DISTRIBUTED SIMULATION CLOUD ARCHITECTURE FOR EXPERIMENTATION (DICE)

A Thesis Submitted for the Degree of
Doctor of Philosophy (PhD)

By
Nura Tijjani Abubakar

Department of Computer Science

June 2021

This page is intentionally left blank for printing purpose.

Abstract

Distributed Simulation (DS) is a method in operational system analysis that has gained interest due to its claimed benefits, including model reusability and interoperability. DS allows the exploitation of geographically distributed resources such as equipment and people. However, the cost of high-performance computing resources, technical skills, and special training required to design, develop, and use DS is an ongoing concern. These are the long-standing challenges that have prevented the broader adoption of parallel and distributed simulation technology. Cloud computing offers an alternative approach to address these issues using the pay-as-you-go economic model, eliminating considerable investments in the required hardware and software.

DS has the potential to benefit Modelling and Simulation (M&S). Nevertheless, relatively limited attention has focused on developing a framework and deployment architecture to enable analysts to run DS experimentation on the cloud. A more in-depth study is needed to understand how modellers will run cloud-based DS and how the cloud platforms will perform with variant parameter inputs. The literature established that DS development is a complex process and requires expertise with immense courage to undertake. This thesis investigated how the cloud can be used to connect geographically distributed federates to analyse operational systems. To achieve that, a deployment architecture is proposed and experimented with potentials benefit modellers. Furthermore, a development methodology is proposed to guide analysts at every step of the cloud-based distributed simulation (CBDS) implementation - from concept to cloud execution.

The experimental results indicate that it is feasible to connect and run geographically distributed simulation using cloud infrastructure. The research further finds that running a federation on a single cloud performs differently than federation execution on multiple cloud platforms. The significant differences are primarily attributed to how each cloud service provider handles network traffic and the overall communication overheads found on the Internet. This research has contributed to the CBDS approach and focussed more on analysing operational research systems by less technical modellers. The principal contributions of this work include a proposed scalable CBDS deployment architecture - **DIstributed simulation Cloud Architecture for Experimentation (DICE)**. DICE becomes the foundation of this research, providing technical specifications and guiding analysts on deploying DS on various cloud platforms.

Acknowledgements

I owe great thanks to all the people who have supported me on a more personal level, starting with big *"thank you"* to; **Professor Simon JE Taylor**, my principal supervisor for always sharing the enthusiasm and excitements and still beating me to it, with wilder ideas. Always helping me when simulation logic made my head hurt and making my head tingle with hope and visions. He is not only a fantastic supervisor but a good and jovial friend. **Dr Anastasia Anagnostou**, my supervisor for teaching and guiding me through the directions I want to follow on research and alternative ways to solving problems arising from simulations. She has been by my side from day one as a supporting researcher until she finally becomes my official second supervisor. Congratulations for being employed as a lecturer in the department after going through fierce and tough competitive selection processes – your hard work paid.

I would also like to seize the opportunity to thank all my office mates, staff and administrators at the computer science department and the Brunel Graduate School. They are legion, and I could not possibly mention them all.

The PhD journey has come to an end but wouldn't have been possible without the love and support of my family and friends. In particular, I want to thank my lovely wife and kids who bore the brunt of my absence during the first 13 months.

I want to acknowledge the **Petroleum Technology Development Fund (PTDF) Nigeria** for full funding, bearing all the financials involved in my studies. Without prestigious scholarship supporting education like you, students such as myself would be unable to pursue advanced degrees in reputable universities for better future.

Finally, I want to thank Brunel University London for their commitment towards students and researcher's well-being throughput study lifetime.

Declaration

I declare that this work has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous review or degree. Except where stated otherwise by reference or acknowledgement, the work presented is entirely my own.

Parts of this work have been presented in conferences in Brunel University and will be in the forthcoming one in and outside the UK.

~ Nura Tijjani Abubakar

Publications

Parts of this thesis has been disseminated as follows.

Abubakar, N. T (2020). *Investigating Cloud-based Distributed Simulation (CBDS) for Large-Scale Systems.* In: Proceedings of the 2020 Winter Simulation Conference, IEEE, Orlando, Florida, USA.

Taylor, S.J.E., Anagnostou, A., Abubakar, N. T., Kiss, T., Deslauriers, J., Terstyanszky, G., Kacsuk, P., Kovacs, J., Kite, S., Pattison, G. and Petry, J. (2020). *Innovations in Simulation: Experiences with Cloud-Based Simulation Experimentation.* In: Proceedings of the 2020 Winter Simulation Conference, IEEE, Orlando, Florida, USA.

A Anagnostou, SJE Taylor, NT Abubakar, T Kiss, J Deslauriers, G Gesmier, G Terstyanszky, P Kacsuk, J Kovacs (2019), *Towards A Deadline-Based Simulation Experimentation Framework Using Micro-Services Auto-Scaling Approach.* In: Proceedings of the 2019 Winter Simulation Conference, IEEE, Orlando, Washington, DC, USA.

NT Abubakar, SJE Taylor, A Anagnostou (2018), *Cloud-based Modeling & Simulation: Introducing the Distributed Simulation Layer*. In: Proceedings of the 2018 Winter Simulation Conference, IEEE, Gothenburg, Sweden.

Abbreviations

AaaS	Analysis as a Service
ACD	Activity Cycle Diagram
ADSO	Australian Defence Simulation Office
AES	Ambulance Emergency Service
ALS	Advanced Life Support
AMHS	Automated Material Handling System
ASP	Application Service Provider
BLS	Basic Life Support
CBDS	Cloud-Based Distributed Simulation
CBMS	Cloud-Based Modelling & Simulation
CBS	Cloud-Based Simulation
CDDL	Common Development and Distribution License
CMB	Chandy-Misra-Bryant
COLA	Cloud Orchestration at the Level of Application
CRC	Centralised RTI Components
CSim	Cloud-Based Simulation
CSO	Case Study Organization
CSP	COTS Simulation Package
CSSP	CloudSME Simulation Platform
DFK	Development Kit Framework
DICE	**Di**stributed Simulation **C**loud Architecture for **E**xperimentation
DoH	Department of Health
DON	Distributed Observer Network
DS	Distributed Simulation
DSaaS	Distributed Simulation as a Service
DSC	Distributed Simulation for Cloud Computing
DSL	Distributed Simulation Layer
DVE	Distributed Virtual Environment
EaaS	Execution as a Service
ENIAC	Electronic Numerical Integrator and Computer
FEDEP	Federation Development and Execution Process
FP7	Seventh Framework Programme
GSP	General Simulation Program
HPCaaS	High-Performance Computing as a Service

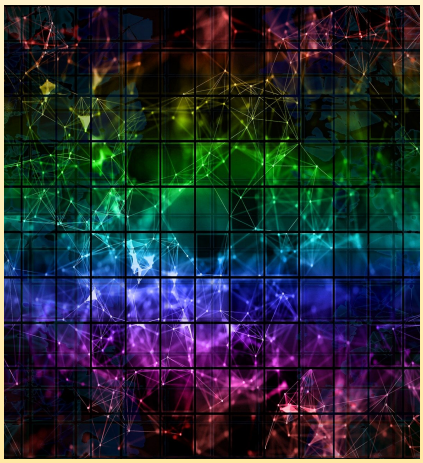
IRMs	Interoperability Reference Models
JaamSim	Java Animation Modelling and Simulation
LAS	London Ambulance Service
LCIM	Levels of Conceptual Interoperability Model
LPs	Logical Processes
LRC	Local RTI Components
MaaS	Modelling as a Service
MiCADO	Microservice-based Cloud Application-level Dynamic Orchestration
MS-IaaS	Modelling & Simulation Infrastructure as a Service
MS-PaaS	Modelling & Simulation Platform as a Service
MS-SaaS	Modelling & Simulation Software as a Service
MSaaS	Modelling & Simulation as a Service
NER	Next Event Request
OSAMS	Open System Architecture for Modelling and Simulation
pRTI	Pitch Runtime Infrastructure
PSE	Parameter Space Exploration
PTDF	Petroleum Technology Development Fund
RePAST	REcursive Porous Agent Simulation Toolkit
SDEM	Simulation Data Exchange Model
SEE	Simulation Exploration Experience
SIMaaS	Simulation as a Service
SIMNET	Internet of Simulations
SimSaaS	Simulation Software as a Service
SIMULA	Simulation Language
SKF	Starter Kit Framework
SO	Simulation Optimisation
SOF	Simulation exploration and Optimisation Framework
SRaaS	Simulation Resource as a Service
TAR	Time Advanced Request
TOSCA	Topology and Orchestration Specification for Cloud Applications
VV&A	Verification, Validation and Accreditation
WSC	Winter Simulation Conference

List of Tables

List of Figures

List of Equations

Table of Contents



CHAPTER ONE

INTRODUCTION

Chapter 1 Introduction

1.1 Chapter Overview

This section introduce the reader with a high-level overview of the work submitted. It begins with an introduction to the research background, context, motivation, and the questions this thesis is out to address. The aims and objectives are presented as a vehicle to design, execute and complete the research. Furthermore, this chapter also gives a brief overview of the succeeding sections.

1.2 Research Context

From the beginning of the electronics era, many inventions were recorded. These include the first general-purpose electronic computer's appearance – ENIAC (Burks and Burks, 1981) in the late 1940s. Subsequently, this period saw the development of the first general-purpose simulator - the General Simulation Program (GSP) by Keith Douglas Tocher (1963) and the first English-like simulation language – SIMSCRIPT by Harry Rice and his team (Rice *et al.*, 2005) in the late '50s and early '60s respectively. The GSP is designed to allow the systematic building of a simulation of an industrial plant that comprises machines with busy, failed, and idle states. In GSP, machine states and the next action times define the plant's state (Tocher, 1963). The world is witnessing many simulation languages, tools, environments and approaches, and techniques from that development. Today, organisations use Modelling and Simulation (M&S) is one realistic way to analyse existing or proposed systems or processes. The method uses computer models to predict how a real-life system will behave, given a set of conditions, parameters, values, and domain-specific data.

Luo et al. (2015) believes that in M&S domain, simulation is increasingly the only method capable of analysing, designing, evaluating, or controlling the large-scale, complex, uncertain systems in which we are interested. Analyst uses M&S to investigate complex dynamic systems. The practice usually involves creating a system model and experiment with different scenarios under varied conditions (Law, 2015). A typical simulation, analysts build a model and run experiments sequentially on a single computer system. This approach exposes modellers to practical limitations such as processing power and time.

The evolution in M&S brought about the Distributed Simulation (DS), which uses parallel and distributed computing techniques and multiple computers to speed up a simulation

program's execution or link together simulations to support reusability (Fujimoto, 2001). Other benefits of DS include speedup experiment, model reuse, data privacy, data consistency, and interoperability (Wu *et al.*, 2007; Anagnostou and Taylor, 2017a; Taylor, 2018). These benefits facilitate distributed experimentation. However, DS development is a complex and multidisciplinary task (Taylor *et al.*, 2014).

Analysts uses modelling and simulation and the community of practice is vying for high-speed experimentation methodology. Researchers and modellers are likely to benefit by composing models from interoperating new or reused ones that reduce model development time. In large models, those interoperating subsystem models can be reused in other simulations running on-premises simulation platforms locally or remotely accessible via the Internet. With the advent of Industry 4.0, inter-organisation simulation models can be loosely coupled through DS, which allows sharing information without compromising information and data security.

Among the significant challenges attributed to DS, is the high-performance computing resources and infrastructure and the amount of time required to execute experimentations (Fujimoto, 2016). These come with the high cost of hardware, configuration, and maintenance. Furthermore, Anagnostou and Taylor (2017) reported that developing DS can be highly complex due to the experience, technologies, and multiple disciplines involved, coupled with a lack of established architecture and guidelines to use. These make it cumbersome and probably, seen as the least alternative by some analysts.

Having some of the challenges identified the cloud concept provides a promising alternative. Cloud Computing enables on-demand network access to shared configurable resources (Mell and Grance, 2011). The cloud services come in three primary models (Mell and Grance, 2011; Qaisar, 2012); *Software as a Service (SaaS)* - The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. *Platform as a Service (PaaS)* - The capability provided to the consumer is to deploy onto the cloud infrastructure. *Infrastructure as a Service (IaaS)* - The consumer's capability is to provision processing, storage, networks, and other fundamental computing resources. All these concepts are detailed in chapter two.

Fujimoto, Malik and Park (2010) reported that cloud computing services are offered to users through the Internet, which reduces the burden associated with managing computing resources and facilities.

Having some of the challenges identified the cloud concept provides a promising alternative. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance, 2011). Cloud computing models are composed based on five essential characteristics, three service models, and four deployment models. The essential characteristics are;

On-demand self-service - A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider. *Broad network access* - Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations). *Resource pooling* - The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacentre). Examples of resources include storage, processing, memory, and network bandwidth. *Rapid elasticity* - Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time. *Measured service* - Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

The cloud services come in three primary models (Mell and Grance, 2011; Qaisar, 2012); Infrastructure as a Service (IaaS) - The consumer's capability is to provision processing, storage, networks, and other fundamental computing resources. Platform as a Service (PaaS) - The capability provided to the consumer is to deploy onto the cloud infrastructure. Software as a Service (SaaS) - The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. This work will immensely benefit from the cloud characteristics, service, and deployment models to build DICE architecture for CBDS. The characteristics are where the thesis will answer the research

questions with the aim to address the challenges identified during the literature review. The Cloud concept has this potential, and Fujimoto, Malik and Park, (2010) reported that cloud computing services are offered to users through the Internet, which reduces the burden associated with managing computing resources and facilities.

Despite the challenges reported above which forestalls the adoption of DS, the literature has not given an authoritative definition of Cloud-Based Distributed Simulation (CBDS) infrastructure; at the time of writing this thesis. However, some publications gave an extended meaning based on concepts they present such as Simulation as a Service – SIMaaS (Tsai *et al.*, 2011; Azevedo, Rossetti and Barbosa, 2015; Shekhar *et al.*, 2016), Modelling and Simulation as a Service – MSaaS (Fujimoto, Malik and Park, 2010; Buora, Giusti and Barbina, 2014; D'Angelo, 2014; NATO, 2015; Wang and Wainer, 2016; Prochazka and Hodicky, 2017) and Distributed Simulation as a Service – DSaaS (Rajaei, Alotaibi and Jamalian, 2017).

Some examples of these concepts are **SIMaaS** - a simulation platform where many independent simulation instances can be executed in parallel. The number of such simulations can vary elastically to satisfy specified confidence intervals for the results (Shekhar *et al.*, 2016). **MSaaS** - delivers value to customers to enable or support modelling and simulation (M&S) user applications and capabilities and provide associated data on demand without the ownership of specific costs and risks (NATO, 2015). **DSaaS** - is a cloud service for simulation, especially targeting extensive simulations requiring parallel executions of simulation modules (Rajaei, Alotaibi and Jamalian, 2017).

For this research, therefore, CBDS deployment architecture for modelling and simulation is defined and used as;

> ***A technique that enables the execution of multiple distributed simulations run across multiple, on-demand, and configurable cloud infrastructure, platforms, and software for the user to use as a service, over WAN or the Internet.***

However, apart from a few concepts reported later in this thesis, there are very few cloud-based distributed simulation infrastructures in research. This identified gap suggests the following research questions.

RQ1 - How can you deploy distributed simulation on the cloud?
RQ2 - What are the factors affecting the interoperability of distributed simulation on the cloud?

RQ3 - What are the factors affecting cloud-based distributed simulation experimentation speed?

1.3 Research Aim and Objectives

This research aims to investigate cloud-based federate development framework and multi-cloud deployment architecture for Distributed Simulation (DS).

The objectives below will be met to achieve the aim and address the research questions posed earlier.

Objective 1: To review the literature and uncover the theoretical perspective on the issue of cloud and distributed simulation. Also, look at the challenges in developing cloud-based distributed simulation infrastructure that can be used in operational research.

Objective 2: Identify a suitable methodology to apply to address the research questions, which will help achieve the thesis' aim.

Objective 3: Design and develop theoretical framework for could-compatible federate development and CBDS deployment architecture for experimentation by operational researchers.

Objective 4: Implement and test the feasibility of the proposed development framework and cloud deployment architecture using an appropriate case study.

Objective 5: To evaluate the proposed framework and deployment architecture using the experimentation results analysis.

1.4 Research Contribution

Distributed Simulation (DS) is used to analyse operational systems such as manufacturing and engineering. DS requires enormous computing resources (huge amount of hardware such as CPU, memory, and storage) and high technical skills to run experiments. This discourages analysts from adopting it. Cloud computing presents an alternative by offering on-demand network access to commodity hardware resources using the pay-as-you-go model. Therefore, *the main contribution of this research to the field of M&S, DS, and Cloud-Based distributed simulation is the **Di**stributed **C**loud Architecture for **E**xperimentation (**DICE**). It is designed and proposed to ease the conceptualising, design, building, deployment, and execution of Cloud-Based Distributed Simulation (CBDS)*. DICE will enable non-technical and other domain modellers to conduct distributed simulation experiments using cloud resources. A prototype distributed and complex hybrid emergency medical service model was used to

test its feasibility. The precise steps in the framework make it easy to follow and iterate sub-activities until the development is complete, and the experiment is successful. To the best of the author's knowledge, this is the only architecture and methodology for developing and deploying CBDS.

1.5 Significance of the Study

In theory, research creates new or extends existing knowledge, methods, or approaches to doing things in our lives. In the same vein, this work presents a new method of conducting DS experimentation using cloud infrastructure for non-technical modellers who may not have software engineering background. This brings the dual benefits of cloud and DS closer to the research community, which will enable them to join the early adopters of cloud-based distributed simulation for system analysis.

