialisation compared to the other two service components. Like the other two services, the DT Directory service creates a TCP/IP server socket to receive any incoming communication. The service also receives, from the bootstrap program, the necessary information about the other service components. In addition to these aspects, the DT Directory service is also initialised with information regarding the structure of the DT aggregation hierarchy. The service will request a list of subordinate DTs from each DT in the hierarchy. It receives the communication address for the DTs in the hierarchy from the configuration file. This list allows for information of the structure of the aggregation hierarchy to be stored in the DT Directory service. Requests can then be made to the service for information about DTs and their subordinate DTs.

A.4 Shared Services Component Operation

Once the Shared Services component has been initialised correctly, it then transitions into the Operating phase. During the Operating phase, each service is continuously awaiting incoming communication to its TCP/IP server socket. Any requests that are made from an external component, in this case the VR application, are made to the Service Gateway as this service is the communication channel for any external components outside of the DT system. The Service Gateway then processes this request and sends it to the correct service to request the necessary information and send the information back to the VR application. The other services are waiting for requests to be made either by the Service Gateway or another service component where they are processed and the correct information is transferred to the sender of the request. Some of the details of the interactions between services is shown in Section 5.3.4.

The sample code provided below contains the source code that was used by the Service Gateway during the operation of the Shared Services component. The Service Gateway receives requests from components and uses the function below to interpret what service has been requested or is required. A request is then made to this service for the desired information. The received information is then transferred to the component that made the original request.

```
// Function used to call other services in the Shared Services Component
public async Task<string> MessageHandlerAsync(string mes)
{
       string message = "";
       var tempMessage = JsonConvert.DeserializeObject<UIMessageModel>(mes);
       // Check if the DT Directory service is requested
       if (tempMessage.ServiceTag == "Directory")
       {
             // Loop through list of services to obtain communication
                address
             foreach(var service in servicesList)
             {
                  // Check if service matches the requested service's name
                  if (service.ServiceName == "Directory Service")
                  {
                        // Request and receive the subordinate DTs list for
                        a specified DT from the DT Directory service
                        var DTList = await myClient.sendMessageAsync(mes,
                        service.IP_Address, service.Port);
                        message = DTList;
                        break;
                  }
             }
       }
       // Check if the Exploratory Analytics service is requested
       else if (tempMessage.ServiceTag == "Exploratory")
       {
              // Loop through list of services to obtain communication
              address
              foreach (var service in servicesList)
              {
                 // Check if service matches the requested service's name
                 if (service.ServiceName == "Exploratory Service")
                 {
                     // Request and receive the specified energy usage
                     information for a DT
                     var response = await myClient.sendMessageAsync(mes,
                      service.IP_Address, service.Port);
                      message = response;
                      break;
                 }
              }
        }
        return message;

  }
```

Appendix B. Configuration File Sample

Figure 32 shows a sample of the CSV configuration file, loaded into Microsoft Excel, that was used in the case study implementations.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Campus	Precinct	Building	Meter Name	Meter Type	meter_id	description	Latitude	Longitude	IP_Addres	Port	Location	Starting date
2	Stellenbosch University	-	-	-	Energy	-	-	-33.9328	18.8644	127.0.0.1	8000	Local	2022-01-01 00:00:00
3	Stellenbosch University	Banghoek Toevoer	-	-	Energy	3037	Voertuigpoel Kiosk Hooftoevoer (Banhoek)	-33.9167	18.91667	127.0.0.1	8007	Local	2022-01-01 00:00:00
4	Stellenbosch University	Banghoek Toevoer	HIV Bestuur Gebou	-	Energy	3041	HIV Bestuur Gebou	-33.9167	18.91668	127.0.0.1	8003	Local	2022-01-01 00:00:00
5	Stellenbosch University	Banghoek Toevoer	Language Centre	-	Energy	3040	Language Centre	-33.9167	18.91668	127.0.0.1	8004	Local	2022-01-01 00:00:00
6	Stellenbosch University	Banghoek Toevoer	Matie Gemeenskapsdiens Luckhofskool	-	Energy	3038	Matie Gemeenskapsdiens Luckhofskool	-33.9167	18.91669	127.0.0.1	8005	Local	2022-01-01 00:00:00
7	Stellenbosch University	Banghoek Toevoer	Voertuigpoel	-	Energy	3039	Voertuigpoel	-33.9167	18.91669	127.0.0.1	8006	Local	2022-01-01 00:00:00
8	Stellenbosch University	Botmashoogte Toevoer	-	-	Energy	3758	Botmashoogte Toevoer	-33.9233	18.883	127.0.0.1	8008	Local	2022-01-01 00:00:00
9	Stellenbosch University	Coetzenburg 11 kV	-	-	Energy	3025	Coetzenburg 11 kV	-33.9394	18.86822	127.0.0.1	8026	Local	2022-01-01 00:00:00
10	Stellenbosch University	Coetzenburg 11 kV	Coetzenburg Atletiek Substasie	Coetzenburg Vodacom	Energy	2849	Coetzenburg Vodacom	-33.9406	18.87086	-	0	-	2022-01-01 00:00:00
11	Stellenbosch University	Coetzenburg 11 kV	Coetzenburg Atletiek Substasie	Coetzenburg Woonhuis 01	Energy	3965	Coetzenburg Woonhuis 01	-33.9385	18.87088	-	0	-	2022-01-01 00:00:00
12	Stellenbosch University	Coetzenburg 11 kV	Coetzenburg Atletiek Substasie	-	Energy	2964	Coetzenburg Atletiek Substasie	-33.9392	18.86975	127.0.0.1	8016	Local	2022-01-01 00:00:00
13	Stellenbosch University	Coetzenburg 11 kV	Danie Craven Rugby Stadion	-	Energy	2955	Danie Craven Rugby Stadion	-33.9405	18.87292	127.0.0.1	8017	Local	2022-01-01 00:00:00
14	Stellenbosch University	Coetzenburg 11 kV	Swembad Minisub	Swembadpompe	Energy	2639	Swembadpompe	-33.9406	18.87086	-	0	-	2022-01-01 00:00:00
15	Stellenbosch University	Coetzenburg 11 kV	Swembad Minisub	-	Energy	2856	Swembad Minisub	-33.9407	18.8709	127.0.0.1	8025	Local	2022-01-01 00:00:00
16	Stellenbosch University	Erica 11 kV	-	-	Energy	2739	Erica 11 kV	-33.9347	18.87145	127.0.0.1	8059	Local	2022-01-01 00:00:00
17	Stellenbosch University	Erica 11 kV	Erica Koshuis	-	Energy	2731	Erica Koshuis	-33.9354	18.872	127.0.0.1	8027	Local	2022-01-01 00:00:00
18	Stellenbosch University	Erica 11 kV	Heemstede Hooftoevoer	Lydia Toevoer	Energy	3137	VM	-	-	-	0	-	2022-01-01 00:00:00
19	Stellenbosch University	Erica 11 kV	Heemstede Hooftoevoer	Lydia Toevoer	Energy	2959	Lydia Toevoer	-	-	-	0	-	2022-01-01 00:00:00
20	Stellenbosch University	Erica 11 kV	Heemstede Hooftoevoer	Studente Gesondheid	Energy	3759	VM	-	-	-	0	-	2022-01-01 00:00:00
21	Stellenbosch University	Erica 11 kV	Heemstede Hooftoevoer	-	Energy	2997	Heemstede Hooftoevoer	-33.9351	18.86858	-	0	-	2022-01-01 00:00:00
22	Stellenbosch University	Erica 11 kV	Metanoia Hooftoevoer	Metanoia Heatpump	Energy	7121	Western Cape - Metanoia - Heat pumps	-	-	-	0	-	2022-01-01 00:00:00
23	Stellenbosch University	Erica 11 kV	Metanoia Hooftoevoer	Metanoia Kombuis	Energy	2743	Metanoia Kombuis	-	-	-	0	-	2022-01-01 00:00:00
24	Stellenbosch University	Erica 11 kV	Metanoia Hooftoevoer	Metanoia Koshuis - VM Remainder	Energy	5455	VM	-	-	-	0	-	2022-01-01 00:00:00
25	Stellenbosch University	Erica 11 kV	Metanoia Hooftoevoer	-	Energy	2742	Metanoia Hooftoevoer	-33.9327	18.87169	127.0.0.1	8044	Local	2022-01-01 00:00:00
26	Stellenbosch University	Erica 11 kV	Minerva + Nerina Hoof Toevoer	Minerva + Nerina Hittepompe	Energy	2960	Minerva+Nerina Hittepompe	-	-	-	0	-	2022-01-01 00:00:00
27	Stellenbosch University	Erica 11 kV	Serruria Koshuis	-	Energy	2733	Serruria Koshuis	-33.9351	18.87298	127.0.0.1	8053	Local	2022-01-01 00:00:00
28	Stellenbosch University	Erica 11 kV	Tienie Louw Hooftoevoer	Tienie Louw Hittepompe	Energy	2969	Tienie Louw Hittepompe	-	-	-	0	-	2022-01-01 00:00:00
29	Stellenbosch University	Erica 11 kV	Tienie Louw Hooftoevoer	Tienie Louw Kombuis (Noodkrag)	Energy	2953	Tienie Louw Kombuis (Noodkrag)	-	-	-	0	-	2022-01-01 00:00:00
30	Stellenbosch University	Erica 11 kV	Tienie Louw Hooftoevoer	Tienie Louw Kombuis Warmwater	Energy	2952	Tienie Louw Kombuis Warmwater	-	-	-	0	-	2022-01-01 00:00:00
31	Stellenbosch University	Erica 11 kV	Tienie Louw Hooftoevoer	Tienie Louw Woonstelle	Energy	2951	Tienie Louw Woonstelle	-	-	-	0	-	2022-01-01 00:00:00
32	Stellenbosch University	Erica 11 kV	Tienie Louw Hooftoevoer	-	Energy	2970	Tienie Louw Hooftoevoer	-33.9348	18.87179	127.0.0.1	8058	Local	2022-01-01 00:00:00
33	Stellenbosch University	Helderberg 11 kV	-	-	Energy	3028	Helderberg 11 kV	-33.9402	18.85938	127.0.0.1	8063	Local	2022-01-01 00:00:00
34	Stellenbosch University	Helderberg 11 kV	Helderberg Hittepompe	-	Energy	2946	Helderberg Hittepompe	-33.9402	18.85938	127.0.0.1	8060	Local	2022-01-01 00:00:00