Some CBDS benefits were presented above and Fujimoto, Malik and Park (2010) argues that cloud computing lowers the barrier to begin exploiting these technologies. The authors further claim that it eliminates the need to purchase, and more importantly, operate and maintain high-performance computing equipment at the local site. As reported in chapter two, the existing literature recorded attempts to put the simulation experiment in the cloud, and its potential benefits. However, the research community comprising a significant number of non-technical analysts are exposed to the sophisticated technical knowledge required to design, develop, and deploy simulation models to the cloud, especially in the distributed environment.

1.6 Outline of the Thesis

This document is organised in chapters from one to seven. Table 1-1 gives an overview and which objective, each chapter addresses.

Table 1-1: Thesis outline mapped to objectives

	Highlights	**Objective Served**
Chapter One	Introduction, research context, aim, objectives and research questions.	

Chapter Two	Background study in the literature used to identify the knowledge gap.	Objective One
Chapter Three	Identifies the methodology to be used to achieve the research aim.	Objective Two
Chapter Four	Conceptual design of the DICE architecture.	Objective Three
Chapter Five	Implement the architecture using a case study prototype.	Objective Four
Chapter Six	Evaluating the architecture.	Objective Five
Chapter Seven	Conclusions, revisiting the research questions, limitations, and future research direction.	

Individual chapter overviews are now presented to prepare the reader.

Chapter 1: Introduction

This section presents the reader with an overview of the research. It begins with an introduction to the research background, context, motivation, and the questions this thesis is aiming to address. The aim and objectives are presented as a basis to design, execute and complete the research.

Chapter 2 (Objective 1): Literature Review

The second chapter reports the theoretical perspectives found in the relevant literature. The review leads to identifying the *gap* in cloud-based distributed modelling and simulation in the context of operational research. Various methodologies were explored; simulation concepts and techniques, available tools, standards, and technologies are also explained.

Chapter 3 (Objective 2): Methodology

Part three explains the rationale behind the research methodological approach for designing, experimenting, and analysing data. The justified reasons for choosing the hybrid EMS model for this project are the challenges in analysing complex systems using simulations.

Chapter 4 (Objective 3): Architecture Development

The thesis contribution starts here. This section explains how the research design and proposed DICE build on Distributed Simulation Engineering and Execution Process (DSEEP), a well-established method. The proposed architecture for cloud-based DS aims to ease simulation development and execution by connecting geographically distributed models. This takes advantage of the high-performance computing resources offered by cloud infrastructure. Overall, the chapter expatiates on the architecture technical components, strengths, and limitations.

Chapter 5 (Objective 4): Implementation and Testing

During implementation, the London Emergency Medical Services (EMS) model is used to implement and test the proposed DICE architecture. It is initially developed by Anastasia Anagnostou (2014) as a hybrid distributed simulation model combining two simulation paradigms – Agent-Based Simulation (ABS) and Discrete Events Simulations (DES). It is used as a case study in this thesis by reconfiguring and upgrading cloud-based deployment and experimentation. The chapter explains the tools used, and the justifications are presented here. Recursive Porous Agent Simulation Toolkit (RePAST) Symphony, the opensource, cross-platform modelling, and simulation toolkit is presented. The model collaboration layer, poRTIco run-time infrastructure (RTI), is also an open-source component that forms part of this chapter. The chapter also narrates the experiment design, execution, and the results accumulated for analysis in the succeeding parts.

Chapter 6 (Objective 5): Results and Evaluation

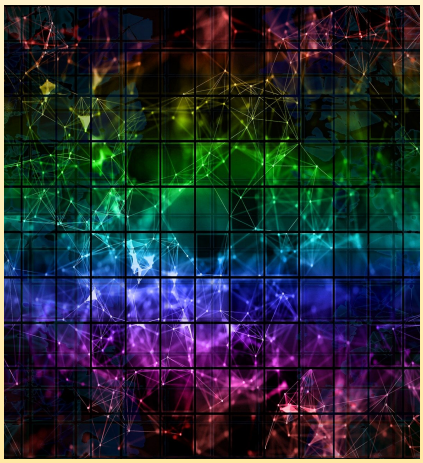
Here, architecture testing results are shown and analysed by comparing the inputs and measuring the performance concerning earlier implementations reported in the literature for a single machine, networked environment, and related cloud implementations. The chapter presented an evaluation report, and the literature underpinning the validity claimed.

Chapter 7: Conclusion and Future Work

The concluding section summarises the work, how the research questions were addressed, the scope, alternative approaches, and research hypothesis. It gives a cue on the future direction where further work is needed and the potential research opportunities.

1.7 Chapter Recap

This section introduces the research field, context, and the questions to be addressed. Limitations and significance of the study were presented, and it ends with all section's overviews. The next part will go further into the literature and the various concepts to aid this research. The background study will help identify and apply the right methodology for the investigation.



CHAPTER TWO

REVIEW OF THE LITERATURE

Chapter 2 Review of the Literature

2.1 Chapter Overview

The previous chapter introduced the research and its context for this thesis. It presented the problems that motivated this work, and the hypothesis that development framework and deployment architecture will bring CBDS closer to analysts was established. It also presented the project aim, research questions, and the objectives it's set to achieve. Further, it discussed the contribution and significance of the study, and finally, the thesis outline was tabulated, and each chapter overview presented.

This chapter reviews the recently published research in Distributed Simulation, Distributed Simulation, and Cloud-based Simulation and identifies the gap in the literature. Moreover, the section gives the reader history and general concepts of simulation, types of modelling, world views, approaches, and experimentation. It introduces some essential aspects of CBDS; the high-level architecture (HLA), time in simulation, and time management (synchronisation). The chapter also analyses, and reports simulation methodologies related to this thesis from both on-premises and cloud infrastructures. Then relates how that relates to the M&S research communities of practice.

Overall, sections in this chapter are dedicated to detail the theoretical perspective for various modelling and simulation approaches; discrete event, agent-based, hybrid, and system dynamics. Finally, the cloud computing concept, cloud-based simulation, and how it is used is discussed in detail and presents the potential benefit to the broader M&S researchers.

2.2 History of Simulation

Modern computers allow the analyst to explore the whole range of feasible options in a decision problem. Some of these options could be examined without a computer. However, the process and the problem may well change significantly before a satisfactory solution is produced (Pidd, 1984). This journey began in the early stage of the electronic era and understood the perspective of simulation. It is valuable to understand the history of simulation.

Goldsman *et al.* (2009) notes the two significant developments that set the stage for the rapid growth of the field of simulation in the mid-1940s. The construction of the first general-purpose electronic computers such as the ENIAC and the work of Stanislaw Ulam, John Von Neumann, to use the *Monte Carlo* method on electronic machines. These tried to solve

specific problems in neutron diffusion that arose in the design of the hydrogen bomb, and that were (and still are) analytically intractable.

In Computer Science (CS), computer-based modelling and simulation, has become the third research methodology, complementing experiment and theory (Dodig-Crnkovic, 2002). Many projects can use simulation methods. (Mohannad and Ayash, 2013) argued complex phenomena, such as the evolution of the universe that cannot be implemented in laboratories, using the simulation method. Researchers today, are witnessing sophisticated computing environments and methods that are powerful enough to enable them to tackle problems of enormous complexity. Some authors believe that CS can further be divided into Theoretical, Experimental and Simulation - three methodologically distinct areas. *Modelling*, however, is one method that is common for all three. Modelling is a process that often occurs in science, where it is an abstraction, and the phenomenon of interest is simplified, to be investigated or studied. Relevant features of a phenomenon are taken into account while building a model. Because in science, there are some theoretical grounds available in the literature and industry, it is crucial to know which features are relevant to the *system under investigation (SUI)*.

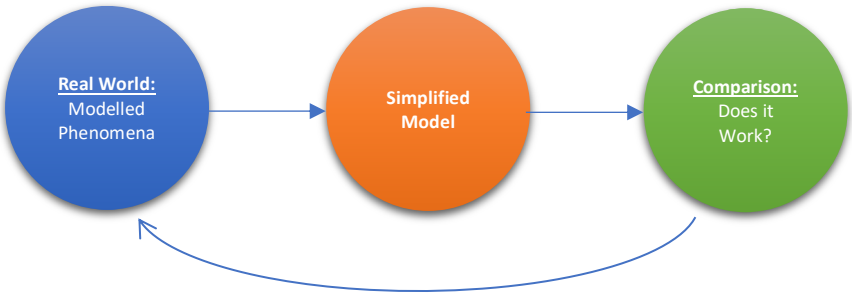


Figure 2-1 Modelling in its purest form

In CS, the model of a phenomenon has a description, which enables analysts to predict measurable consequences of a given change in system behaviour over time. Figure 2-1 illustrates how new or modified models are systematically compared (or benchmarked) with existing ones and analyse their relation, and relative strength or weakness. The earlier three; theory, experiment, and simulation work with models of phenomena.

A professor of Operations Research at the University of Southampton by the name Keith Douglas Tocher developed the GSP (General Simulation Program). It is the first general-purpose simulator (Goldsman *et al.*, 2009) to be used as a tool to systematically construct a simulation model for industrial plants. The program uses a set of machines with a transition between states (e.g., failed, unavailable, busy, or idle). Douglas' contributions to simulation technique include The Art of Simulation, a pioneering textbook on simulation and the Activity Cycle Diagram (ACD) in 1964. Figure 2-2 shows an ACD for healthcare operation. The ACD

became a cornerstone of simulation teaching in the UK and the core of research in program generators during the 1970s.

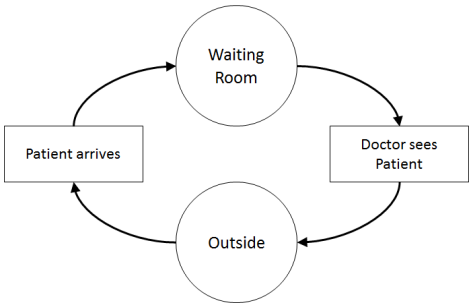


Figure 2-2 Activity Circle Diagram showing a health clinic model (Adapted from Pidd, 1984)

Simulation, from this view, is one of the most widely used, in literature as the preferable quantitative methods due to its flexibility and presents many statistical results helping decision-makers take the right direction towards improving as reported by (Balachandran, 2000);

> *"Simulation is extensively being used as a tool to increase production capacity. Simulation software used by Cymer Inc. (a leading producer of laser illumination sources), increased the production capacity from 5 units/month at the beginning of 1999 to 45/month at the end of 1999, an increase by around 400%."*

2.2.1 Modelling and Types of Models

Modelling is the process of producing a model (Anu, 1997). A model in a simulation project is a representation feature or behaviour of a system of interest. One purpose of a model is to enable the analyst to predict the effect of changes to the system. Earlier than Maria's contribution, (Pidd, 1984) stated that models are representations of the system of interest and are used to investigate improvements in the real system or to discover the effect of different policies on that system. From the M&S perspective, models are built, and experiments are conducted to analyse operational systems, uncover bottlenecks, improve processes, or test proposed new systems against established criteria. There are two fundamental types of modes; deterministic models and stochastic models (North and Macal, 2007).

Deterministic - models always produce the same outputs given the same inputs, since each of the agents involved always acts the same way given identical inputs. In these models, there are no random variables and usually contains equations. Designers use known inputs and outputs, which can be used to capture natural process. **Stochastic –** models can produce different outputs when they are repeatedly run identical inputs. Stochastic models can produce different output because they include agent behaviours

or environmental response based on random or probabilistic elements. Pidd & Michael also revealed that this type of model' behaviours could not be entirely predicted.

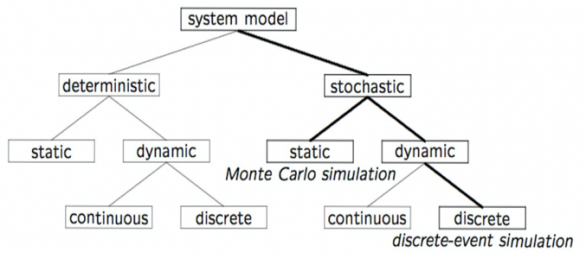


Figure 2-3 Model Taxonomy (Adapted from Law and Kelton, 1991)

Figure 2-3 is a model taxonomy proposed by (Law and Kelton, 1991). The authors show that models can be either deterministic or stochastic. They believe stochastic types are more complex and better represent actual systems than deterministic models.

2.2.2 Simulation World Views

The evolution of modelling and simulations tools focused more on balancing between flexibility and ease-of-use. A worldview is a modelling framework that a modeller uses to represent a system and its behaviour. The main terminology and concepts include systems and models, system state variables, entities and their attributes, lists, resources, events, activities and delays (Carson, 1993). On another note, (Pegden, 2010; Chan and Pegden, 2017) reveals that over the 50-year history of simulation, there have been three distinct worldviews in use. They are event, process, and objects[1]. The objective remains similar; a worldview provides a definitive set of rules for advancing time and changing the discrete or contiguous states of the model under investigation.

2.2.3 Time in Simulation

Simulation deals with time in two ways; simulation time and run time. Analysts require run times small enough to get a result within the resources available. However, the simulation time is more critical in terms of the result and how the simulation is organised (Garrido, 1999). On the other hand, event time is used in simulation projects to monitor various events. Therefore, this research work uses *simulation time* as the time on the simulation clock - the virtual time or logical time in the simulated world, *runtime,* the amount of processor time consumed, and *event time* as the simulation time at which an event occurs.

[1]. These terms will be explained later.

2.3 Discrete Event Simulation (DES)

Discrete Event Simulation (DES) is a technique that refers to the process of codifying the behaviour of a system as an ordered sequence of well-defined event series. A discrete event simulation model assumes the system being simulated only changes state at discrete points in simulated time. The simulation model jumps from one state to another upon the occurrence of an *event* (Fujimoto, 1990). DES is also seen as a tool that quantitatively represents the real world, simulates its dynamics on an event-by-event basis, and generates a detailed performance report (Babulak and Wang, 2010). This means DES is used to model a system with a changing state at a specific (discrete) point in time. During DES execution, every event occurs at a particular point in time and marks a state change in the system (Kiran, 2019). DES has some properties worth noting (Lucas *et al.*, 2015); DES can be s*tochastic (probabilistic)* where inter-arrival times and service times are random variables and have cumulative distribution functions. DES has *discrete intervals* of time, which separate instantaneous events. The state variables change instantaneously at separate points in time. The system can change at only a countable number of points in time, and these points are the ones at which an event occurs. Lastly, DES has a *dynamic* property that changes over time. It uses a simulation clock to track the current value of simulated time as the simulation proceeds—a mechanism to advance simulated time from one value to another.

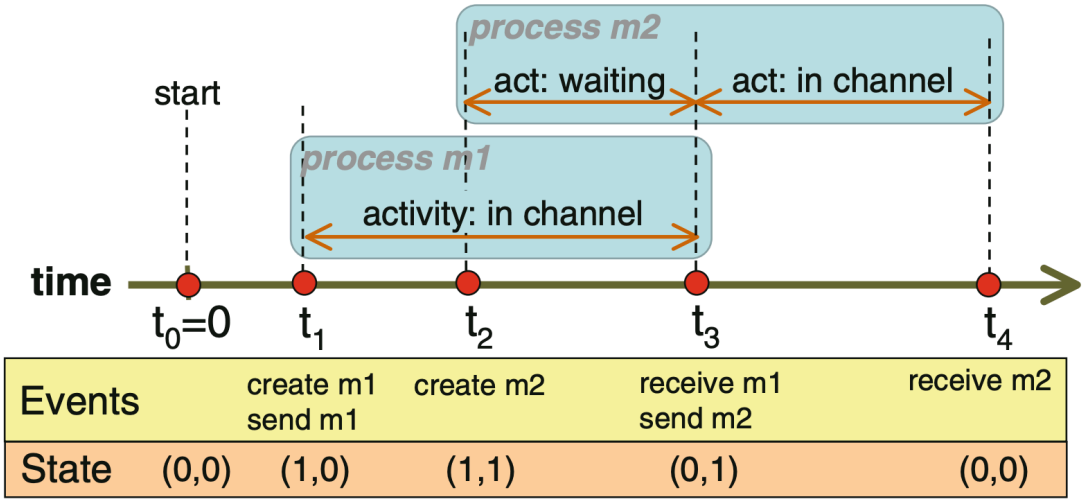


Figure 2-4 Concept of Discrete Event Simulation (Adapted from Lara, Guerra et. al., 2012)

DES possess certain features which are used to design a discrete model. These features are used to capture information about various components of the system under study. Briefly, the features include *Activities* - Where things happen to entities during some time (which may be governed by a probability distribution). *Queues* - entities wait an undetermined time. *Entities* - Wait in queues or get acted on in activities. *Attributes* - defines entities like

kind, weight, due date, and priority. *Simulation clock* - is a variable giving the current value of simulated time. *Event list* - a list containing the next time when each type of event will occur. *Statistical counters* - are variables used for storing statistical information about system information. Figure 2-4 illustrates the arrival of two messages: *m1* at time t_1, *m2* at t_2 and their dispatch via a channel with a single capacity. This means message *m2* must wait while *m1* is using the channel. This concept shows that a message may perform two activities: *waiting* in a queue or *moving* through the channel. The figure also shows events and states with two parameters (<number of messages in the channel>, <number of messages waiting in the queue>. To process these messages, the system performs the two activities in sequence (de Lara *et al.*, 2014).

As the DES started to gain ground, three concepts were defined to help programmers, and developers implement DES in an event, activity, and process-oriented approaches (Fishman, 1973; Nance, 1993). Event Scheduling approach offer primitives to describe events, future events, and their effect on the current state. Here, simulation time advancing to the next event occurrence is used to manage time efficiently. Activity Scanning focuses on describing the starting condition of activities. They are less efficient because they advance the time using a small discrete increment and check at each time whether new activities can be started. Process Interaction, which provides constructs to describe the life cycle (the processes) of each active entities of the system under study.