Figure 32: Configuration file sample

Appendix C. Virtual Reality Application

This section contains additional information about the developed VR application used in the *DT implementation* and *Non-DT implementation* of the FM case study. The Unity development environment is briefly discussed and showcased. The VR application UI developed for use in the VR environments is shown and explained. Various examples of using the VR application are then provided.

C.1 Unity Environment

Figure 33 shows the Unity development environment that is used to develop the VR applications. This figure shows the various aspects of the development environment. In addition to this environment, Visual Studio Code was used to develop the custom C# scripts that are attached as Components to the GameObjects in the application.

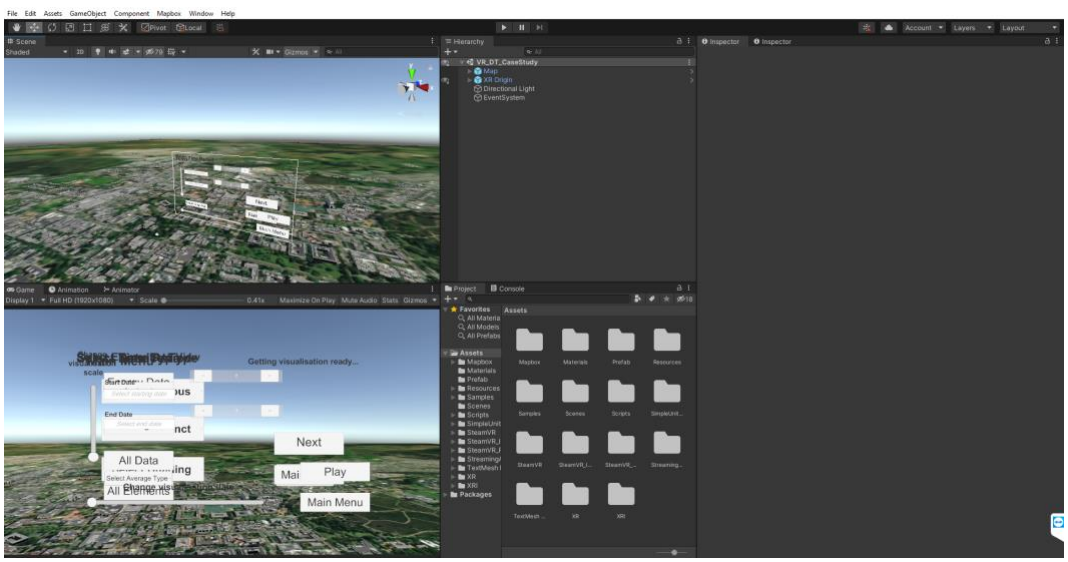


Figure 33: Unity VR application software environment

Figure 34 shows the scene view where an overview of the various GameObjects present in the environment are displayed and can be adjusted and modified as desired. This figure depicts the map of Stellenbosch that is used in the visualisation, as well as the UI GameObject that a user interacts with in VR. This is not the perspective that the user has in VR, rather it is the developer's perspective for creating the VR environment.