The features of DES listed above makes it applicable in many domains such as factories where *entities* in this context can be products, people, transporters, tools. *Activities* may be fabrication and assembly. *Queues* can be implemented at conveyors or warehouses. If highways are considered another example, then their *entities* are emergency booths, cars, trucks, and cops. *Activities* are, go, stop, rage, and switch lanes. *Queues* can be formed on highways ramps, rest stops, and traffic at maintenance spots. Other aspects of DES are the Parallel and Distributed Simulation of DES, which is explained in detail late in this chapter.

2.4 Agent-Based Modelling and Simulation (ABMS)

Agent-based modelling offers a way to model social systems that are composed of agents who interact with and influence each other, learn from their experiences. Agents adapt their behaviours, so they are better suited to their environment (Macal and North, 2010). An *agent* is simply regarded as an entity, notion, or software abstraction similar to the well-known programming specifications such as objects, methods, procedures and functions. An element or object abstraction wraps the methods and attributes of a software module (Abar *et al.*, 2017).

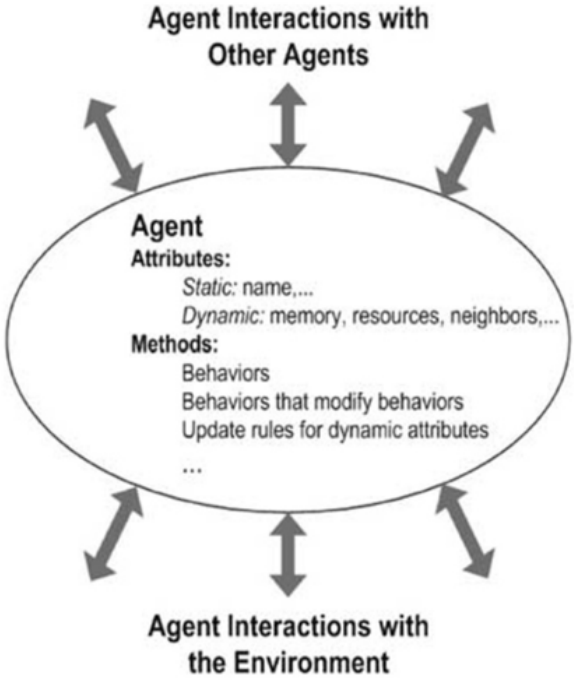


Figure 2-5 Structure of Agent showing attributes, methods, and interactions (Adapted from Macal and North (2010)

In ABMS, entities are referred to as agents and their behaviour defined. Example of agents, depending on the context, include household, equipment people, vehicles, products, corporation or whatever is related to the system under investigation (SUI). In Figure 2-5, Agent name is an example of a static attribute while memory and resources are dynamic. Behaviour is categorised as a method, for instance, move, fly and so forth. The modeller establishes connections between the agents, sets the necessary environment variables and simulations run. This allows the SUI dynamics to emerge from the interactions of the many behaviours. It is believed that anything that can choose in a business or system can be viewed as an agent. Agents are identified as an individual with a set of attributes which defines what they are and behavioural characteristic like what they do.

2.4.1 Structure of Agent-Based Modelling and Simulation (ABMS)

A model developer must identify, model, and program these elements to create an agent-based model. A computational engine for simulating agent behaviours and agent interactions is then needed to make the model run (Macal and North, 2011). An agent, as shown in Figure 2-6, relationship, and environment are the three essential elements found in an agent-based model: first, a set of *agents*, their attributes, and behaviours. Secondly, agent *relationships* and interaction methods are the underlying topologies of connectedness, which defines how and with whom agents interact. The *environment* is where agents interact with their environment in addition to other agents.

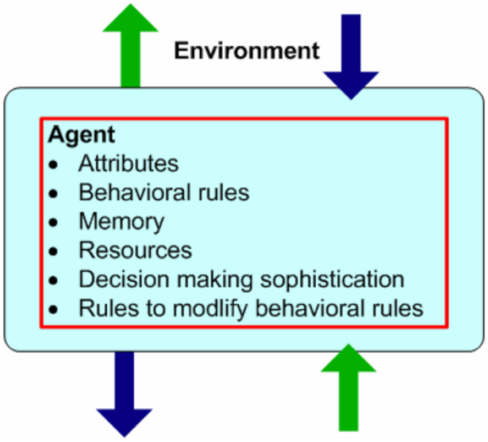


Figure 2-6 An Agent with its properties (Adapted from Macal and North, 2011)

Everything associated with agents in the ABMS model is either methods or attributes. Attributes can be static (does not change at runtime) or dynamic, which changes as simulation progress.

2.4.2 Agent-Based Modelling and Simulation (ABMS) Methods

There are many approaches to implementing ABMS. Usually, when designing and developing an agent-based model, it is important to pose a series of questions. The answers will form an essential part of the ABMS design process (Macal and North, 2010). These questions include *What specific problem should be solved by the model?* What specific questions should the model answer? What value-added would agent-based modelling bring to the problem that other modelling approaches cannot bring? *What should the agents be in the model?* Who are the decision-makers in the system? What are the entities that have behaviours? What data on agents are merely descriptive (static attributes)? What agent attributes would be calculated endogenously by the model and updated in the agents (dynamic attributes)?

Agent-based modelling can be implemented using general, all-purpose software or programming languages. It can also be done using specially designed software and toolkits that address agent modelling particular requirements. Agent modelling can be done in the small, on the desktop, or large, using large-scale computing cluster, or it can be done at any scale in-between these extremes. Projects often begin small, using one of the desktops ABMS tools, and then grow in stages into the larger-scale ABMS toolkits. Often, one begins developing the first agent model using the approach that one is most familiar with. It could be the approach that one finds easiest to learn given their background and experience. Regardless of the specific design methodology selected, a range of services may be required for implementing small or large-scale models that

include real data and geospatial environments. These are becoming more prevalent. Some of the more common capabilities include project specification services; agent specification services; input data specification and storage services; model execution services; results storage and analysis services; and model packaging and distribution services.

Finally, ABMS's ability to capture emergent phenomena has been claimed as a distinct feature that makes it suitable for various purposes and applications. Publications revealed that agent-based models are developed for a wide range of different purposes. For simplicity (Rixon, Moglia and Burn, 2005) classified them as ABMS for Constructive Learning such as companion modelling, resource management, and environmental modelling. ABMS can be implemented as virtual laboratories for theory development, building exercises or economic investigations. ABMS is used in technological and engineering applications such as those relating to optimisation and distributed problem-solving.

2.5 Hybrid Simulation

Growing globalisation, rising outsourcing, deepening of information technology, sophisticated products, expanding horizontal integration and increasing customer demands are some of the reasons contributing to complexity in the world (North and Macal, 2007). This makes the systems that need to be analysed increasingly complex too. To analyse large and complex systems, the idea of combining more than one simulation technique – the Hybrid Simulation (HS), has a strong practical appeal for the research community. Scholars contributed several different modelling methods, combined in various ways. Most real-world problems and systems are complex. With many different features and characteristics, and very rarely is one single method ideally suited to capture all of them (Brailsford *et al.*, 2019).

From the M&S perspective, HS can mean several things (Shanthikumar and Sargent, 1983). For example, models that are simultaneously implemented on both analogue and digital computers, contain both discrete and continuous variables, or models that combine simulation with an analytical method such as optimisation. In this project, HS refers to *models that combine more than one simulation paradigm*.

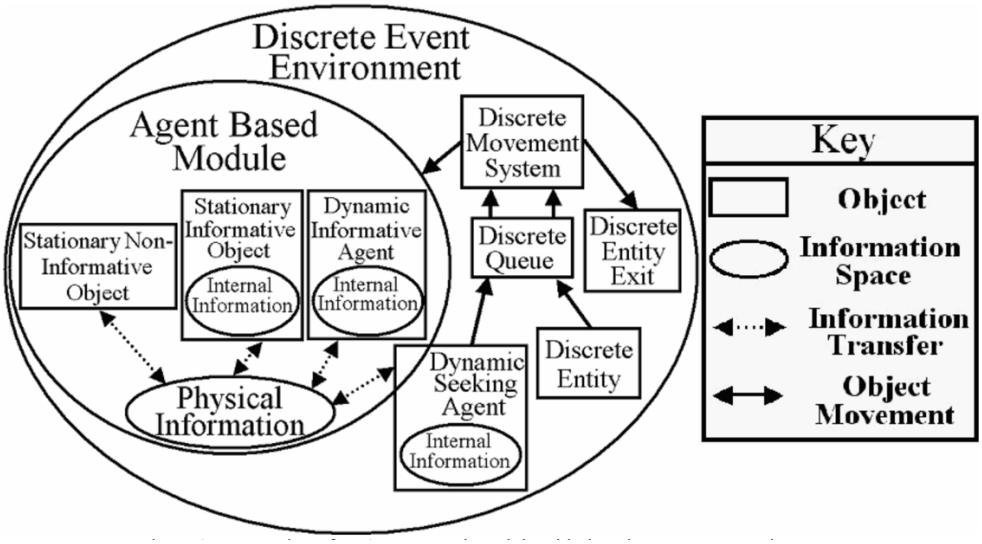


Figure 2-7 Diagram of a hybrid simulation model of a Theme Park integrating ABS into DES (Adapted from Dubiel and Tsimhoni, 2005)

Research communities have been experimenting and proposing solutions by using HS such as ABS-DES (Wang, Zheng and Zhao, 2013), DES-SD (Viana *et al.*, 2014), ABS-SD (Djanatliev *et al.*, 2014), and ABS-DES-SD (Block, 2018). Let us look at a scenario related to this work. AutoMod (Rohrer, 2002; Muller, 2011) simulation package was used by (Dubiel and Tsimhoni, 2005) and developed a Theme Park model where a new visitor wanders around, walks to the information centre, and asks for a location map. The visitor has the option to either use a tram - a discrete movement or walks around the parks, which is in the form of ABS. The model illustrated in Figure 2-7, predicts arrival patterns to the discrete parts of the system - tram or ques. The model also aims to identify location, signs, quantity of maps and inform park employees to minimise travel time of visitors and maximise flow around parks.

The decades of the recorded exponential growth of HS makes it an attractive system analysis technique of choice within M&S in the areas of healthcare, manufacturing, and supply chain management. This thesis uses the healthcare system to investigate HS feasibility and performance in a Cloud-Based Distributed Simulation (CBDS).

2.6 Parallel & Distributed Simulation (PADS)

Sharing common challenges and overlapping issues, parallel and distributed simulation are two different simulation methods, each with its research community that emerged in the ‘70s and ‘80s (Fujimoto, 2015a). Parallel Discrete Event Simulation (PDES) practitioners are concerned with accelerating discrete-event simulations by exploiting high-performance computing platforms. Distributed Simulation (DS) came from defence efforts and is more interested in linking up individually developed simulation executing on computing resources interconnected local or wide area network environments. This thesis's core contribution is

directly connected to the latter – the DS and explained more in-depth. Meanwhile, below is an introduction to both PDES and DS.

2.6.1 Parallel Discrete Event Simulation (PDES)

The parallel Discrete Event Simulation is a concept where a single simulation run is distributed over many processors associated with high-performance computing platforms. During simulation execution, the PDES program, Figure 2-8 uses several sequential discrete event simulations that interact by exchanging time-stamped messages, which is referred to as a logical process (LP). The messages exchange happens in the form of events. PDES contributes to significant success in simulating large systems in defence, computer systems design, and smart urban environments.

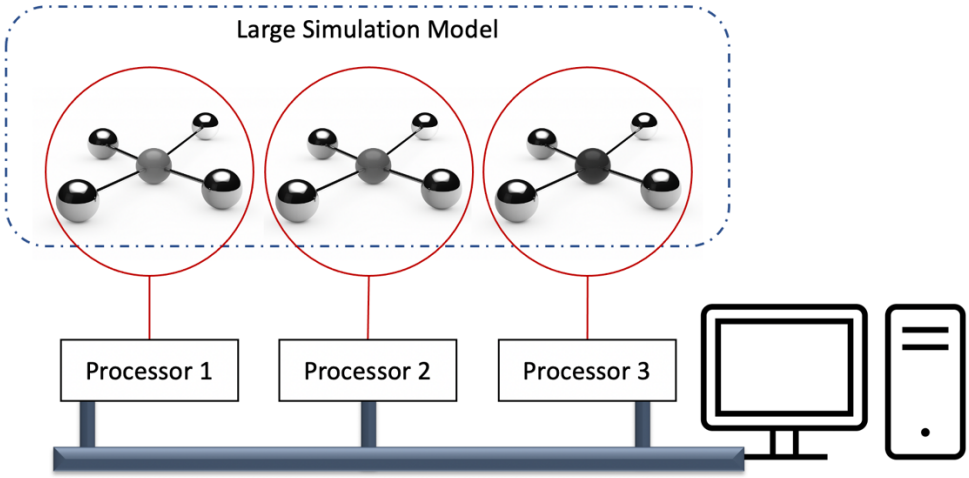


Figure 2-8 Parallel Discrete Event Simulation (PDES)

The parallel concept in PDES uses computing platforms equipped with shared-memory multiprocessors, which fits the main goal - to speed up the simulation execution. PDES is typically composed of sequential discrete event simulations interacting with messages. Each message is representing a scheduled event between simulators or LPs as they are called in the literature. By collecting together sequential DES for parallel execution, PDES should produce the same result or very close in some cases to sequential DES (Fujimoto, 2016). However, PDES is expected to produce results faster.

2.6.2 Distributed Simulation (DS)

Unlike PDES where the processors needed to run the simulation are usually in very close proximity, distributed simulations in Figure 2-9 take the idea of running experiments on computing resources geographically distant from one another. DS's are interconnected via local, wide area networks or the Internet. DS focuses mainly on the

use of many resources and parallel and distributed computing approaches to speed up the execution of simulation applications and links individually developed simulations that facilitate model reusability (Fujimoto, 2000). Among the many reasons encouraging DS are reduction in execution time by dividing a large and complex model into many submodules that can execute simultaneously over multiple processors. Another reason is the geographic distribution where computing resources can be lactated on different physical sites and linked up via communication networks. DS also allows computing resources from different vendors to be able to interact using the standardised protocol such as High-Level Architecture HLA - discussed below.

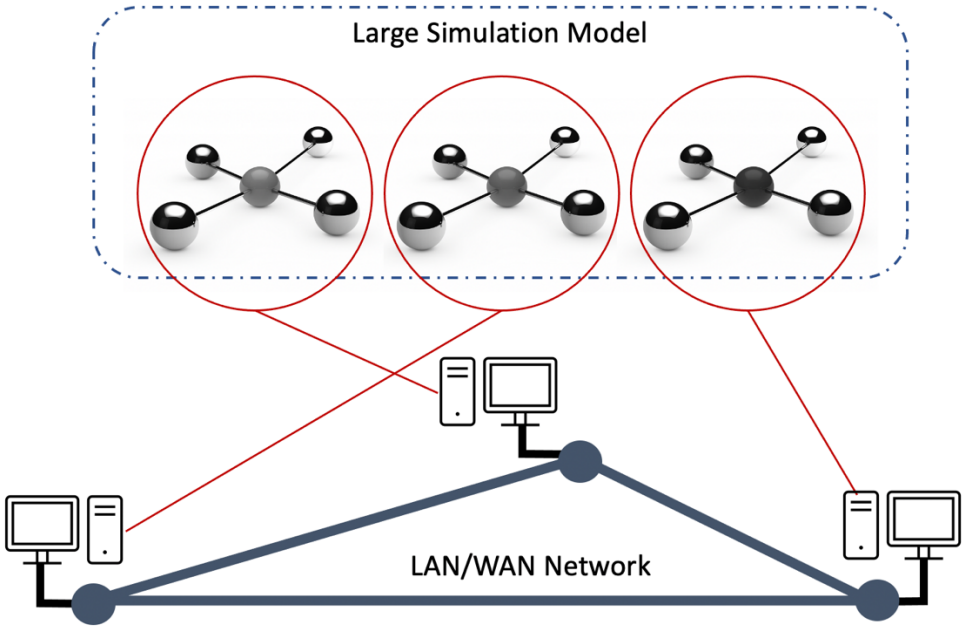


Figure 2-9 Distributed Simulation Running on Networked PCs

As the DS field grows and researchers work increasingly lean towards interoperability between simulations and among simulation applications, standards organisations put together resources and develop and publish standards to enable developers to interconnect simulations using established guidelines. Some of the DS standards include the High-Level Architecture (HLA) standardised as IEEE Std. 1516-2010, Distributed Interactive Simulation (DIS) standardised as IEEE Std. 1278-1995, and Distributed Simulation Engineering and Execution Process (DSEEP) standardised as IEEE Std. 1730-2010. This thesis leverages the HLA and DSEEP standards which are explained here and other sections as well.

The High-Level Architecture (HLA)

One form of DS in the market today is High-Level Architecture (HLA). The HLA IEEE 1516-2010 (Pedrielli *et al.*, 2012) is a well-known and accepted standard that

provides a distributed infrastructure in which each simulation unit runs on an independent computer (in general, geographically distributed) and communicates with the others in a typical simulation scenario (IEEE Std. 1516-2010). HLA was developed by the DoD Modelling and Simulation Coordination Office (M&S CO) in the period 1995-1996 as a general architecture to facilitate the integration of distributed simulation models within a common simulation environment (Falcone, Garro, Taylor, *et al.*, 2017). To facilitate interoperability and reusability, HLA differentiates between the simulation functionality provided by the members of the distributed simulation and a set of basic services for data exchange, communication, and synchronisation (Straßburger, 2006). Figure 2-10 shows a functional overview of a federation using HLA standard implementation.