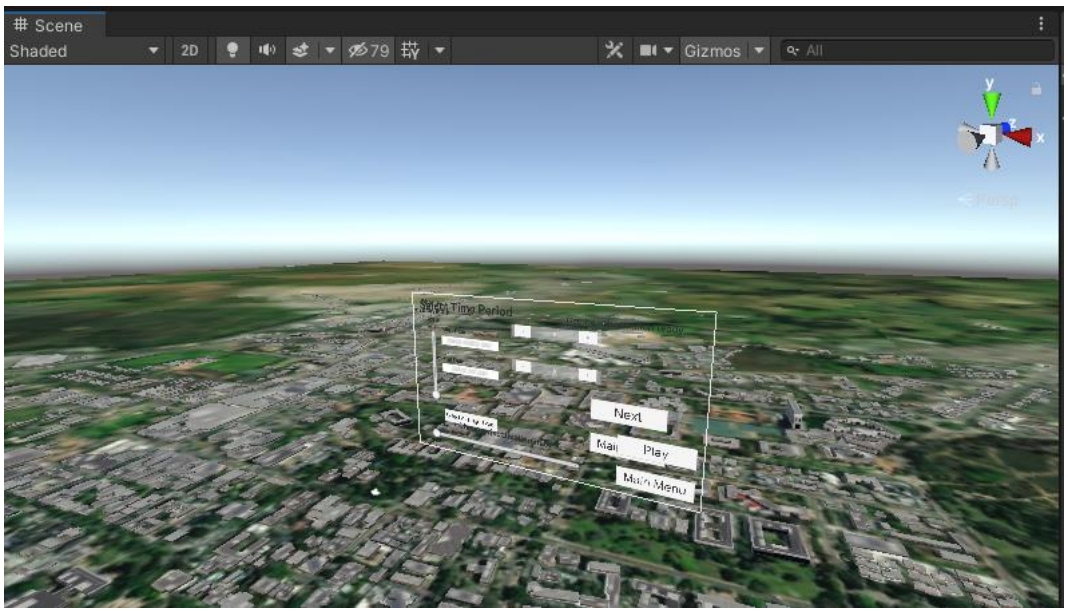


Figure 34: VR application scene view

Figure 35 shows the tabs that contain the GameObjects and the Components attached to those GameObjects. The “Hierarchy” tab indicates the various GameObjects that are present in the current environment. These GameObjects include the Stellenbosch Map, the user’s GameObject denoted as “XR Origin”, and a lighting GameObject to light the VR environment. Below the “Hierarchy” tab is the “Project” tab that contains all of the folders and files that contain scripts or models used in the VR application. The “Inspector” tab, on the right of the “Hierarchy” and “Project” tabs, shows the various Components, and their details, attached to a GameObject. In this figure, the displayed Components are for the user’s GameObject.

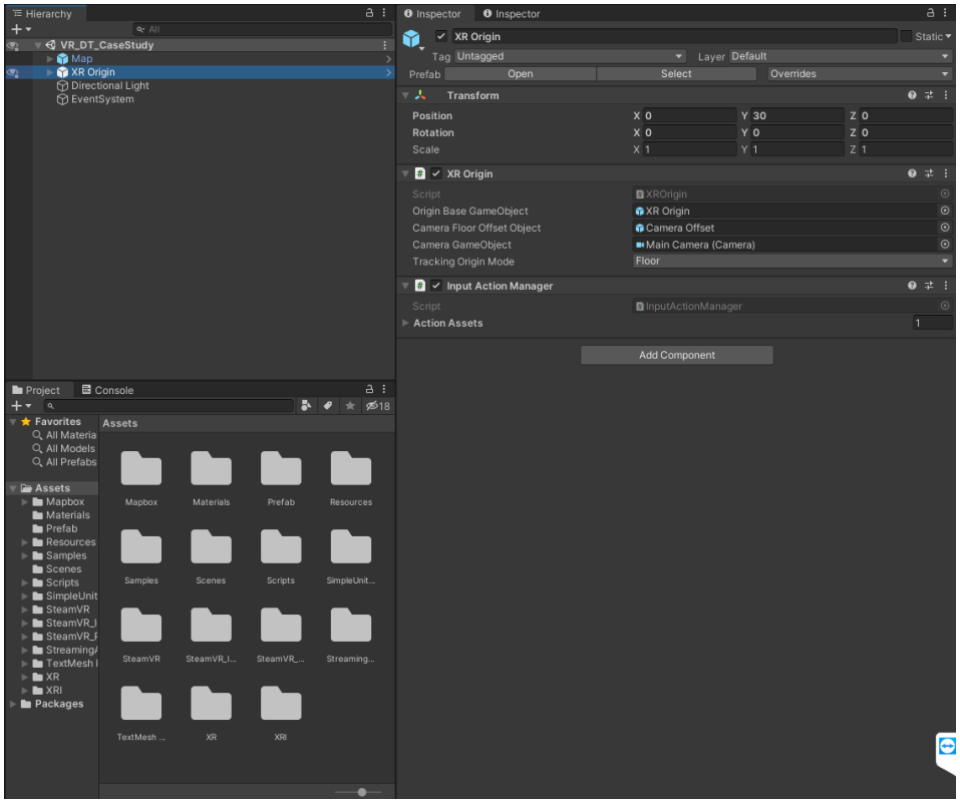


Figure 35: VR application GameObjects and Components

C.2 VR Application UI

The user can select elements on the UI using the VR controllers. Figure 36 shows the UI with the cascading menus and the various selections that can be made. Figure 36a is the main menu of the UI, here the user is able to select which span of reality they want energy information from, either the campus, a precinct, or a building. Figure 36b shows the menu of which elements in the span of reality the user would like to visualise. This could either be at the building level, the precinct level, campus level, or all levels depending on what span of reality was selected. If a building was selected as the span of reality, the only possible elements to visualise would be the building, whereas if the campus was selected, the possible elements to visualise include elements at a building level, precinct level or campus level. Figure 36c is the selection of which data type to visualise. In this case only energy data is used and is the only selection that can be made here, but provision has been made for the addition of other data types. Figure 36d shows a selection of the service type for visualising desired information. The user is able to select either the latest energy usage or they are able to select to visualise average usage information (day, monthly, or yearly averages). Figure 36e is the menu shown after selecting the “Averages” option in Figure 36d. The menu is not displayed if the “Latest” option is selected. Figure 36e allows the user to input a start date and end

date of the time span of information they would like to visualise. They are also able to select what type of average usage information they would like to visualise (either day, month, or year average usage). Figure 36f is the visualisation menu. This visualisation menu allows the user to adjust the visualisation as they desire. In all instances, the user is able to adjust the scale of the visualisation that is presented to them. In the case of visualising average usage information, the user is able to "Play" through the information presented to them or, for example, manually adjust which day's average usage information they would like to visualise.