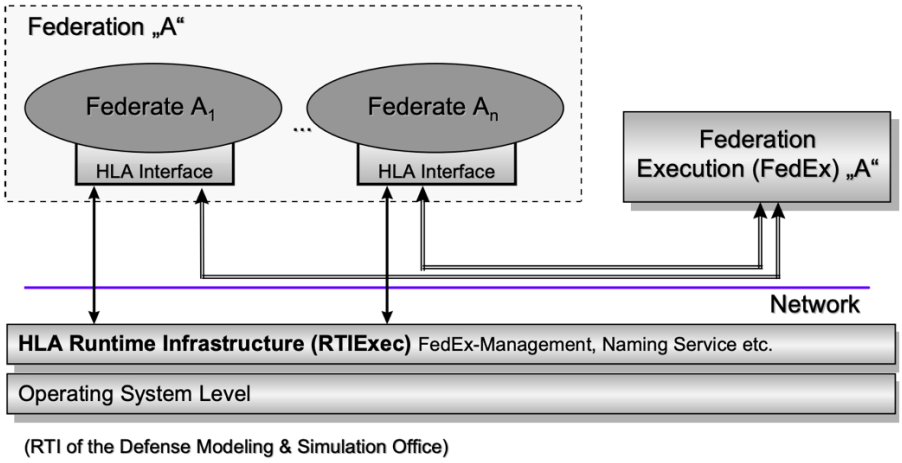


Figure 2-10 HLA: Functional View of a DS (Adapted from Straßburger, 2006)

Federation is a term used in the HLA to denote the composition of individual simulation models called federates. These federate communicates with each other via a Runtime Infrastructure (RTI)'s defined protocol. The RTI manages the interaction and exchange of messages among the connected federates. However, it is argued in (Möller *et al.*, 2016) that building complex and extensive distributed simulations, especially those based on the HLA standard, is usually a challenging task and requires considerable development experience in distributed systems, simulation, middleware, and software programming.

The HLA Framework and Rules

Contained in the IEEE Standard 1516-2010 official document, the standard provided five rules for federation and five rules for federates (altogether ten rules).

These enforce specific structure and responsibilities to make sure DS simulation models can be reused across applications.

Framework Rules for Federation:

a. Federations shall have an HLA FOM, documented in accordance with the HLA OMT.
b. In a federation, all simulation-associated object instance representation shall be in the federates, not in the RTI.
c. During a federation execution, all FOM data exchange among joined federates shall occur via the RTI.
d. During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification.
e. During a federation execution, an instance attribute shall be owned by at most one joined federate at any given time.

Framework Rules for Federation:

a. Federates shall have an HLA SOM, documented in accordance with the HLA OMT.
b. Federates shall be able to update and/or reflect any instance attributes and send and/or receive interactions, as specified in their SOMs.
c. Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs.
d. Federates shall be able to vary the conditions (e.g., thresholds). They provide updates of instance attributes, as specified in their SOMs.
e. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

The HLA Interface Specification

In HLA DS, communication, and data exchange pass through the RTI. The interface specification defined seven services used by the federates to interact with the RTI - the federation communication layer. These service groups were summarised by (Huiskamp and van den Berg, 2016) and quoted as follows;

a. **Federation Management.** These services allow for the coordination of federation-wide activities throughout the life of a federation execution. Such services include federation execution creation and destruction, federate application joining and resigning, federation synchronisation points, and save and restore operations.

This, for example, can be used to create "snapshots" of the simulation to resume execution at a later stage.

b. **Declaration Management.** These services allow joined federates to specify the types of data they will supply to, or receive from, during the federation execution. This process is done via a set of publication and subscription services along with some related services.
c. **Object Management.** These services support the objects' life-cycle activities, and interactions used by the joined federates of a federation execution. These services provide for registering and discovering object instances, updating, and reflecting the instance attributes associated with these object instances, deleting, or removing object instances as well as sending and receiving interactions and other related services.
d. **Ownership Management.** These services are used to establish a federates-specific privilege to provide values for an object instance attribute. It also facilitates the dynamic transfer of this privilege (ownership) to other joined federates during a federation execution.
e. **Time Management.** These services allow joined federates to operate with a logical concept of time and maintain a distributed virtual clock jointly. These services support discrete event simulations and assurance of causal ordering among events.
f. **Data Distribution Management.** These services allow joined federates to specify further the distribution conditions (beyond those provided via Declaration Management services) for the specific data they send or ask to receive during a federation execution. The RTI uses this information to route data from producers to consumers in a more tailored manner, for example, to receive only updates from objects that are in the geographical vicinity in the simulated world.
g. **Support Services.** This group includes miscellaneous services utilised by joined federates for performing such actions as name-to-handle and handle-to-name transformations, the setting of advisory switches, region manipulations, and RTI start-up and shutdown.

The HLA Object Model Template (OMT)

At federation execution runtime, data exchange between federates is defined in the object model template. OMT describes the state and interactions during simulation events. The template contains three sub-models in HLA implementation - FOM, SOM and MOM (Falcone, Garro, Taylor, *et al.*, 2017).

a. **Federation Object Model (FOM) -** is created in line with the OMT and contains object classes, interaction classes and data types. Optionally, FOM can contain federation-wide information for efficient data distribution.
b. **Simulation Object Model (SOM) -** provides details of the object attributes and interactions of what the federates send or receive.
c. **Management Object Model (MOM) -** a group of predefined constructs in HLA that supports monitoring and managing the federation execution.

Runtime Infrastructure (RTI)

The RTI controls the simulation execution during an experiment. It provides the federation with management functions and services using an XML file formatted FOM. RTI decides the kind of data that can be exchanged by the participating models within the federation. In the proposed DICE, the simulation requirements call for the use of RTI in the project design. Determining which RTI package to use depends on the objective and expected outcome. Many RTI implementations are available for free or commercially developed by vendors, academic institutions, and government agencies. Table 2-1 from Huiskamp and Berg (2016) gives a few examples of known implementations.

Table 2-1 RTI known implementations with HLA supported versions (Adapted from Huiskamp and Berg, 2016)

Vendor	URL	Standard	Binding	License
Pitch	http://pitch.se	HLA 1.3	C++, Java	Commercial
		IEEE 1516-2000	C++, Java	
		IEEE 1516-2010	C++, Java	
MÅK	http://www.mak.com	HLA 1.3	C++, Java	Commercial
		IEEE 1516-2000	C++, Java	
		IEEE 1516-2010	C++, Java	
CERTI	http://savannah.nongnu.org/projects/certi	HLA 1.3 (partial)	C++, Java	Open source: GPL (sources) and LGPL (libraries)
		IEEE 1516-2000 (partial)	C++	
		IEEE 1516-2010 (partial)	C++	
poRTIco	http://porticoproject.org	HLA 1.3 (partial)	C++, Java	Open source: CDDL 1.0
		IEEE 1516-2000 (partial)	C++	

		IEEE 1516-2010 (partial)	C++, Java	
Open HLA	http://sourceforge.net/projects/ohla	HLA 1.3 (partial)	Java	Open source: Apache Licence 2.0
		IEEE 1516-2000 (partial)	Java	
		IEEE 1516-2010 (partial)	Java	

RTI also provides the seven services listed under the interface specification section above. These facilitate easy integration of existing models with different applications and platforms such as Windows, Linux and Mac OS. RTIs are known for having a standard interface for communication using different programming languages such as Java, C++, and Python. In Figure 2-11, the *FederateAmbassador* is used to deliver information to federates using *callbacks* and *RTIAmbassador* instances invoked by federates to access RTI services.

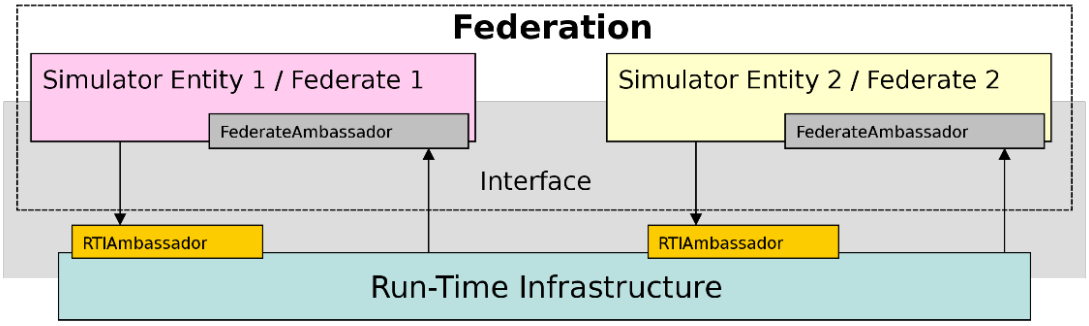


Figure 2-11 Federation with RTI implementation (Adapted from Wikipedia)

This project utilises the HLA, as shown in the EMS simulation Scenario in chapter 5.

Interoperability

Since 1996 the Simulation Interoperability Standards Organization (SISO) has been at the fore of reporting developments (Wilcox, Burger and Hoare, 2000). SISO gives directions for groups in defence and non-defence who are interested in developments by promoting modelling and simulation interoperability and reuse for the benefit of a broad range of communities including developers, procurers, and users worldwide (Serna *et al.*, 2010).

With the growing emphasis in the area of DS to enhance inter-operability amongst simulation applications separately developed, (Fujimoto, 2015a) revealed

that a substantial amount of effort has focused on developing standards to interconnect simulations. The Distributed Interactive Simulation (DIS) - IEEE Std 1278.1-1 995 1995; IEEE Std 1278.2-1995 1995 and the High-Level Architecture (HLA) - IEEE Std 1516-2010 20 10; IEEE Std 1516.1-2010 2010; IEEE Std 1516.2-2010 2010 (Open-DIS) and (Mccall and Murray, 2010) standards.

Moreover, the research communities work on solving not only the problems caused by interoperability issues but the underlying causes. A few examples of these developments in the DS are simulation methodology by (Banks *et al.*, 2013) and (Tolk and Muguira, 2003; Wang, Tolk and Wang, 2009)'s levels of conceptual interoperability model (LCIM). Figure 2-12, promotes composability via the application of engineering methods and principles, easing the transition out of the ad-hoc approaches. A short description is also given below the figure.

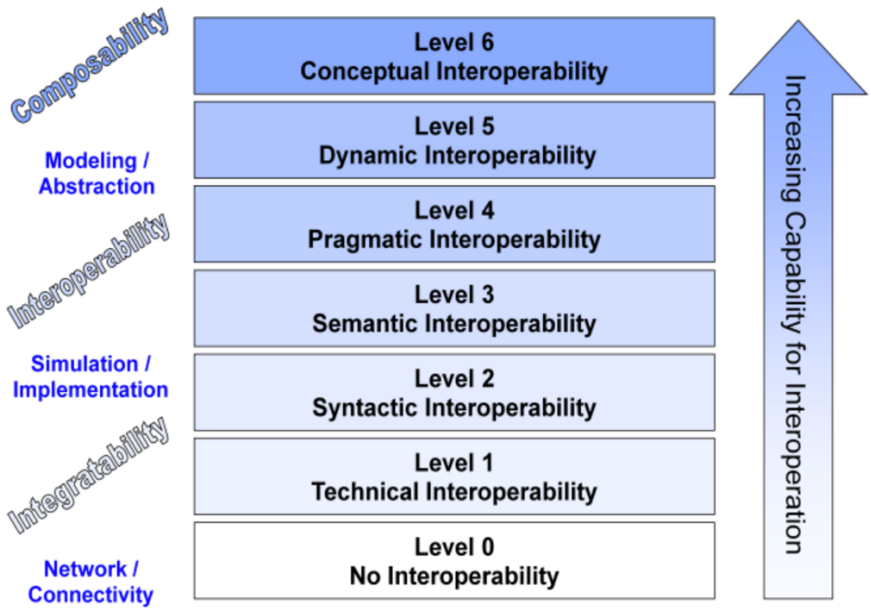


Figure 2-12 The Levels of Conceptual Interoperability Model (Adapted from Wang, Tolk and Wang, 2009)

The Figure above shows layers of conceptual Distributed simulation model interoperability discussed above in section 2.10.2. A quick run-through; *Level 0 (No):* Models built with this layer have no interoperability of any kind - internal or external. *Level 1 (Technical):* Data is exchanged between systems with the presence of technical connections between them. *Level 2 (Syntactic):* At this layer, models and systems have an agreement on a protocol to exchange the right data in a particular order but the meaning is not established yet. *Level 3 (Semantic):* Systems interoperating with this level can exchange terms that can be parsed semantically between them. *Level 4 (Pragmatic):* Systems interoperating here are fully aware of the

system state, processes, and meaning of data being exchanged. *Level 5 (Dynamic):* As the simulation time increases, a system with this level of interoperability can re-orient data consumed and produced based on the understood changing meanings. *Level 6 (Conceptual):* At the highest level, systems are interoperating with full knowledge of each other's processes, information, modelling assumptions, and contexts.

Within the distributed simulation theory, model composability is where the analyst selects various systems components, combined to fulfil simulation project requirements. (Petty and Weisel, 2003) defined it as "the ability to combine and recombine components into different simulation systems for a different purpose." To make M&S an entirely scientific area of discipline, a body of knowledge is required to be in place, which comprises methods in engineering and standards to follow in operations.

The interoperability issue will be among the main factors in the core proposed architecture for the cloud-based distributed simulation. The SISO put efforts in the M&S direction as we see in Taylor (2018) – the SISO-STD- 006-2010 Standard for COTS Simulation Package Interoperability Reference Models (IRMs) which is designed explicitly with Operational Research/Management Science (OR/MS) in mind.

Time Management

Time Management is one of DS's significant features, and it is a technique used to determine proper distributed simulation execution with required synchronisation. Notably, the DS uses three types of time. When referring to time management, it can be a physical, simulation, or *wallclock time* (Fujimoto, 1998). *Physical Time* refers to the time in the physical system, i.e., the system being modelled by the simulation. For example, in a simulation of the attack on Pearl Harbour, physical time might extend from midnight until 6 p.m. on December 7, 1941. *Simulation Time* refers to the simulator's representation of time. In the Pearl Harbour simulation, simulation time might be represented as a double-precision floating-point value that can hold values in the interval [0.0, 18.0] where a unit of simulation time corresponds to an hour of physical time. *Wallclock* Time refers to a time when the simulator is executed. For example, the Pearl Harbour simulator might require three and a half hours to execute. If it were executed in the afternoon of September 10, 1996, wallclock time might extend from 1330 until 1700.

Typically, it assumes that a simulation is composed of a set of Logical Processes (LPs) representing different components of a physical system, which communicates via time-stamped messages. A process in LP is described as a sequence of states and events, which change state in response to events that can be generated internally or arrived from another process (Nutaro and Sarjoughian, 2004). Figure 2-13 from Rizvi (2013) show several LPs interact through a communication medium. Simulation applications use Simulation Executives and interfaces to send and receive messages using a finite simulation time T_s.

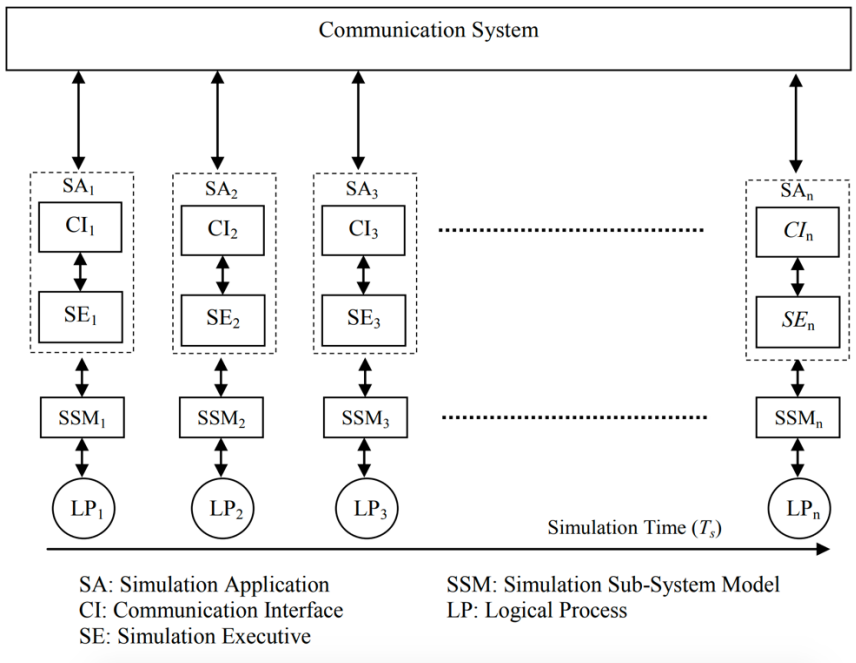


Figure 2-13 LP Architecture in Simulation Model (Adapted from Rizvi, 2013)

There are several processes in a physically distributed system and are connected via a directed channel. In DS, the concept is called a logical process that executes sequential code using two commands: *send* and *receive* (Misra, 1986). LPs uses *send* command to define the outgoing message and the channel to follow to a destination in sequence as they are sent. The sent messages have predefine time to reach their designation, which is determined arbitrarily. When LP is ready to *receive*, it waits until a message arrived from the incoming channels.

2.6.3 Distributed Simulation (DS) Methodologies

DS uses several LPs which are distributed over multiple computers and interconnected via a LAN, MAN, or WAN. Due to time management or synchronisation during DS execution and communication between these subprocesses, problems do occur and poses some challenges. For example, a network may be down, and data may be lost

alongside. The DS simulation coordinating centre must find a way to continue from a downtime using some recovery mechanisms. These mechanisms are spelt out in the DS implementation approaches, including conservative, optimistic, and real-time.

Conservative Approach – This approach resulted from algorithms developed in the late '70s by Chandy and Misra (1979), and Bryan (1977). It is referred to as *Chandy-Misra-Bryant (CMB)* and laid the foundation for parallel and distributed simulation. Conservative techniques are based on the idea of determining when it is safe to process an event. Suppose a process contains an unprocessed event with time-stamp T. That process can determine that it is impossible for it to later receive another event with a time-stamp smaller than T. In that case, the process can safely process that event. Processes containing no safe events must be blocked, which can lead to a deadlock situation, in general.

This approach has many sub-units. Those relevant to this research include the **Deadlock Avoidance** technique where *null messages* are exchanged among LPs during simulation run to avoid deadlock situations. Null messages do not usually represent or correspond to any physical system activity under study but are used for synchronization purposes. For example, when a null message with a timestamp of T_{null} is sent to LP_B from LP_A, it indicates a promise by LP_A that it will not send a message to LP_B with a timestamp less than T_{null}. Another one is the **Deadlock Detection and Recovery** mechanism developed by Chandy and Misra (1981). It is similar to deadlock avoidance only that this does not use or exchange null messages. This technique is used to detect when a simulation is in a deadlock. Another mechanism is sent to break the deadlock and free the simulation. The mechanism breaks the simulation deadlock by observing and making messages having the smallest timestamp that is safe to be processed. Last but not least is **Lookahead**, which refers to the ability to predict what will happen in a distributed simulation. The common form of lookahead is the minimum timestamp increment of an LP for processing any event.

Optimistic Approach - Contrast to conservative algorithms, optimistic algorithms allow causality errors to occur and then recover from them so that all events are correctly processed in order by the end of the simulation. The aim is to generate more parallelism to process the simulation faster. Time Wrap is a well-known and used optimistic protocol (Jefferson, 1985). With Time Warp, a mechanism detects causality error whenever an event message arrives and contains a smaller timestamp than that of the process clock (Jefferson and Sowizral, 1982) which is the timestamp of the last processed event. It

uses an event that causes a rollback and is called a straggler (Fujimoto, 1990). Recovery is achieved by reversing the effects of all events that were processed before the due time by the process receiving the straggler. The affected events are those with timestamps larger than that of the straggler.

Real-time Approach - In a DS, multiple simulations need to interact with multiple "players" and their responses, as close to real-time as possible. This goal requires different approaches to balancing processing and communication. Taylor (2018) revealed that these are classed as "Real-Time Approaches" and do not use time management as described above.

Several kinds of research in distributed simulation, especially the discrete-event area such as Jha and Bagrodia (1994), Fujimoto (2003), Park, Fujimoto and Perumalla (2004), Tang *et al.* (2005), Xu and McGinnis (2006) Wang *et al.* (2004), and Carothers and Perumalla (2010) are found to be focusing on performance evaluation of either conservative and optimistic approaches or both on various types of applications. Table 2-2 below shows a summarised comparison between conservative and optimistic approaches by the first two techniques (Vee and Hsu, 1999). It is noticed that the authors compared features implementation strategies such as parallelisation, synchronisation, lookahead, deadlock and others.

Table 2-2 A comparison between conservative and optimistic approaches (Vee and Hsu, 1999)

Strategy	Conservative Approaches	Optimistic Approaches
Principle	strict adherence to local causality constraint	allow violation of local causality constraint; provide rollback mechanism to recover if causality errors occur
Parallelism	allow less parallelism for exploitation	allow aggressive exploitation of parallelism
Synchronization	block LPs which may violate local causality constraint; may cause deadlocks if precaution is not taken	rollback mechanism to recover from causality errors; incur state-saving overhead
Deadlock	adopt avoidance strategy, detection-and-recovery strategy, or synchronous approaches	no deadlock problem
Lookahead	rely heavily on lookahead to achieve good performance; lookahead however might not be statically guaranteed but dynamically available	no dependence on lookahead to achieve good performance; lookahead can be exploited automatically with lazy cancellation or lazy reevaluation
Configuration of LPs	require static configurations by most existing conservative techniques	network configuration can be changed dynamically
Memory Requirement	require less memory than that of optimistic protocols	require more memory; more complex and complicated memory management
Implementation	straightforward implementation and data structures	notoriously hard to implement; complex data manipulations

Applications of Distributed Simulation (DS)

Based on his discussions with distributed simulation enthusiasts and personal experience with the available literature, Professor Stewart Robinson of Warwick Business School in the UK, over a decade ago carefully categorised (Robinson, 2005) the application of DS into four main headings; Model Execution, Data Management, experimentation, and Project Processes. The following Table 2-3 gave an overview of what falls under which category.

Table 2-3 Application areas of Distributed Simulation (Robinson, 2005)

Category	Application
Model Execution	- Distributing Model Execution - Linking Separate Models
Data Management	- Linking to databases and other software - Linking to Real-time Systems
Experimentation	- Gaming - Distributed Multiple Applications - Distributed Multiple Scenarios
Project Processes	- Sharing Models - Applications Sharing - Virtual Meetings - Searching for Model Components

2.7 Modelling and Simulation in Cloud Computing

DS research communities have been working in the areas of interoperability, model reuse, networked distributed simulation, and cloud-based simulation applications. This project reviews over a decade and recent publications in three different areas: Distributed Simulation, and Cloud-based Simulation. Moreover, the thematic areas reviewed are used to uncover the gap in the literature, which this research aims to address.

Strassburger, Schulze, and Fujimoto (2008) conducted a study to understand the relevance and economic potentials of distributed simulation. The survey collected practitioners' opinions concerning the current state-of-the-art and research challenges that need to be addressed. At the time of this publication, the Distributed Simulation/Distributed Virtual Environment (DS/DVE) adoption was limited. Nevertheless, the survey results show that the DS/DVE is believed to have high practical relevance for improving organisational processes and the overall product life cycle. Moreover, the ultimate practical relevance is

linking and integrating (possibly heterogeneous) computing resources for conducting large and complex distributed simulations as well as virtual distributed training sessions.

Distributed simulation (DS) is concerned with the execution of simulations over computing platforms that span a much broader geographic extent than parallel computers (Fujimoto, 2015). His paper introduces the concepts of parallel and DS and focuses on the concurrent execution of discrete-event simulation programs. The paper further gave the M&S field background that evolved and grew from its origins in the '70s and '80s and its active research opportunities today. The author gave an overview of parallel, and distributed research is presented ranging from a few works in the field aimed at addressing some identified problems such as synchronisation to recent research in executing large-scale simulations on supercomputing platforms. The article suggests directions for future research in the field of both parallel and DS.

Distributed Agent-Based and DES simulation of Emergency Medical Services (EMS) was proposed by (Nouman, Anagnostou and Taylor, 2013). The authors demonstrated how to develop a DS using the RePAST (https://repast.github.io) simulator and poRTIco (*http://porticoproject.org*) middleware opensource tools. The paper uses the SISO's COTS interoperability reference models (IRM) (Morse *et al.*, 2010) to connect discrete event simulation (DES) accident & emergency (A&E) model with an agent-based simulation (ABS) ambulance service model. A simplified version of the London Ambulance Service is used as a case study for applying the distributed ABS-DES architecture. The experiment is conducted under a controlled setting to maintain consistent communication between models (federates). They use HP desktop computers each equipped with a 1GBPS network card, Core i5-2500 processor at 3.30GHz, and 4GB of RAM. Each machine is loaded with MS Windows 7 OS, Java v1.7 JRE, poRTIco, and RePAST Symphony simulator. Each experiment scenario ran for a one-month simulation time and presented results of an average of five runs.

Ficco *et al.* (2016) developed a framework for integrating local and distributed hybrid simulation environments, which are widely used to simulate large-scale critical systems. The paper presents a simulation and emulation-based subsystems integration framework. Due to the challenges posed by the difference in time domains, a large number of involved entities, and communication overhead, the authors used HLA and performed integration in a more robust and standardised scenario. The experimentation adopts a cloud-based virtualisation platform to reasonably reproduce the architecture of a complex system on an adaptive and elastic controlled testbed.

The performance of DS strategies is examined (Fujimoto, 1989) using deadlock and avoidance detections. The author further investigates the recovery techniques using synthetic and actual workloads. The DS experiment consists of four key components; the application package that simulates the real system, the distributed simulators responsible for executing the application package. Next is the software to implement the system functions such as inter-process communications—lastly, the multiprocessor hardware on which the application package executes. The late '80s research demonstrated that distributed simulation algorithms could provide significant speedups over sequential event list implementations for some workloads.

Guan *et al.* (2019) propose a multi-layered cloud-based scheme to enable DS standards such as Distributed Interactive Simulation (DIS) and HLA. The authors give reasons to migrate DS to the cloud among which are *Resource Sharing* - where computing resources are provided on-demand during runtime. *Virtualisation* – Cloud computing virtualisation cover both the soft and hardware layers providing better isolation and manageability. *Scalability* – Cloud infrastructure often offers automatic resizing of virtual hardware. *Payment* – User can configure cloud resources on-demand using pay-as-you-go options. The work proposes a multi-layer cloud platform comprising of a five-layer stack, the raw resources layer, the integration and virtualisation layer, the simulation function layer, the user management layer, and the deployment layer. This scheme claims to address the challenges modellers face by hiding the management of the underlying cloud resources. During the experiment, a prototype of the proposed platform is implemented on a cluster, a single machine, and Amazon's EC2. Based on the experimental results, the paper concludes that the use of cloud technologies is a promising method to facilitate distributed simulations, especially when the network environment demands efforts towards optimisation and performance.

Concerning OR, Taylor (2018) reports a DS state-of-the-art. Various attempts by researchers in investigating the use of the cloud for simulation were presented. Examples include CloudSME, WS-PGRADE, mobile devices as a platform for DS in terms of energy consumption. The work argues the potential benefits and advantages of DS in the context of Operational Research (OR). The author also discusses future challenges such as those coming from Cyber-physical systems, Digital Twins, Industry 4.0, and Smart Environments.

A distributed simulation methodological framework for Operations Research and Management Science (OR/MS) was proposed by Anagnostou and Taylor (2017b). It attempts to bridge the gap between DS and OR/MS communities. The authors use an Emergency Medical Services (EMS) model to demonstrate how the framework eases the DS

implementation by non-technical OR/MS modellers. The research work employed the London Ambulance Service (LAS) as a case study. The models (federates) are built using High-level Architecture (HLA). The experiment involves Ambulance Service and Accident & Emergency (A&E) departments. The Ambulance is an Agent-Based Simulation (ABS) federate, and the A&E is a Discrete Event Simulation (DES) model. The author uses homogenous computing resources connected via Local Area Network (LAN). Each node is loaded with Microsoft Windows 7, 4GB RAM, Intel 3.30GHz Java 1.7 JRE, and portico v2.0. The work concludes that approaching DS from OR/MS point of view, it will make a massive experiment widely attractive to non-technical modellers.

HLA Development Kit Framework (DKF) was developed (Falcone, Garro, Taylor, *et al.*, 2017) to reduce the complexity associated with HLA-based distributed simulations. The framework allows for easy conceptualisation, definition, and build of HLA DS. In the tutorial paper, the authors guide developers through the required steps to defining and creating an HLA-based simulation for experimentation. The DKF aids modellers by allowing them to focus on their federate-specific needs and leave the HLA technicalities, which are taken care of by the framework. DKF aim to spontaneously coordinate the interaction between federates, simulation management, publish/subscribe. This framework has successfully experimented in the Simulation Exploration Experience (SEE) (Wei *et al.*, 2018) project since the 2015 edition. DKF can be placed and run on the existing HLA/RTI implementation, such as Pitch RTI (*Pitch Technologies – Pitch pRTI – a Certified HLA RTI*, 2017) and VT MÄK (*MAK RTI - VT MAK*). DKF is IEEE 1516-2010 compliant, and this allows developers to operate under the paradigm "write once and run anywhere" - its primary benefit. To promote DKF implementation, a domain-specific extension called SEE HLA Starter Kit (SEE-SKF) is developed by (Garro *et al.*, 2015). Finally, the paper compares developing HLA-based federates with and with DKF.

Carillo *et al.* (2018) presented Simulation exploration and Optimisation Framework for the cloud (SOF) - a framework that allows analysts to run and collect results for two kinds of optimisation scenarios; Simulation Optimisation (SO) and Parameter Space Exploration (PSE). The framework leverages the computing power of a cloud computing environment to accomplish effective and efficient simulation optimisation strategies for DS. The authors claim that SOF provides a set of functionalities that allows developers to construct their simulation optimisation strategy. The main objectives of the framework are *Zero Configuration* - the framework uses only Hadoop and SSH, and neither requires the installation of any additional software to the hosting platform. *Ease of Use* - the tool is transparent to the users, and they are unaware that the system operates in a distributed environment. *Programmability* - the simulation and the SO functionalities can be implemented using different toolkits, e.g.,

NetLogo or MASON or different programming languages supported by the hosting platform. *Efficiency* - can execute several independent simulations concurrently by adequately exploiting hosting platform available resources.

Cloud-Based Simulation (CBS) offers reliability, availability and scalability capable of executing simulations on the Cloud. Adapting this advancement poses challenges such as synchronisation, virtualisation and multi-tenancy overhead. In an attempt to mitigate these, Distributed Simulation for Cloud Computing (DSC) is proposed by Rajei *et al.* (2017). The multi-layer platform includes middleware and High-Performance Computing - HPC as a Service (HPCaaS). This proposal gave birth to Distributed Simulation as a Service (DSaaS) - a cloud service for large simulations that requires parallel executions of modules. DSaaS has many layers and implemented on the Platform as a Service (PaaS) which paved the way to cloud implementation and execution of PADS applications.

Chaudry *et al.* (2016) investigated a cloud-based framework for distributed simulation using WS-PGRADE workflows (Farkas, Hajnal and Kacsuk, 2014). The workflow uses CloudSME Simulation Platform (CSSP) to support the execution of varied sizes of the federation using a single large instance on the cloud. The authors test and verify the framework feasibility using a hybrid distributed simulation model of an Emergency Medical Service (EMS). The federation consists of one Ambulance (ABS) and six Accident and Emergency departments (DES) in the study area.

Another closest research effort towards cloud-based distributed simulation architecture is the MiCADO - *Microservice-based Cloud Application-level Dynamic Orchestration* framework proposed by Visti *et al.* (2016). The framework places microservices in lightweight virtualisation containers in worker nodes, and the orchestration and coordination mechanism is required to enable service discovery and performance management. The MiCADO generic architecture definition in this regard is to identify a modular and pluggable framework where different functionalities can be delivered by different components on-demand, and where these components can be easily substituted. The authors claim this solution to be technology-neutral that will not be depending on one particular component implementation.

A web-enabled High-Level Architecture (HLA) federate is proposed (Tu, Zacharewicz and Chen, 2011) after the release of the IEEE Standard 1516-2010. The research uses the poRTIco middleware tool with the idea to improve interoperability between components and agility. The paper explains the benefits found in the new standard such as fault tolerance support services, Extended XML support for FOM/SOM, Web Services (WSDL) support/API,

Encoding helpers, Modular FOMs, Smart Update rate reduction. The simulation of *WebServiceFederate* is used to facilitate the enterprise data exchange simulation, which fulfils one of the new features specified in HLA 1516-2010.

Guidelines for experimentation in the cloud was presented by NATO's MSG-131 team (NATO, 2015). They proposed the *MS Software as a Service (MS-SaaS)* - allows the MSaaS Consumer to use the MSaaS Provider's applications running on a cloud infrastructure. *MS Platform as a Service (MS-PaaS)* - allows the MSaaS Supplier to deploy onto the cloud infrastructure own-created or acquired applications created using programming languages, libraries, services, and tools supported by the MSaaS Provide. *MS Infrastructure as a Service (MS-IaaS)* - allows a consumer to utilise processing, storage, networks, and other fundamental computing resources. It allows the consumer to deploy and run arbitrary software.

Brito *et al.* (2016) proposed, designed, developed, and evaluated a distributed simulation platform. The research combines heterogeneous simulators and middleware - runtime infrastructure that are HLA-based. The authors assessed the platform with different scenarios using discrete event simulation. The proposed solution was aimed at DS environments executing high-performance large-scale heterogeneous and complex embedded systems. The experimental results indicate successful application in the power estimation of circuit design, robotic simulation, and wireless sensor networks. The hybrid platform combines many aspects of distributed simulation, such as network architecture and computation among many others.

Researchers in the DS field are promoting efficient ways to analyse systems using plug-and-play model components instead of building simulation models. Such an example is published by Jeffrey *et al.* (2007). The paper proposes an Open System Architecture for Modelling and Simulation (OSAMS). The primary motivation is to standardise the use of component-based interoperability technology to drastically lower the cost of development, operation, maintenance of next-generation models. The authors identify five technical areas crucial in establishing the SOAMS: flexible hierarchical composition structure, standard modelling framework, abstract polymorphic methods, distributed object techniques, consolidated trace file generation, and data logging. The publication discussed how OSAMS could be integrated with traditional distributed simulation standards such as HLA, TENA, and DIS. Targeted at defence and military applications, the architecture can well suit industry applications.

HLA is used with cloud-based services by Azevedo *et al.* (2015) proposed an agent-directed transportation management simulation platform to overcome the interoperability issues when combining several simulation tools and concepts to analyse systems. The proposal used HLA for interoperability among simulators. The paper aims to enable experts to diagnose complex problems cutting across domains, allowing co-simulation from different application domains.