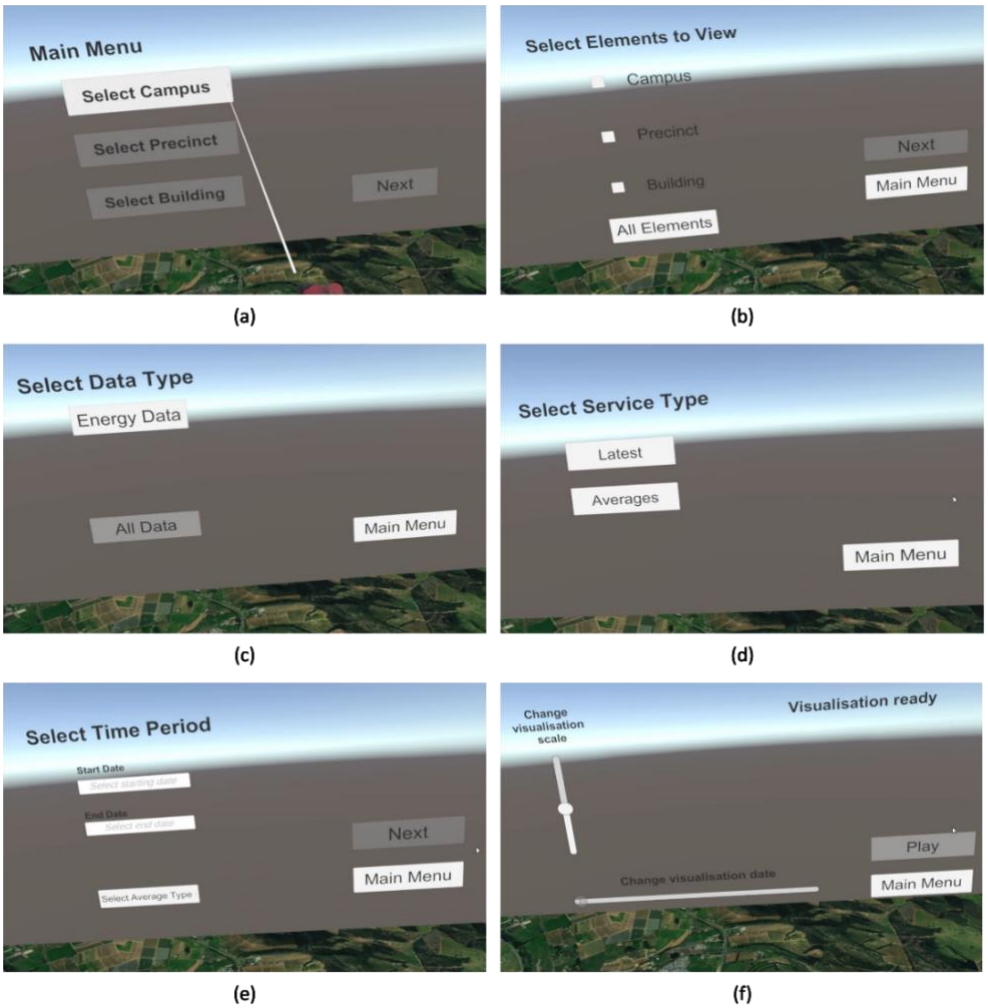
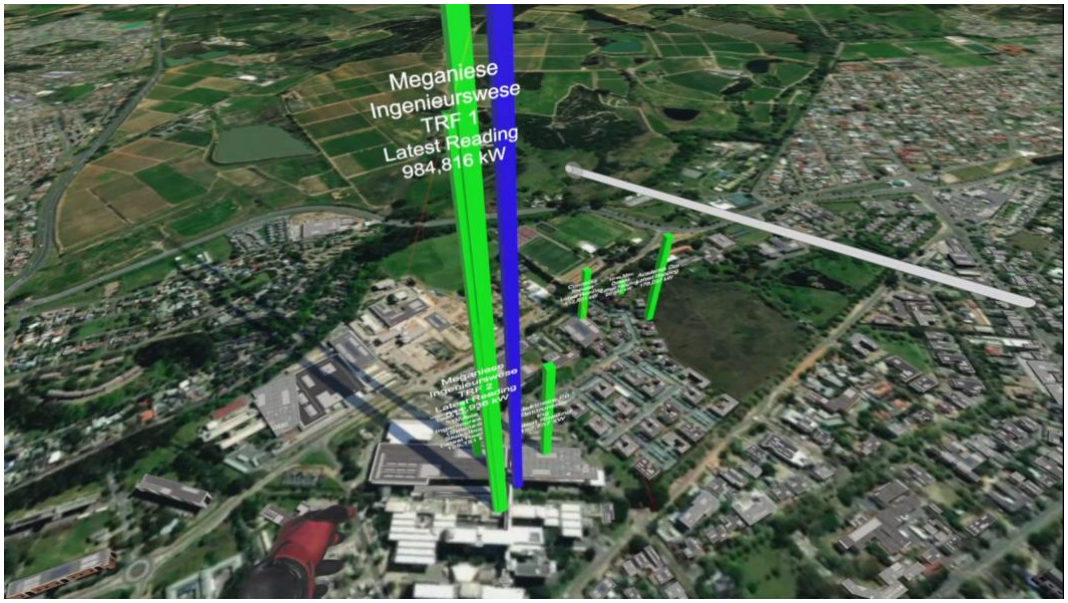


Figure 36: VR UI menus

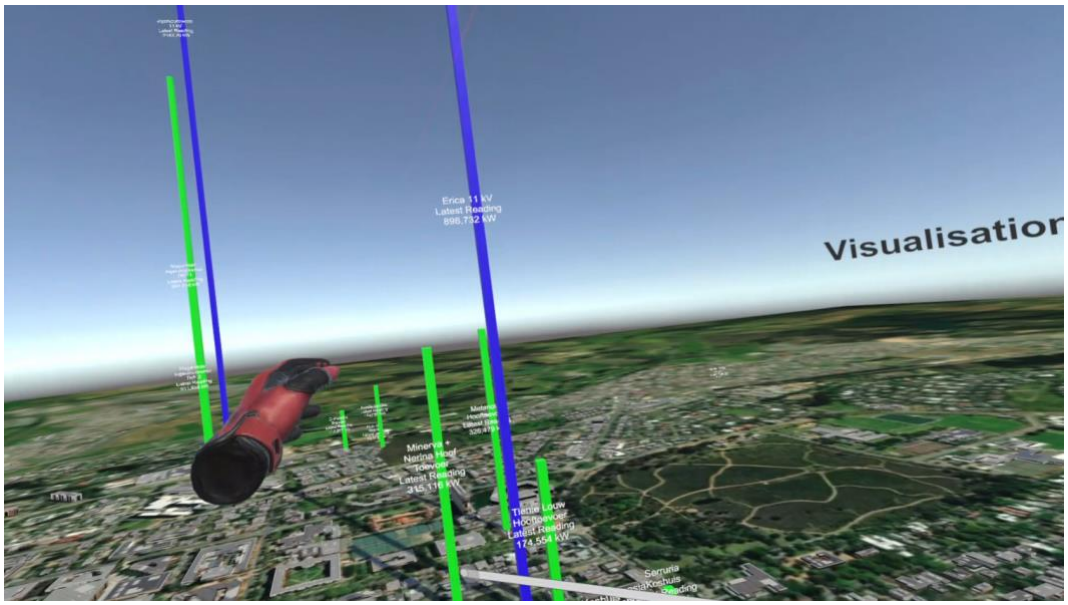
The functionality of creating the request message based on the selections by the user in the UI is encapsulated in the UI functions component of the architecture shown in Figure 12 and Figure 14. This component also contains the functionality for displaying the correct menu after one another as the user works through the UI.

C.3 VR Application Examples

This section provides examples of a user in VR using the VR application to visualise energy information of Stellenbosch Campus. Figure 37 shows a user visualising the latest energy reading for various precincts on the campus and buildings within those precincts.



(a)



(b)

Figure 37: Visualising latest precinct and building energy usage information

Figure 38 shows an example of the user selecting the end date for a time period to make a request to visualise the average energy usage for a facility. After selecting the start date, the user is then required to select the end date of the time period and the type of average energy usage they would like to visualise (either daily, monthly, or yearly usage).