Furthermore, other publications try to enable the integration of simulation resources amongst distributed models which are geographically separated. This leads to the development of DS standards such as **DIS** - a standard networking protocol for exchanging information among various simulation applications (https://www.mak.com/), **DSEEP** - a generalized systems engineering process for building and executing distributed simulation applications (https://www.sisostds.org/), and **HLA** - a more recent standard for interoperability among simulations, with interoperability and model reusability at the core. Non-software engineering modellers like those in operational research (Anagnostou and Taylor, 2017), however, may find it challenging to work with a distributed system for simulation project due to ever-evolving types of computing resources, network, and other technical components.

Cloud infrastructure is used to tackle the technical details, configuration, and maintenance of simulation environments where simulation applications run. Guan, Grande and Boukeriche (2019) proposed a multi-layered scheme to enable M&S based on different DS standards. The authors developed a deployment model aiming to ease technical installation, migration, configuration, and replication of simulation platforms. They proposed a self-managed lightweight terminal that can be used by modellers to manage simulation resources for direct usage. This simulation cloud architecture is composed of Deployment Layer, User Management Layer, Simulation Function Layer, Integration and Virtualisation Layer, and Raw Resource Layer. This work is also reported in the literature review section of this thesis.

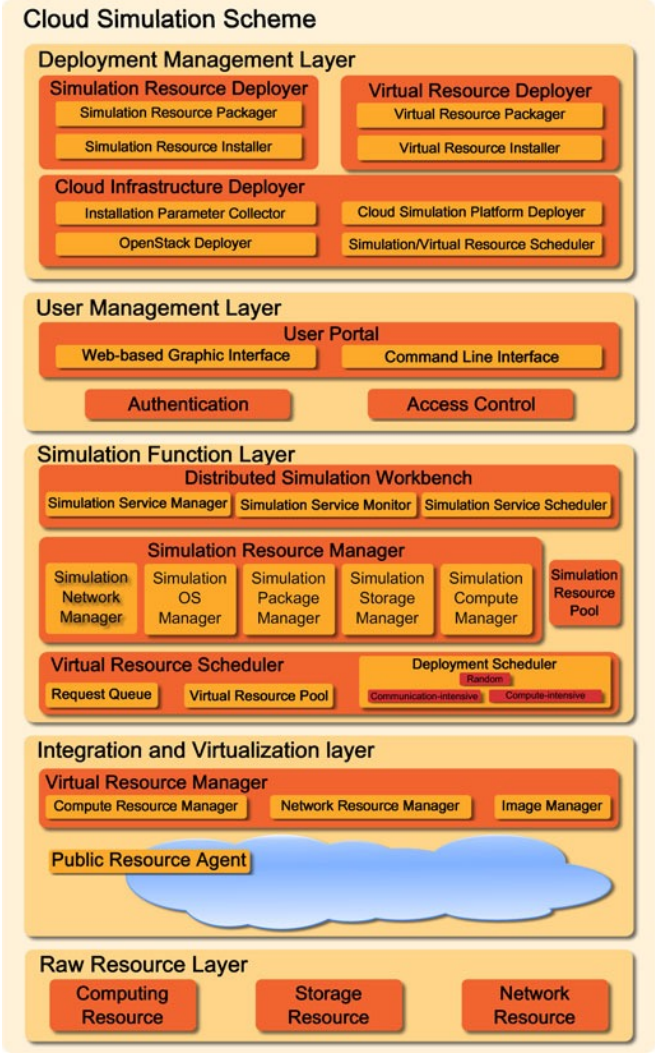


Figure 2-14 A multi-layered Cloud Simulation Framework (Adapted from Guan et al., 2019)

As shown in Figure 2-14 is from each layer in the framework proposes to ease the CBDS experimentation. The main layers in this framework are *Deployment Management Layer* – Fast and automatic deployment and migration of the proposed scheme in new environments is a priority. To achieve this, the Simulation Resource Deployer, the Virtual Resource Deployer, and the Cloud Infrastructure Deployer (as depicted in the Deployment Management Layer of Figure 2-14) cooperate to backup the current platform status. It packs and transmits core resources to new environments, and deploys the platform based on its previously saved status. *User Management Layer* - This layer focuses on security issues in the cloud environment. In this layer, an authentication and access control mechanism are proposed. The idea is to protect users against threats from within and outside the cloud. Besides, this layer also provides a web-based graphic portal and a command-line interface. Using it, users can access the simulation resources and the computing capability of the proposed cloud simulation platform. This user portal enables users to design, code, analyse and test complex distributed simulations via lightweight terminals such as laptops, tablets, or even smartphones.

Simulation Function Layer - This layer provides core functions and services that naturally enable and support different types of distributed simulation standards. It supports HLA (High-Level Architecture), DIS (Distributed Interactive Simulation), and DDS (Data Distributed Service). Others include the Distributed Simulation Workbench, the Simulation Resource Manager, and the Virtual Resource Manager. *Integration and Virtualisation Layer* - To utilise the raw computing capacity from diverse resources, the Integration and Virtualization Layer is a designed environment. The Compute Resource Manager deals with the establishment of virtual instances based on the selected scheduling algorithm and the characteristics of the user-defined virtual instances. The Image Manager handles the operating system and simulation-related soft-layer issues. *Raw Resource Layer* – This is a pool of untreated resources, such as computing resources, storage resources, and network resources. The whole cloud simulation platform is built on top of these resources. In this layer, physical hosts are not configured and coordinated, as they contain only an operating system and essential software that such an operating system brings.

As established in chapter one, section 1.2, many simulation projects require high-performance computing (HPC) resources and a lot of time for experimentation execution. Acquiring HPC can be expensive for Small and Medium Enterprise (SMEs). Cloud computing concept presents an alternative to the HPC investment in hardware and software. Furthermore, developing cloud-based simulation application is demanding and costly too, for SMEs and non-technical modellers. In an attempt to overcome this challenge (Taylor , 2018) designed and proposes the CloudSME Simulation Platform (CSSP).

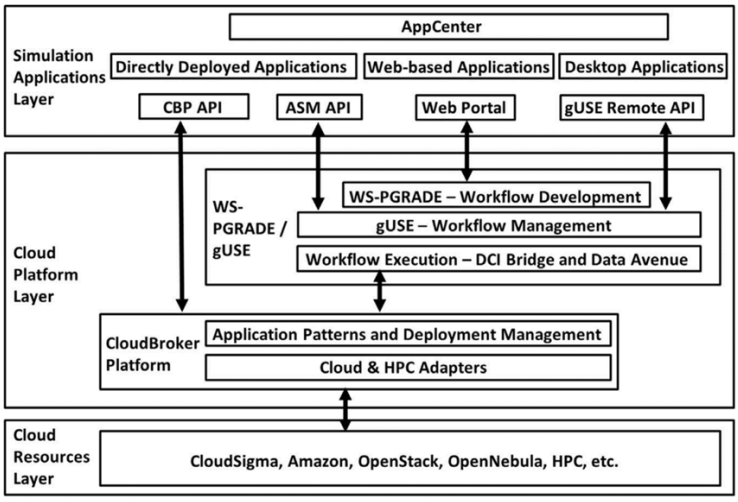


Figure 2-15 The Layered CloudSME Simulation Platform (Adapted from Taylor, 2018)

Figure 2-15 shows the CSSP, a layered architecture, which is created from existing technology; CloudBroker (*www.cloudbroker.com*), WS-PGRADE/gUSE (Gottdank, 2014) and extended to have CloudSME AppCenter. The generic CSSP is aimed at commercial vendors

and consulting companies offering simulation software services. Though it can be used for research, it is targeted at scenarios supporting different types of simulation software applications. For example, it works well with process simulation, computational fluid dynamics (CFD), and computer-aided design (CAD). A useful feature of the layered design in Figure 2-15 is the CSSP. It allows commercial vendors to utilise different cloud providers, which means they port their applications once and switch between multiple cloud resources or use them simultaneously. The three layers in this deployment are *Simulation Applications Layer* that allows software vendors to deploy and present simulation products to end-users as Software as a Service (SaaS) in a wide range of scenarios and deployment models. *Cloud Platform Layer* that provides access to multiple heterogeneous cloud resources and supports the creation of complex application workflows — a Platform as a Service (PaaS) to create and execute cloud-based simulations. *Cloud Resources Layer* represents the Infrastructure as a Service (IaaS) clouds connected to the platform.

These publications presented many aspects of distributed simulation run locally and in the cloud. However, a gap in the literature is exposed due to a lack of evidence of study on how the cloud environment can be used to deploy, submit jobs and run multiple DS runs. Those challenges listed in chapter one, section 1.2 presents a barrier and probably discourages analyst from simulating large-scale systems. This could be due to the complexity in cloud infrastructures (Avram, 2014) such as dependencies, network environment, protocols, and other resources management.

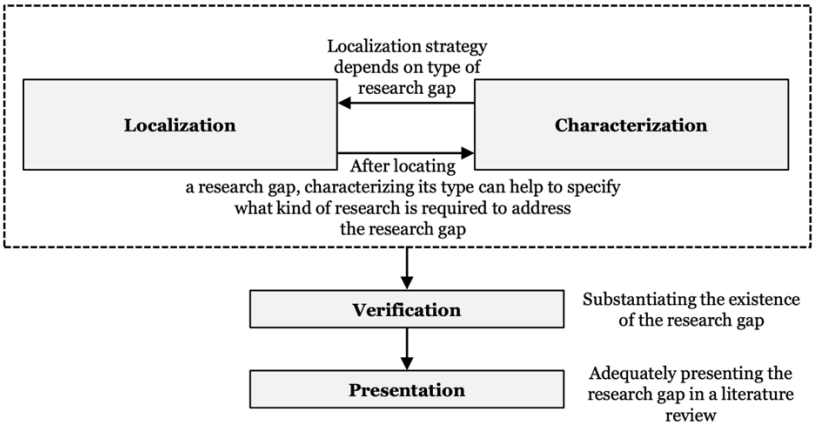


Figure 2-16 Framework for identifying research gaps in literature reviews (Adapted from Müller-Bloch, C., & Kranz, J. (2015))

Müller-Bloch and Kranz (2015) developed a framework for identifying gaps in the literature review show in Figure 2-16. The figure illustrates that research gap identification is separated from the localisation of the research gap. The authors refer to *localisation* as information suggesting a gap, derived from the literature review but require further research

to be resolved. At the same time, *identification* has broad meaning comprising localisation, characterisation, verification, and presentation.

While there is clear evidence on the reviewed literature that there has been much research into the distributed and cloud-based simulations, no single cloud-based deployment architecture or cloud-compatible federate (stepwise) development framework was reported, exposing a theoretical literature gap in the study. This project employs the framework mentioned above. It evaluates the contributions based on the selected and domain-relevant criteria. As they appear in Table 2-4, the comparison criteria are Cloud-Base DS, Simulation Paradigm, Architecture/Framework, Standard and Domain. None of the reported works provides methodological framework with steps to build cloud-based DS or CBDS deployment architecture as can be seen from the comparison table below. This is important to modellers as researchers believe that building DS is challenging, and when the cloud is targeted, the difficulty increases, which discourages them from adapting CBDS. To bridge the identified gap, this research proposes a development framework and scalable cloud-based deployment architecture. The table is purely based on the reported publications as the author could not find an authorised criteria on how to sort and point out research gap in a tabular form.

Table 2-4 A comparison of some reviewed publications with components of CBDS

Source	DS/Cloud Simulation	Simulation Paradigm	Has Architecture/ Framework?	Standard	Domain
(Ficco *et al.*, 2016)		Hybrid (ABS/DES)		HLA	General
(Guan, Grande and Boukerche, 2019)	Cloud-based DS		Layered Architecture	HLA/ DIS	General
(Nouman, Anagnostou and Taylor, 2013)	LAN-Based DS	Hybrid (ABS/DES)		HLA	Health
(Fujimoto, 1989)	LAN-Based DS				General
(Falcone, Garro, Taylor, *et al.*, 2017)	LAN/WAN		Development Framework	HLA	General

(Carillo *et al.*, 2018)			Simulation Optimization Framework		General
(Anagnostou and Taylor, 2017b)	LAN-Based DS	Hybrid (ABS/DES)	Developed Framework	HLA	Health
(Rajaei, Alotaibi and Jamalian, 2017)	DSaaS				General
(Chaudhry *et al.*, 2016)	Cloud-Based DS	Hybrid		HLA	Health
(Visti *et al.*, 2016)	Cloud-Based Simulation		Microservices-Based Framework	HLA	General
proposed (Tu, Zacharewicz and Chen, 2011)	Cloud-Based DS			HLA	General
(NATO, 2015)	MS-SaaS				General
(Taylor *et al.*, 2018)	Cloud-Based Simulation		Layered Architecture		General
(Brito *et al.*, 2016)	LAN-Based DS			HLA	Electronics
(Jeffrey *et al.*, 2007)			Open System Architecture	HLA, TENA, DIS	Military and Industry
(Rossetti and Barbosa, 2015)			Simulation Platform	HLA	General

Although several cloud-based simulation tools, methods, and concepts are reported in the literature review, Researchers such as Fowley, Phal and Zhang (2013) believe that choosing an existing or developing cloud architecture or development framework depends on the research problem to solve. *Therefore, this project proposes a novel **DI**stributed Simulation*

***C**loud Architecture for **E**xperimentation (**DICE**), which extends the established development concepts of running DS on a local environment or single cloud (running single federation on a single cloud) to the multiple distributed cloud (running single federation on many clouds) infrastructures.* From the M&S perspective, non-technical modellers can benefit from the proposed deployment architecture and the development framework. The new approach attempts to ease the development, deployment, and execution of distributed simulation on the cloud using existing technologies and cloud deployment models.

2.8 Reiterating the Research Questions

Conducting successful experiments with simulation models requires careful planning, which allows analysts to understand the result more effectively. What is the purpose of a simulation project? What should the output performance measure be? What is optimal system configuration? And measuring how changes in the input affects the output are some of the questions modellers ask when designing a simulation experiment (Kelton and Barton, 2003).

Systems study using design simulation techniques carefully can yield a useful result, and that helps organisations make informed decisions. There are established experimental methods, framework and architecture for both physical and nonphysical experimentations. Researchers reported simulation-specific experiment design approach such as perturbation analysis (Johnson and Jackman, 1989). Frequency-Domain Method (Crum *et al.*, 1998; Chavez *et al.*, 2010; Mistry *et al.*, 2019), (Donohue, 1994). The IEEE recommended Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE, 2011), and experimentation applications such as CloudSME (Taylor *et al.*, 2018). This research featured the last two, which are closely related; DSEEP and the CloudSME layered architecture for simulation experimentation.

2.9 Simulation Study Life Cycle

Designing and developing M&S can be a complicated activity. This phenomenon continues to pose challenges to simulation project stakeholders and other domains such as system analysts, developers, project managers, and engineers. Researchers have proposed and developed some stepwise guidance on how to execute a simulation project, starting from the project initiation stage, moving through conceptualisation, design, development, execution, result, and experimentation report stages. For example, Anu Maria (1997) explained 11 steps for the schematic simulation study in their paper. How developers go through the stages and iteration, depends on the system under study. Figure 2-17 shows the real-world system and the simulation study, which altered the former.

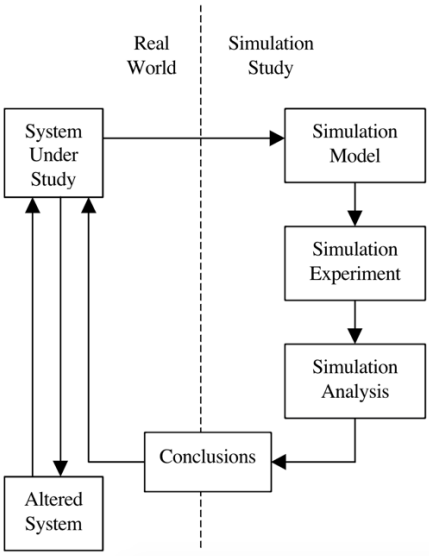


Figure 2-17 Steps in Simulation Study (Adapted from Maria, 1997)

The 11 steps; Identifying the problem, Formulate the problem, Collect and process real system data, Formulate and develop a model, Validate the model, Document model for future use, Select an appropriate experimental design, Establish experimental conditions for runs, Perform simulation runs, Interpret and present results, and Recommend a further course of action. As seen in the figure, these steps are categorised into Simulation Model, Simulation Experiment, Simulation Analysis, and Conclusions. Robinson (2001) is among the authors who published stepwise simulation lifecycle in Figure 2-18. In this paper, he reported DES perceived as 'hard' technique. The author used DES model to help improve user support helpline service at Warwick Business School. The work adopted a methodology with three main stages: sub-activities and validation are performed from the start of the project to the end. The author did not reveal the framework's details. However, the key stages clearly show the fundamental activities carried out in a simulation project. The stages are conceptualisation, model development, and facilitation. These will be dealt with in later sections.

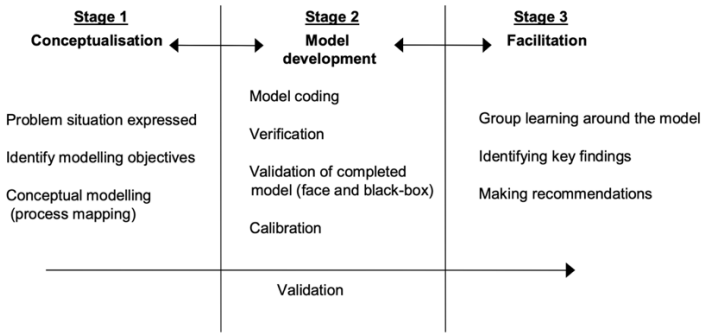


Figure 2-18 Summary of a Model Development Steps (Adapted from Robinson, 2001)

Banks *et al.* (2013) in a book presented basic steps in simulation study illustrated by Figure 2-19. The framework implementation may vary depending on the system under review and the objective of the study. Some steps in this flow can be repeated until the analyst achieves the desired outcome. The first items on the chart are the Problem Formulation, where the analyst and the simulation sponsoring organisation define the problem or system to study for improvements. When an agreement is reached, and objectives are set, the designers can concurrently carry out conceptualisation and data collection simultaneously and translate them into simulation models for the study. The simulation development steps continue. Some are repeated until the final implementation where the project may be terminated for another cycle.