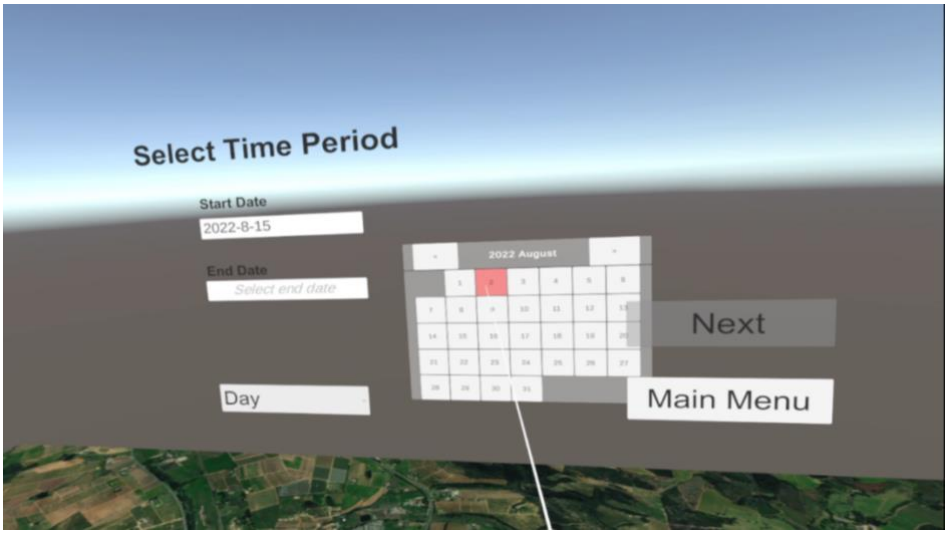


Figure 38: Selecting time period start date

Figure 39 is an example of a user making a request to visualise new information after they have already visualised the previously requested energy information. The previously requested information that has been displayed to the user is shown in the background, and the foreground shows the user using the UI and menus to make a request for new information to be displayed.

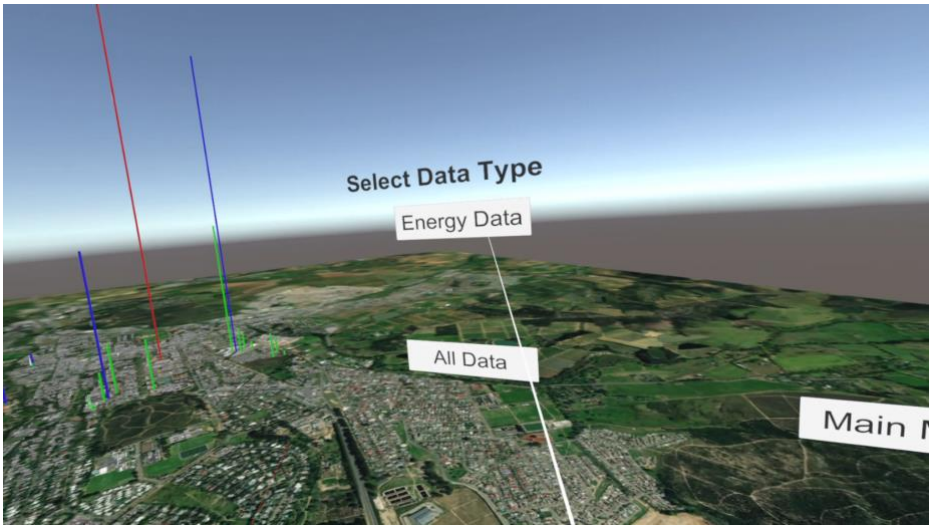
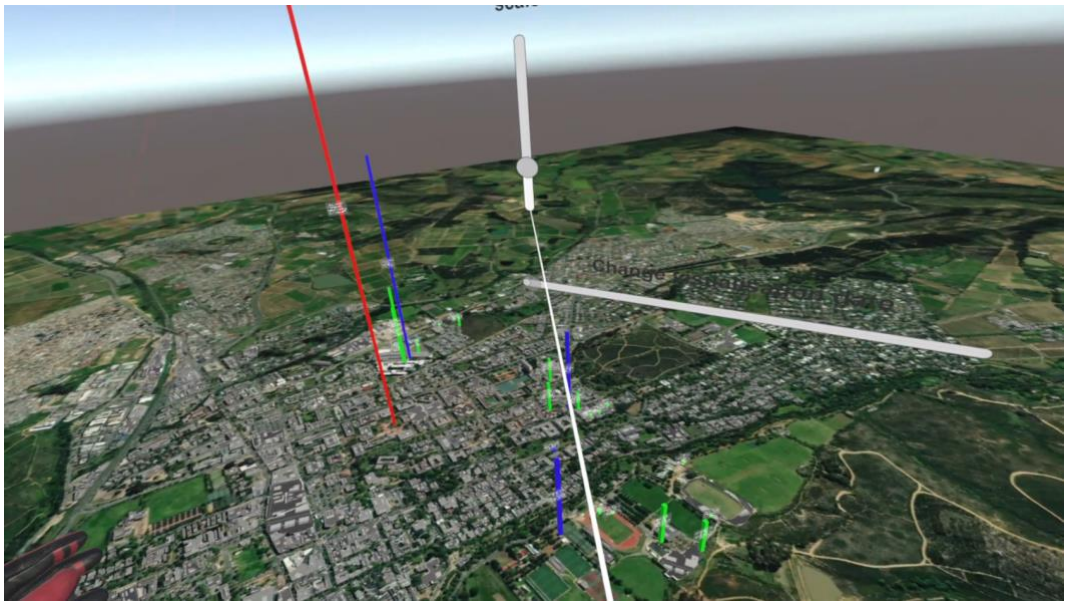
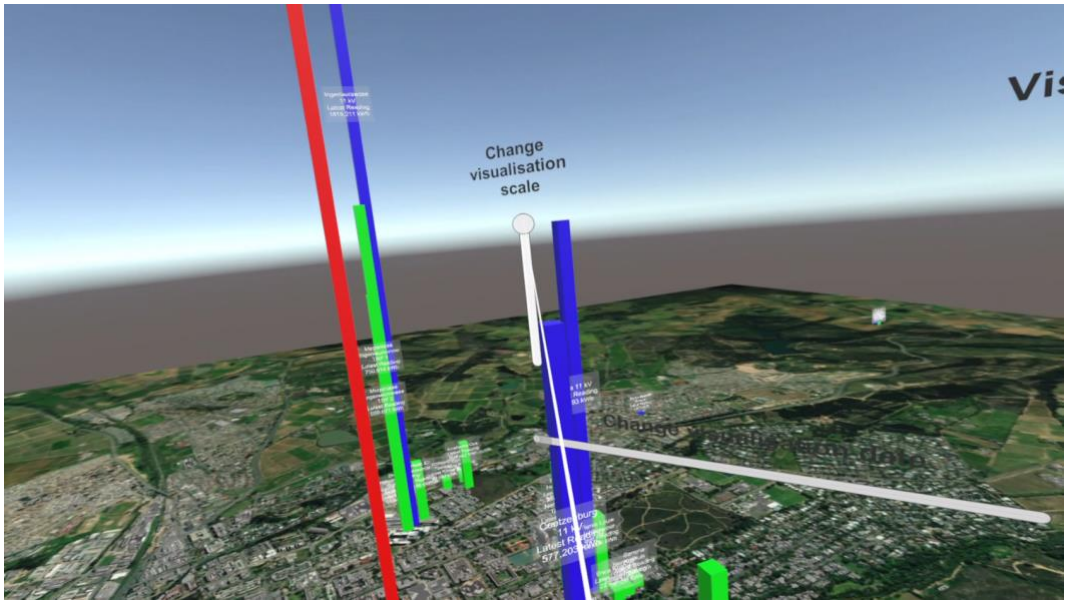