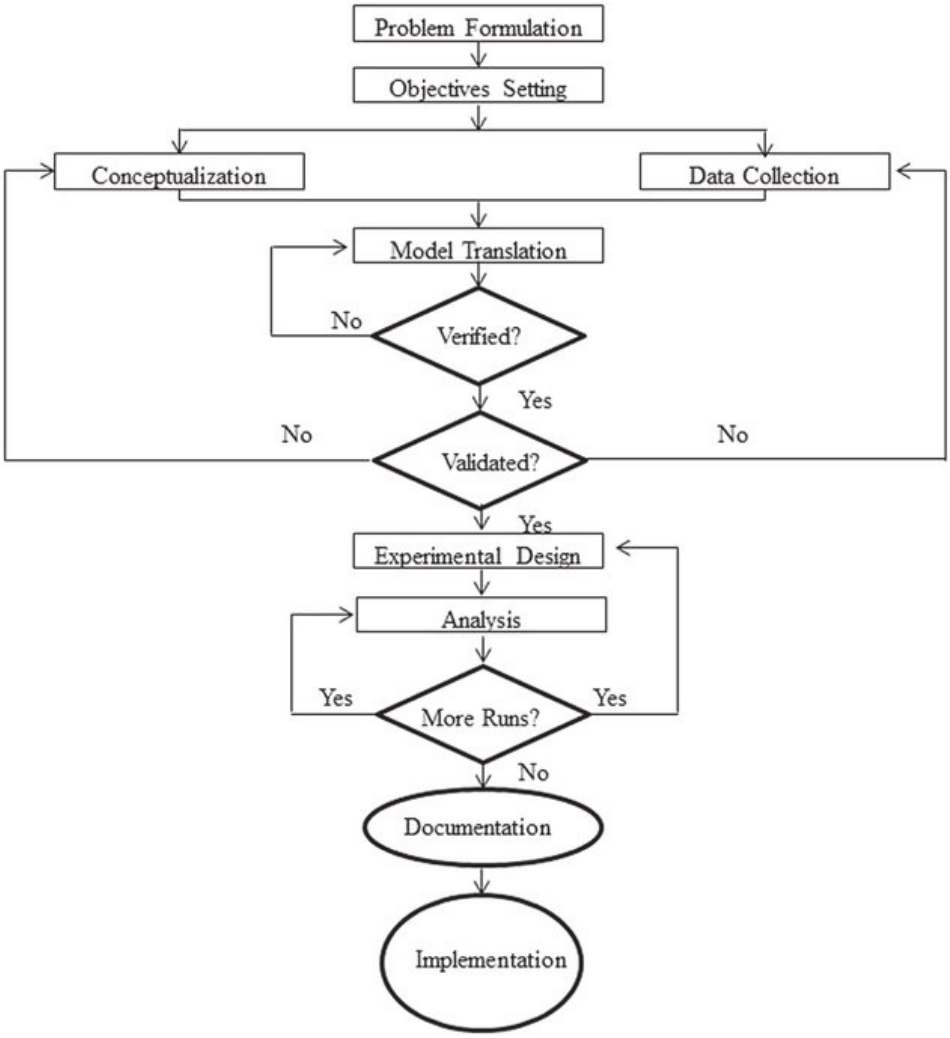


Figure 2-19 Steps in Simulation Study (Adapted from Banks et al., 2013)

The "*More runs?*" step strangely have two "Yes" branches. The book author explains that this is based on the analysis of runs that have been completed. The analyst determines whether additional runs are needed or follow another design those additional experiments should follow.

Moreover, Balci (2012) proposed a detailed simulation life cycle framework that enables analysts to view simulation project design and development from four critical perspectives. These are the process, product, people, and the project itself. In Figure 2-20, it can be observed that each step is accompanied by quality assurance and verification and validation. The nine steps involved are problem formulation, requirements engineering, conceptual modelling, architecting, design, implementation, experimentation, and presentation. These stages are not strictly sequential. Depending on the situation, there is flexibility for a step or set of steps to be repeated when the need arises. Balci used notations to flows to indicate which activity happens at every step as the project progress.

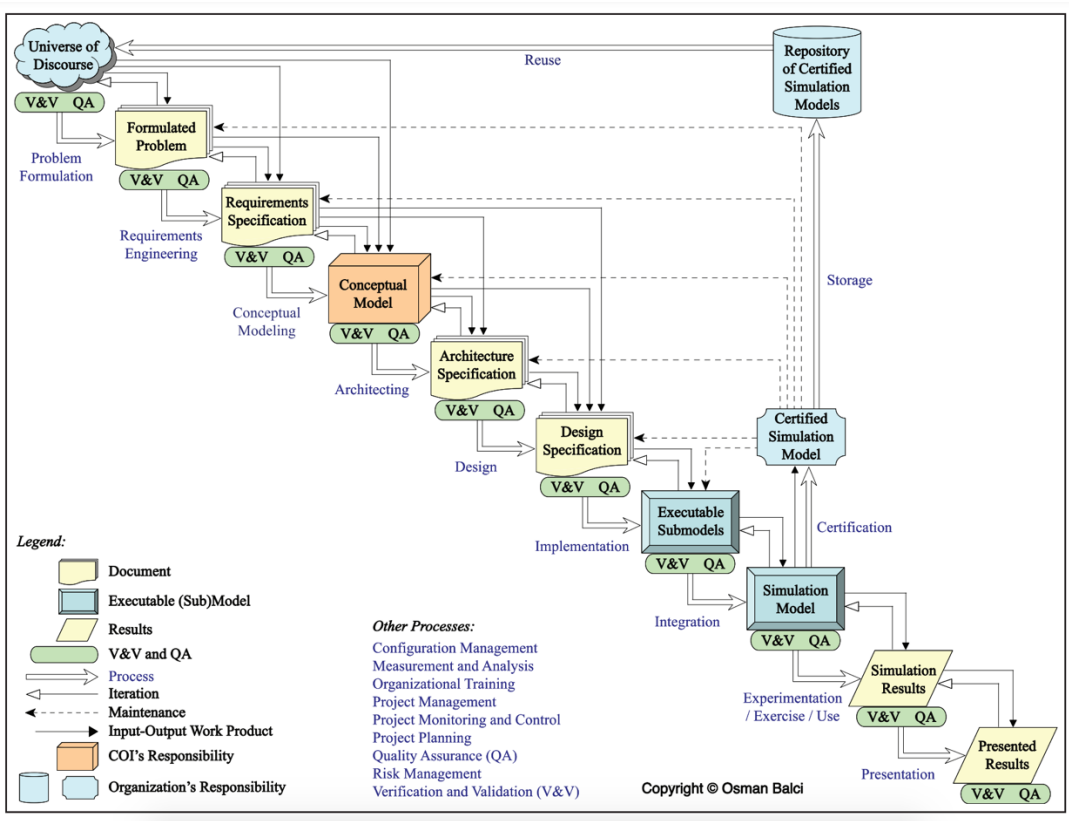


Figure 2-20 Modelling and Simulation Life Cycle (Adapted from Balci, 2012)

From another perspective, developing DS is a demanding and complex task. In 2003, the Simulation Interoperability Standard Organization (SISO) sponsored the IEEE Standards Association to develop recommended practice for DSEEP. The updated standard document – the IEEE Std 1730-2010 states that DSEEP describes a high-level framework for developing and executing distributed simulation environments. The DSEEP intends to specify a set of guidelines for the development and execution of these environments that stakeholders can leverage to achieve the needs of their application (IEEE, 2011).

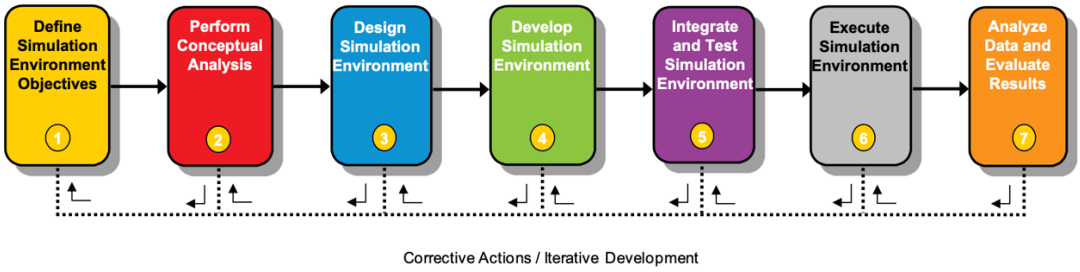


Figure 2-21 Engineering and Execution Process (DSEEP), Top-level Process Flow (Adapted from IEEE, 2011)

The seven DSEEP steps shown in Figure 2-21 (Topçu *et al.* 2016) began with the name Federation Development and Execution Process (FEDEP) ("IEEE Recommended Practice for Hig- Level Architecture (HLA) Federation Development and Execution Process (FEDEP)" 2003. Today, it can be implemented with various DS interoperability standards like Test and Training Enabling Architecture (TENA) (Noseworthy, 2011), High-Level Architecture (HLA) (Falcone, Garro, Taylor, *et al.*, 2017), and Distributed Interactive Simulation (DIS) (Mccall and Murray, 2010). The standard's seven steps can be executed in a linear, spiral, or iterative fashion depending on the DS project is being studied. Table 2-5 tabulates the seven iterative steps. Introductory description of activities at each stage is as follows.

Table 2-5 Tabular view of the DSEEP (Adapted from IEEE, 2011)

Step	(1) Define simulation environ-ment objectives	(2) Perform conceptual analysis	(3) Design simulation environment	(4) Develop simulation environment	(5) Integrate and test simulation environment	(6) Execute simulation	(7) Analyze data and evaluate results
Activities	Identify user/ sponsor needs Develop objectives Conduct initial planning	Develop scenario Develop conceptual model Develop simulation environment requirements	Select member applications Design simulation environment Prepare detailed plan	Develop simulation data exchange model Establish simulation environment agreements Implement member application designs Implement simulation environment infrastructure	Plan execution Integrate simulation environment Test simulation environment	Execute simulation Prepare simulation environment outputs	Analyze data Evaluate and feedback results

Step 1 - Define simulation environment objectives. The user, the sponsor, and the development/integration team define and agree on a set of objectives and document what

must be accomplished to achieve those objectives. *Step 2* - Perform conceptual analysis. The development/integration team performs scenario development, conceptual modelling and develops the simulation environment requirements based on the characteristics of the problem space. *Step 3* - Design simulation environment. Existing member applications that are suitable for reuse are identified. Design activities for member application modifications and/or new member applications are performed. Required functionalities are allocated to the member application representatives, and a plan is developed for the development and implementation of the simulation environment. *Step 4* - Develop a simulation environment. The simulation data exchange model (SDEM) is developed. Simulation environment agreements are established, and new member applications and/or modifications to existing member applications are implemented. *Step 5* - Integrate and test the simulation environment. Integration activities are performed, and testing is conducted to verify that interoperability requirements are being met. *Step 6* - Execute simulation. The simulation is executed and the output data from the execution is pre-processed. *Step 7* - Analyse data and evaluate results. The output data from the execution is analysed and evaluated, and results are reported back to the user/sponsor.

DSEEP is a well-established methodology for designing and executing DS. The framework guides modellers on how to run simulations on local and networked machines. To deploy the same framework on the cloud, there are many factors and challenges to consider, such as IP addressing, security, WAN networking, and data exchange protocol. Extending DSEEP to cloud infrastructure is the focus of this project and are captured in the thesis' objectives number three and four as presented in chapter one, section 1.3. The design, development, and execution steps to achieve are presented in the next section - chapter four. Meanwhile, cloud-based simulation has been studied, and references are given above. For this thesis, CloudSME layered architecture is particularly interesting and academically linked to the proposed cloud infrastructure architecture and framework produced in this work.

The above M&S and DS development processes, the author believed to be systematic frameworks for actions, activities, and multiple sub-tasks that are required to conceptualise, design, build, and successfully execute simulation experiments within acceptable quality bounds.

2.10 Simulation Using Cloud Infrastructure within the Context of the RQs

Cloud computing offers the ability to provide computing services remotely to users via the Internet, easing them of the burden caused by managing computing resources and facilities (Fujimoto, Malik and Park, 2010). Cloud infrastructures provide means for an organisation to lease and conduct sequential simulation without spending enormously on

buying and maintaining high-performance computing resources. This opens more opportunities for parallel and distributed simulation communities that can exploit distributed computing technology by leveraging the available flexible computing platforms. There are reasons for DS to use clouds such as resource sharing, virtualisation, scalability, and flexible payment models. First, let us look at the cloud computing concept and then cloud-based simulation.

2.10.1 Cloud Computing

Cloud computing can be defined as the use of new or existing computing hardware and virtualisation technologies to form a shared infrastructure that enables web-based value-added services (Gibson *et al.*, 2012). Figure 2-22 shows different types of cloud deployment models where each fits particular user needs. The three models; infrastructure, platform, and software-as-a-service, are given below from the same author: *Software-as-a-Service (SaaS)* - Gives subscribed or pay-per-use users access to software or services which reside in the cloud and not on the user's device. *Platform-as-a-Service (PaaS)* - Offer access to APIs, programming languages, and development middleware which allows subscribers to develop custom applications without installing or configuring the development environment. *Infrastructure-as-a-Service (IaaS)* - Is the use of servers, storage, and virtualisation to enable utility-like services for users. Security is a big concern within IaaS, especially considering that the rest of the cloud service models run on top of the infrastructure and related layers.

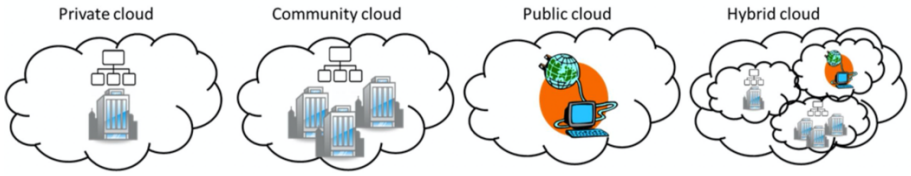


Figure 2-22 Cloud deployment models; Private, Community, Public & Hybrid

Elasticity is the main concept of Cloud Computing. That is, the use of computational resources, storage, applications, etc. that can be instantly increased according to user needs and ceases when the user does not need these services (Mell and Grance, 2011; Taylor and Anagnostou, 2014). Because cloud computing attempts to provide convenient access to on-demand computing resources, minimal management effort, or service provider interaction, the authors opined that simulation could be implemented as a cloud service using licences running on cloud computers on a "pay-as-you-go" basis. However, while this is a potentially excellent deal for the end-user, the simulation company would still need the expensive in-house expertise to provide this.

2.10.2 Cloud-Based Simulation

Researchers and developers come up with the Cloud-based Simulation platform for Manufacturing and Engineering (CloudSME). The project attempted to make cloud computing more feasible for M&S. It is still a strong belief that the use of computing infrastructure, such as cloud computing, opens opportunities to reduce computation time by distributing the workloads over the cloud, on-demand (Hwangbo and Lee, 2016). Further than that, to promote composable models and speed up the simulation, researchers are optimistic in achieving that even better with Modelling & Simulation as a Service (MSaaS).

Liu *et al.* (2012) presented their proposal on the Cloud-based Simulation (CSim) architecture. It covers the software involved in the whole process of M&S by providing the Modelling as a Service (MaaS), the Execution as a Service (EaaS), the Analysis as a Service (AaaS) and the reuse of available simulation resources with the aid of the Simulation Resource as a Service (SRaaS). These innovative leaps will go a long way in easing cloud simulation service provision and attracts more practitioners in the domain.

Cloud-Based Distributed Simulation (CBDS) as indicated earlier, appears to be an understudied technique. However, some publications have given some basis to the concept such as Simulation as a Service – SIMaaS (Tsai *et al.*, 2011; Azevedo, Rossetti and Barbosa, 2015; Shekhar *et al.*, 2016), Modelling and Simulation as a Service – MSaaS (Fujimoto, Malik and Park, 2010; Buora, Giusti and Barbina, 2014; D'Angelo, 2014; NATO, 2015; Wang and Wainer, 2016; Prochazka and Hodicky, 2017) and Distributed Simulation as a Service – DSaaS (Rajaei, Alotaibi and Jamalian, 2017).

2.10.3 Cloud-Based Simulation Method

Cloud computing is a trend in IT that moves applications and data away from traditional desktop and portable personal computers into large "invisible" data centres. Dikaiakos *et al.* (2009) defined cloud computing as applications delivered as services over the internet. Also, the actual cloud infrastructure — the systems software and hardware in data centres that are providing these services. With these facilities gaining wider acceptance serving many purposes, the simulation research communities began to explore the benefits offered by the cloud concept as contained in (Mell and Grance, 2011); on-demand self-service, elasticity, and broad network access.

As discussed in chapter one, above, research involving distributed simulation has attracted many researchers and publications are coming with a solution to different problems. Some notable cloud-based simulation works include; (Fujimoto, Malik and Park, 2010; Tsai *et al.*, 2011; Buora, Giusti and Barbina, 2014; D'Angelo, 2014; Azevedo, Rossetti and Barbosa, 2015; NATO, 2015; Shekhar *et al.*, 2016; Wang and Wainer, 2016; Prochazka and Hodicky, 2017; Rajaei, Alotaibi and Jamalian, 2017). This research is added to these efforts in using cloud infrastructure for large and complex experimentation for operational system analysis.