Figure 39: Requesting new energy information

Figure 40 shows an example of a user adjusting the scale of the energy usage information they are visualising. Adjusting the visualisation scale does not affect the scale of the map, rather, the size of the displayed coloured columns is adjusted according to the scale selected by the user.



(a)



(b)

Figure 40: Adjusting visualisation scale

Appendix D. Additional Evaluation Results

This section provides additional evaluation results of the latency and computer resource utilisation experiments carried out for the *DT implementation* and *Non-DT implementation*. Results are also provided for the *Non-DT implementation* with the modification of using a local database to access the information.

Figure 41 shows the latency results for the *DT implementation* where a range of data points were requested from each element in the system. The results show, as expected, that the latency of the system increased with the number of elements in the system, as well as the number of data points requested per element. The latencies in the results were relatively low compared to the latencies of the *Non-DT implementation*.

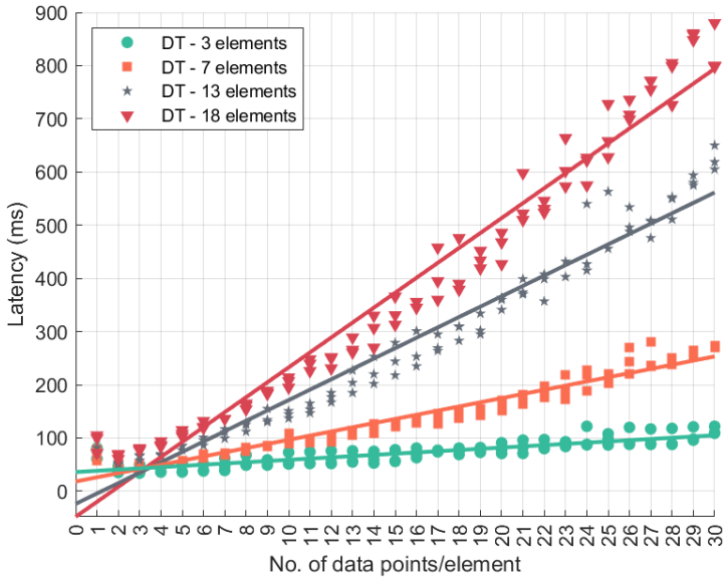


Figure 41: DT implementation latency results

Figure 42 provides the results of the RAM usage experiment for the *DT implementation*. The results indicate that the amount of RAM used by the *DT implementation* was fairly constant as more system elements were added and number of requested data points per element was increased.

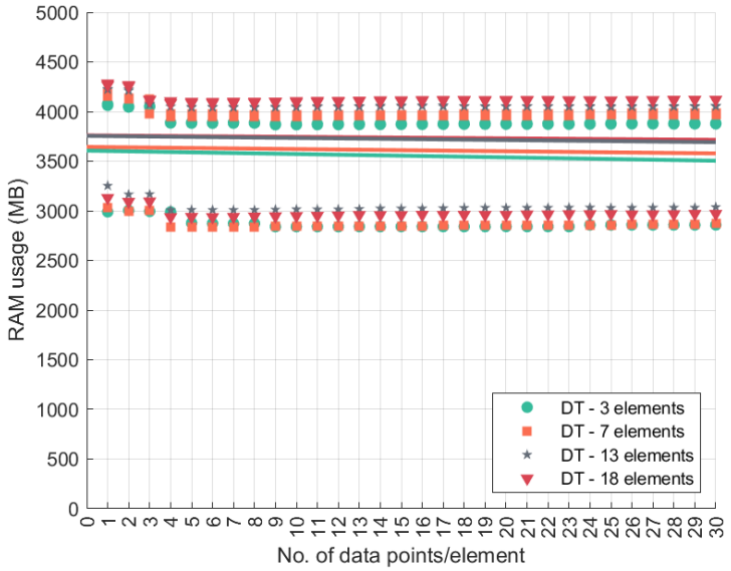


Figure 42: DT implementation RAM usage results

Figure 43 shows the results for the CPU usage experiment for the *DT implementation*. The results indicate that the CPU usage remained constant, like the RAM usage, as elements were added to the system and requested data points per element was increased.

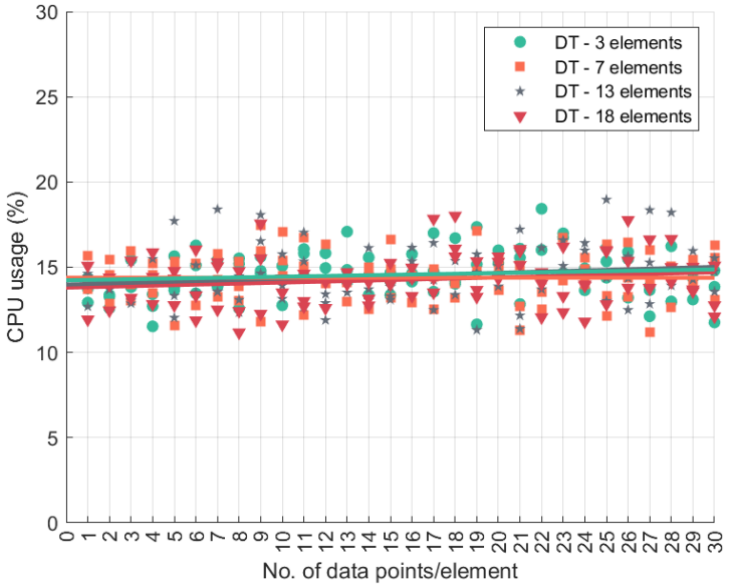


Figure 43: DT implementation CPU usage results

Figure 44 provides the results for the latency experiment for the *Non-DT implementation*. These latency results were in the degree of tens of seconds which was significantly higher than the latency results for the *DT implementation*. The

results indicate that the latency of the system increased almost linearly as more system elements were added and more data points per element were requested. This was a similar trend to the *DT implementation* latency results.

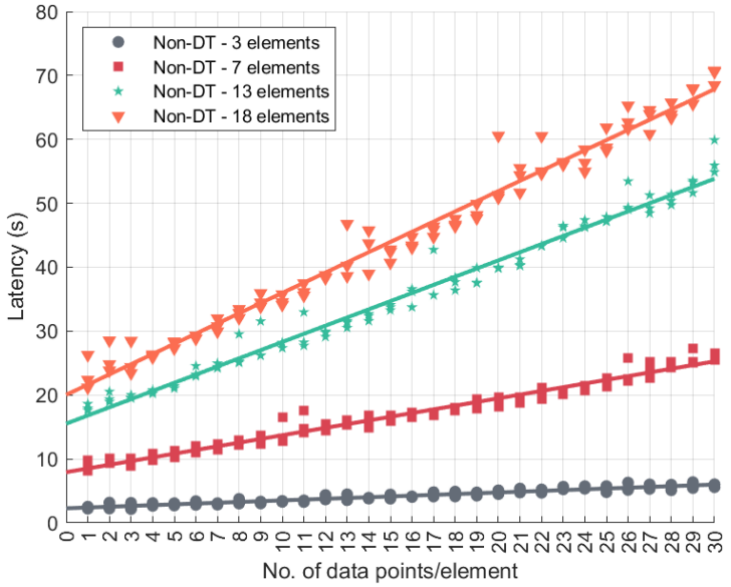


Figure 44: Non-DT implementation latency results

Figure 45 shows the RAM usage experiment results for the *Non-DT implementation*. The results indicate that the RAM usage remained constant and did not increase as more system elements were added and more data points per element were requested.

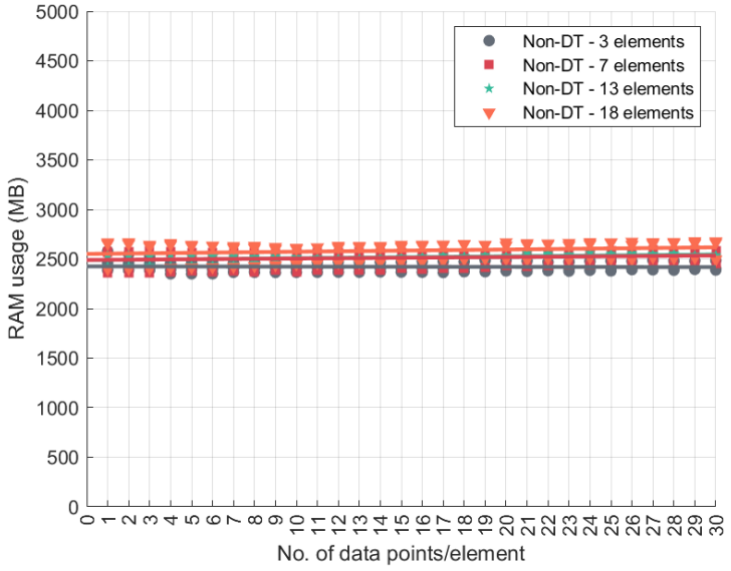


Figure 45: Non-DT implementation RAM usage results

Figure 46 shows the results for the CPU usage experiment for the *Non-DT implementation*. These results show that the CPU usage remained constant, at approximately between 11% and 14%, for the duration of the experiment. The addition of more system elements and requesting more data points per element did not have a significant impact on the CPU usage of the system.

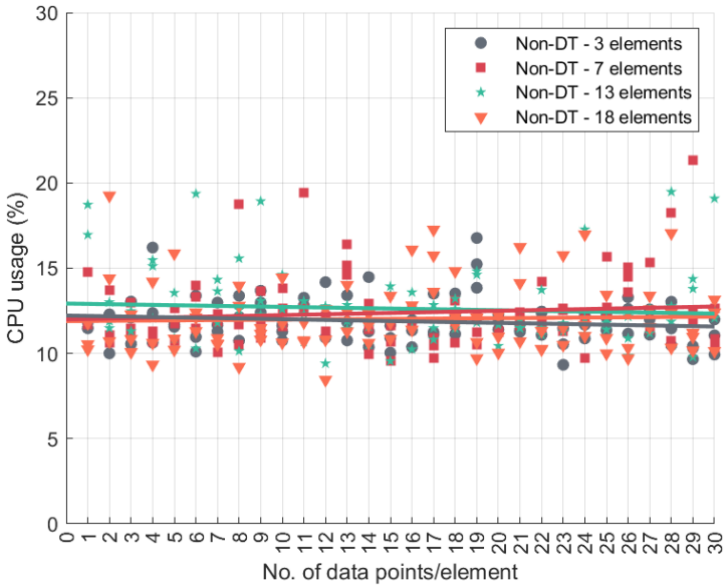


Figure 46: Non-DT implementation CPU usage results

It is mentioned in Section 6.3.1 that a modification was made to the *Non-DT implementation* to conduct further evaluation of the *Non-DT implementation's* latency. This modification was the use of a local database to store the requested energy usage data. The API originally used in the implementation was removed and the local database was then queried for the same energy data that was available using the API. Figure 47 provides the latency experiment results for the *Non-DT implementation* with this modification. The results show a similar trend as in Figure 44, however, the degree of the latency with this modification was halved in comparison to the original *Non-DT implementation*. The degree of latency was still significantly higher than for the *DT implementation*.

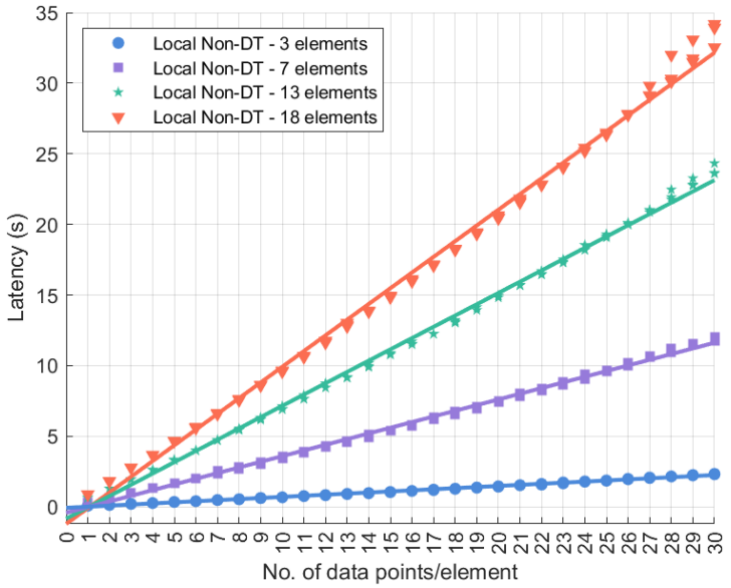


Figure 47: Local database Non-DT implementation latency results

Figure 48 shows the RAM usage experiment results of the *Non-DT implementation* with this local database modification. The results indicate that the RAM usage increased slightly with more elements in the system, but remained constant as more data points were requested per element.

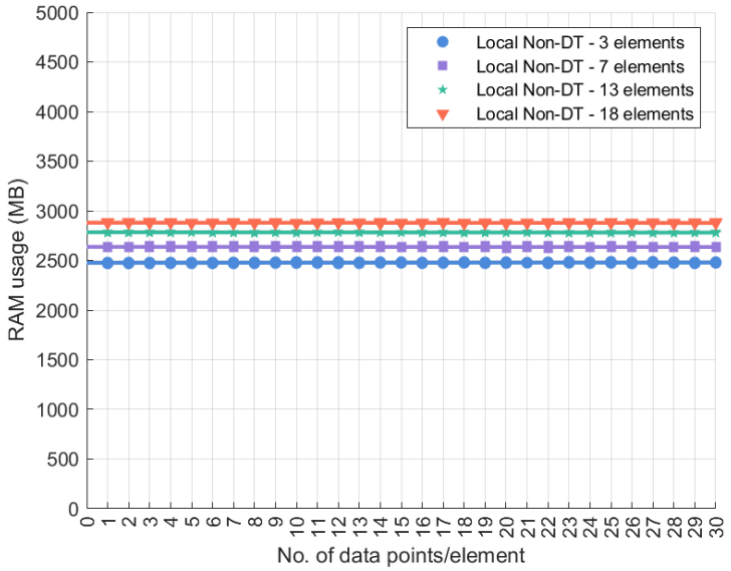


Figure 48: Local database Non-DT implementation RAM usage results

Figure 49 provides the results of the CPU usage experiment for the *Non-DT implementation* with the local database modification. The results show that when the system only had three elements that the CPU usage was lower than for the

other scenarios. The CPU usage did not, however, change significantly when more data points are requested per element.

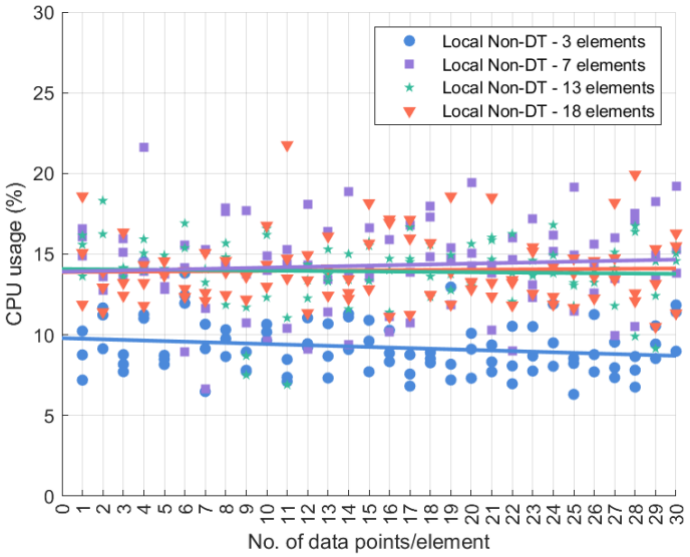


Figure 49: Local database Non-DT implementation CPU usage results

Figure 50 to Figure 52 provide results for the latency and computer resource utilisation experiments for Scenario 4 where only a single data point is requested for a building element in the system. Figure 50 shows the results for the latency experiment for this scenario. Much like the other results, the latencies for the *Non-DT implementation* with and without the local database modification was significantly higher than for the *DT implementation*. The degree of latency for the *DT implementation* was in the degree of milliseconds, and the degree of latency, in both instances, for the *Non-DT implementation* was in the degree of seconds.

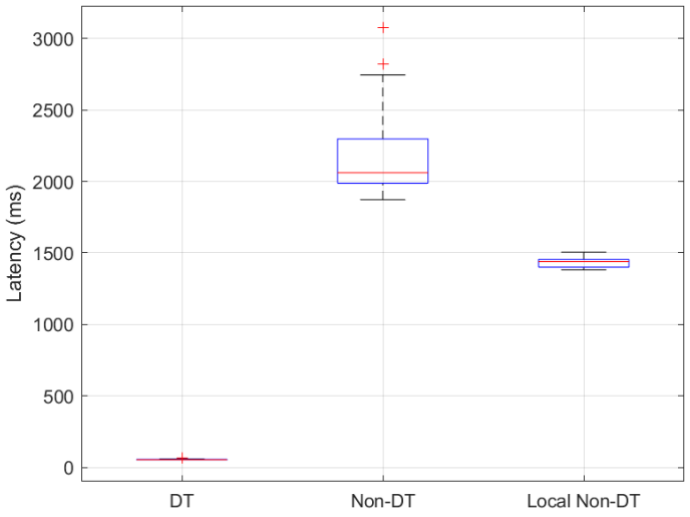


Figure 50: Single data point request latency results

Figure 51 provides the results for the RAM usage of the implementations for the scenario. The results show that the average RAM usage between the implementations was not significantly different. The *DT implementation* used more RAM which was expected due to the independently operating components, unlike the *Non-DT implementation* that only had the VR application.

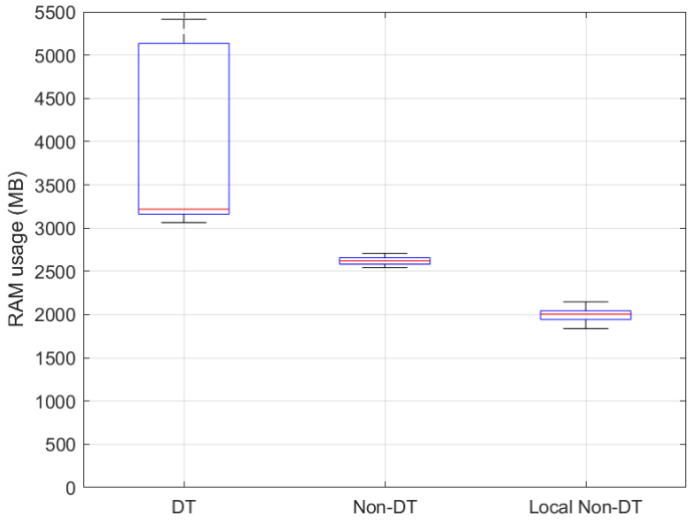


Figure 51: Single data point request RAM usage results

Figure 52 provides the CPU usage experiment results for the scenario for the implementations. The results show that the CPU usage did not differ drastically between the implementations and that the average CPU usage of the implementations was between 10% and 15%.

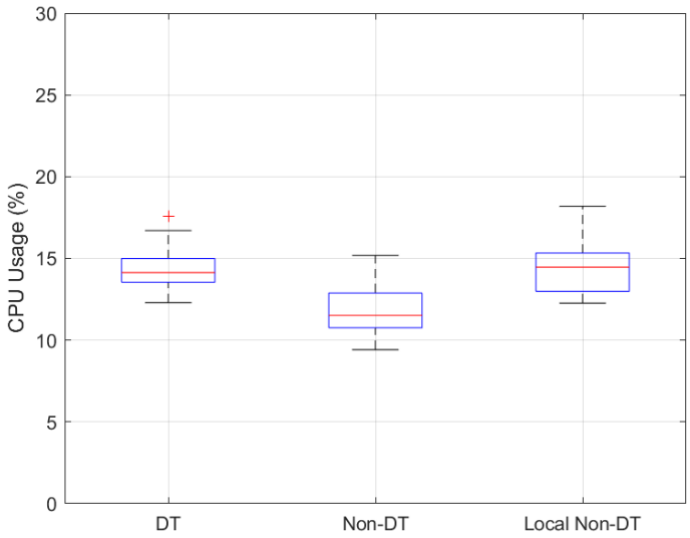


Figure 52: Single data point request CPU usage results