2.10.4 Cloud-Based Simulation

Modelling and Simulation (M&S) is one of the techniques used by analysts for decision support in many domains. Research communities have seen implementation of DS applications, usually for predictive and perspective analytics (Lustig and Dietrich, 2010). M&S community uses DS to gain information without interrupting existing or proposed new system, understanding the systems operational behaviours under various conditions and as a test tube before a new system or policy implementation. DS allows researchers to design, build and run multiple simulations.

Recently, cloud computing technologies are making an impact on modelling and Distributed Simulation (DS) by enabling on-demand network access to a variety of computing resources and services via the Internet. Authors, Onggo and Selviaridis (2017) have noted that the cloud-based M&S (CBMS) literature has focused on how to use existing technologies to develop CBMS tools. As published, despite its potential benefits, some researchers community rarely uses DS in areas such as healthcare and manufacturing. There are factors affecting adoption by users. For example Anagnostou and Taylor (2017) identified two main reasons; technical complexity required for implementation and the difference between the world views DS and M&S communities. World views were discussed in section two.

Arguably, some modellers lack software engineering training, which makes it challenging to adapt DS during analysis due to the extensive technical training and experience required to implement. When cloud technology is added to the DS sophistication, it will become even more challenging. However, pieces of evidence have shown that scholars contributed solutions to enable modellers to use the cloud for simulation. For example, Delen and Demirkan (2013) presented how service-oriented decision support systems (DSS) can be developed and run on cloud infrastructure. The

work mimics the three cloud service models; Data as a Service (DaaS) allows organisations to have on-demand access to data over the cloud. Information as a Service (IaaS) delivers information from multiple sources using cloud services. Lastly, Analytics as a Service (AaaS) helps organisations use simulation models in the cloud as a component of decision-making.

Furthermore, cloud-based simulation possibility for M&S community is backed by (Fujimoto, Malik and Park, 2010) who believed the clouds offer the potentials to make parallel and distributed simulation capabilities more accessible to non-technology expert users. While implementing cloud-based simulation, certain aspects need to be addressed that are key to successful and useful adoption by modeller. For example Johnson and Tolk (2013) identified five of those concerns; technical view, governance view, business view, security view, and conceptual view. This thesis focuses on the technical view, which deals with requirements such as the protocol, cloud infrastructures, interoperability, data exchange formats, and networking environment.

2.10.5 Potential Benefits of Cloud-Based Simulation

Research publications suggest that simulation is one of the system evaluation methods of choice in many fields. Systems are getting more complicated due to several reasons listed in item 3.6.3 above. One way of coping up with the increasing power-demand coming from the simulation scenario nowadays is to make use of more processor units, running on different architectures and dispersed around a larger area (Mihai, Valentin and Legrand, 2011). The idea behind DS is to use a set of execution units in a simulation. These execution units are responsible for a part of the simulation (a subset of the entities that compose the simulated system) and their interactions (D'Angelo and Bracuto, 2009). Moreover, the high-performance computing resources needed to effectively run a DS require, among many other things, a considerable investment in hardware and software. Cloud computing services present a viable alternative DS modeller through the on-demand network access to configurable resources (Mell and Grance, 2011) and pay-as-you-go payment option (Barbosa and Charão, 2012).

Using cloud infrastructure for distributed simulation is a relatively new field. It has not been studied in-depth as this work intends to. This research believes that the Cloud-Based Distributed Simulation (CDBS) will offer benefits to practitioners. For over two decades, researchers have identified and classified potential benefits of Web-based, as it is earlier called, to modellers. These include (Whitman, Huff and Palaniswamy, 1998; Leong *et al.*, 2000; Rao and Wilsey, 2000; Kuljis and Paul, 2001; Miller *et al.*, 2001;

Yücesan *et al.*, 2001; Byrne, Heavey and Byrne, 2010; Guan, Grande and Boukerche, 2019; Sokolowski *et al.*, 2019). These publications mostly agree on the following.

a. **Model Reuse -** One of the claimed benefits of DS is the reuse of the existing model. Through the HLA and appropriate middleware configuration, the web can support linking and exchanging data between models within simulation experiments.
b. **Interoperability -** Through the Application Programming Interface (APIs) web-based DS can seamlessly integrate and interoperate with existing and future cloud applications residing on local or remote servers.
c. **Ease of use –** Web enables easy navigation and use to obtain data and information by most users. Many web applications hide the underlying technical complexities to the user, and this characteristic makes web activities familiar to interact with and control, such as web-based DS.
d. **Collaboration -** Developing CBDS in some cases may require collaboration by a team located in separate geographical locations. With web technologies growing, modellers can communicate and develop a large simulation model, which may reduce the model design and development time and cost.
e. **License and Deployment -** Through cloud concepts, many applications are accessible using a browser. Web-based simulation services can be rented for a certain period such as Application Service Provider (ASP) and Software as a Service (SaaS). This saves in the then prohibitive investment of time and money.
f. **Cross-Platform Capability -** CBDS developers can focus on the model logic and do not have to worry about the client's platform. Web-based simulation may be configured to be accessible from any device, any operating system or any browser with required network access.
g. **Control Access -** Access Control List (ACL) can be used to keep an inventory of CBDS users and grant them permission to access a whole or a portion of the model, simulation, or application.
h. **Wide Availability -** CBDS can be accessed 24/7/365 and from any device with internet connectivity. This means CBDS management and control can be done within and outside working or office hours.
i. **Visioning and Maintenance -** Web-based simulations can be modified and instantly update models - real-time. With cloud unlimited storage capability, versions of experiments can be kept and roll-back when the need arises.

Many trends are contributing to the increase in complexity, reductions in inventory, rising outsourcing deepening information technology, expanding horizontal integration,

ever more sophisticated products, and escalating demands of customers. Each of these trends increases the range of possible outcomes that must be considered by decision makers, while simultaneously reducing the time available for choices (North and Macal, 2007). This project further identified more advantages of CBDS to the community of practice based on the current technology trend and innovations. These include but not limited to the following.

a. Easy integration to industry 4.0 tools and technologies for broader experimental capabilities
b. Enable real-time on-demand access to distributed simulation models for collaboration
c. Update to simulation models and data can happen anywhere, anytime.
d. Further, reduce the technical complexity faced by non-technical modellers.
e. Enable on-demand access scheduled and deadline-based computing resources for scalability and elasticity.

Naturally, some of the authors gave the downside of the web-based simulation such as loss in speed, Graphical User Interface (GUI) limitations, Security vulnerability, licence restrictions, and simulation application stability. Though these are areas of concern and can be useful research challenges in the field of CBDS, this thesis did not cover them in the aim and scope.

2.11 Chapter Recap

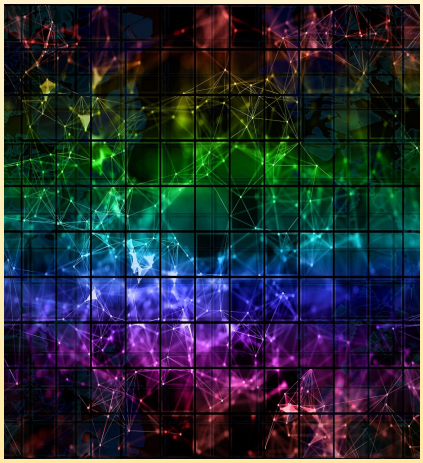
This chapter presented related works published in various academic, governmental, and industry platforms. The section acquaints the reader with the gap identified in the literature, which makes this research in Cloud-Based distributed simulation a worthy academic endeavour. The work aims to come up with a cloud-based architecture to address, at least, one of the challenges mentioned above in the simulation field. Various history, timelines, terminologies, technologies, approaches, and concepts in M&S, DS, and Cloud were presented. The next chapter will report the research methodology employed to achieve the project aim. Perspectives, outputs, and justification for the chosen methods were discussed.

Once again, let us reiterate that there is very few cloud-based distributed simulation in research. This identified gap suggests the following research questions.

- **RQ1** - How can you deploy distributed simulation on the cloud?

- **RQ2** - What are the factors affecting the interoperability of distributed simulation on the cloud?
- **RQ3** - What are the factors affecting cloud-based distributed simulation experimentation speed?

It is observed that the challenges in the literature on distributed simulation is still standing and has gained less attention from researchers. Therefore, this work aims to address the RQs to fill in the identified knowledge gap.



CHAPTER THREE

RESEARCH APPROACH: DESIGN SCIENCE RESEARCH

Chapter 3 Research Approach: Design Science Research Methodology

3.1 Chapter Overview

The preceding chapter explains simulation and its rudiments, it identifies the challenges researchers have not yet addressed through a literature review and the various simulation tools available for analysis. This section of the thesis states the research design approach and offers possible alternatives to address the questions posed from the academic perspective. It also describes data collection and experimentation tools, methods of result analysis, and justifications for the chosen methods. The chapter further explains the cloud architecture development approach taken in this work and the case study method adapted to implement and evaluate the proposed framework and architecture.

3.2 Research Approach

Having previously identified a gap in the literature, the next step in answering the research questions is to establish a suitable methodology. This will be done by taking outputs from the previous section and the methodological approach in this work. Therefore, a discussion on the research problem, data, tools, and analysis method will lay a foundation that further builds the chapter.

The Problem

This research investigates how cloud infrastructures connect and run geographically distributed simulation experiments. It aims at addressing the practical challenges faced by the modellers. On the one hand, the complex technical skills and training required to design, develop, and run distributed simulations for large-scale system analysis are high. On the other hand, distributed simulation requires a high amount of computing resources for experiments, which are expensive to acquire and maintain. This thesis proposes a distributed simulation development framework as a guide and deployment architecture to run DS on the cloud, offering on-demand access to high-performance computing resources using pay-as-you-go models.

Data Collection

A suitable operational system case study is an emergency medical service prototype which is used to validate and evaluate these two proposals and run numerous cloud-based distributed simulations. The experiment is designed to run multiple cloud infrastructures with different configurations, that generate a considerable amount of quantitative data, precise

execution time in minutes for performance, and scalability analysis. This helps the researchers to understand the characteristics of the cloud infrastructure under various conditions and configurations. The work focuses on cloud performance when running the distributed simulation using a single cloud, multiple clouds or even a mixture of cloud and physical systems. For example, one experiment set up routes for all simulation traffic through a physical system where the WAN router is configured with all source and destination federates.

Experimental Tools

Opensource software and runtime environment tools are used to run experiments and generate exciting results. The model is developed using object-oriented Java on an eclipse development environment customised by the RePAST Symphony team. The runtime environment uses an opensource called poRTIco runtime infrastructure, which serves as the middleware to control the federation execution. The cloud platforms used for the fundamental research are CloudSigma, DigitalOcean, Amazon EC2, Scaleway, and Google Cloud Computing. Each has its characteristics which can be configured in the model to run successful simulation runs.

Method of Analysis

Tabulation of execution times are used to report and illustrate the results, they are purely experiments, execution time and federates scaling. The scope of this thesis also includes the measurement how long it takes to run distributed simulation models under varied configurations, such as running experiments on a single cloud or distributing over multiple cloud platforms. This is addition to the RQ2 and RQ3 is enquiring to address. Furthermore, another metric is how the traffic is affected when the simulation datagram traffic is routed purely on the internet or through a local (on-premises) router.

3.3 Research Paradigms

Research work always has a philosophical underpinning paradigm. The literature reported some of these paradigms but here, only four are reported that are relevant for the discussion - the positivist, interpretive, critical, and design research. Below is a brief introduction to each of these methods.

Positivist: In a generic sense, positivism is an ideology that adheres to the knowledge facts gained through some measurements and observation. Creswell (2011) believes it promotes that anything that cannot be observed or measured has little or no importance. Therefore, scientific knowledge is gained from accumulating data obtained from observation - theory-free and value-free." Bryman and Bell (2011) view positivism as an epistemological

position that advocates the application of the methods of natural science to the study of social reality and beyond.

Interpretive: Interpretivism argues that truth and knowledge are subjective, culturally, and historically based on lived experiences and understanding them (Ryan, 2018). A researcher can never be separate from their own experience, values, and beliefs. Therefore, these may inevitably influence how research data is collected, processed, and analysed the results.

Critical: This paradigm is based on the transformation of the condition of humanity amongst people. It assumes that reality is socially constructed. Research employing this approach aims to critique the status quo through structural contradictions with social systems, and in the process, restrictive social conditions are said to be alienated.

3.3.1 Design Science Research (DSR)

The DSR paradigm has its roots in the sciences and engineering of the artificial (Simon, 1996). It is fundamentally a paradigm that aims to solve problems. DSR seeks to increase human knowledge with the generation of design knowledge and the creation of innovative artefacts using innovative solutions to real-world problems (Hevner et al., 2004). DSR research paradigm, as such, has generated an upsurge of interest in the past two decades, precisely due to its potential contribution to the innovative capabilities of organisations. Moreover, it contributes to the much-needed sustainability transformation of society (vom Brocke et al., 2020). With the opportunity presented in this approach, by DSR approach, this thesis used it to design and propose the Distributed Simulation Cloud Architecture for Experimentation (DICE).

3.3.2 The DSR Framework

Using DSR as a research project aims to extend the boundaries of organisational and human capabilities by designing novel and innovative artefacts represented by models, constructs, methods, and instantiations of what they represent (Gregor and Hevner, 2013). This indicates that DSR aims to create a knowledge of how it should be constructed or arranged - designed by humans to achieve a desired set of goals.

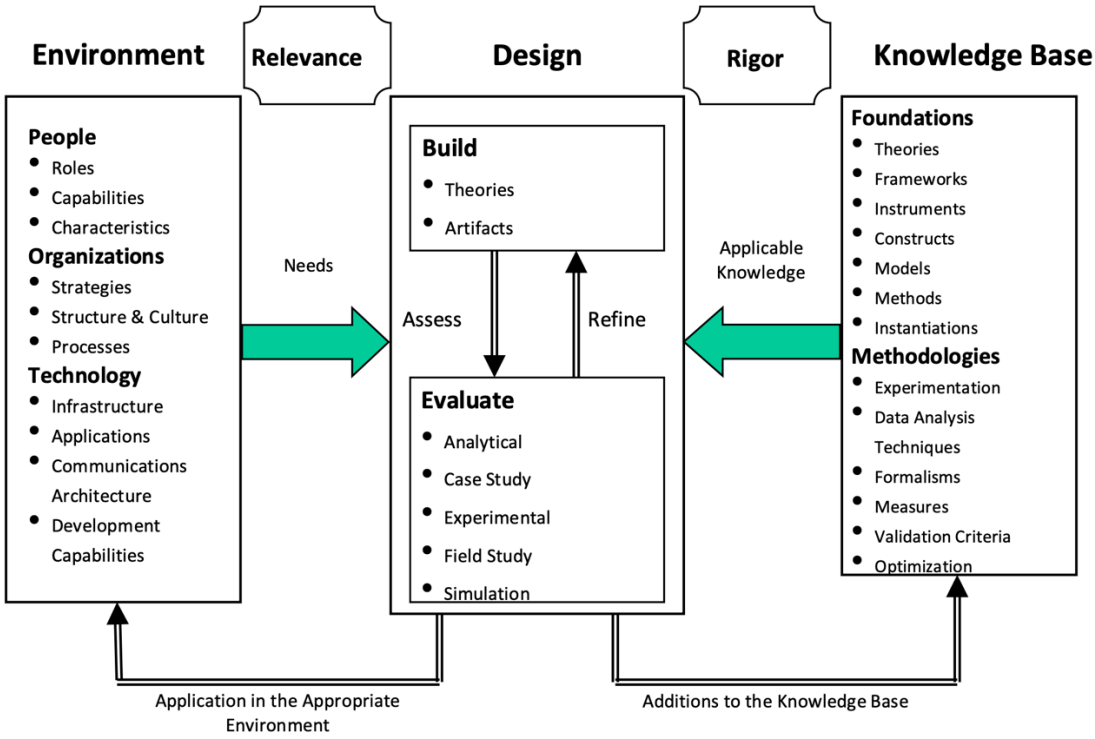


Figure 3-1 Design Science Research Framework (Adapted from Hevner et al. 2004)

For researchers to conduct design science research to scholarly standards, a DSR conceptual framework is published by Hevner et al. (2004) as shown in Figure 3-1 to help understand, execute, and evaluate DSR. The environment defines the problem space where the phenomena of interest are placed, which is a composition of people, organisations, and planned or existing technologies. It contains the problems, goals, tasks, and opportunities that define the needs perceived by an organisation's stakeholders. Needs are evaluated in the context of organisational structure, existing work processes, strategies, and culture, positioned in relation to existing infrastructure, applications, development capabilities, and communication architectures. These define the perceived research problem from the researcher point of view. The knowledge base section provides the "raw materials" from which the DSR is accomplished. The knowledge base is a combination of Foundations and Methodologies. Published research results from various disciplines provide foundational theories, frameworks, instruments, constructs, models, methods, and instantiations used in the build phase of a research study. Methodologies provide the guidelines on how to evaluate the research. Finally, Rigor is achieved by appropriately applying existing foundations and methodologies.

This research is linked to the “need” for CBDS deployment architecture solutions to be empirically investigated with case studies in modelling and simulation using cloud

infrastructure technology. In context, the DSR in this work also analyses the already available knowledge to solve a problem of interest. Such knowledge is established throughout this thesis in the form of frameworks, theories, or design artefacts - models, constructs, methods. The research is applied and guided by the design science research methodology (DSRM).

3.3.3 DSR Processes

Peffers, Tuuanen, Rothenberger, & Chatterjee (2007) proposed and published the most widely referenced model shown in Figure 3-2. The DSRM process model has six steps - (1) problem identification and motivation, (2) definition of the objectives for a solution, (3) design and development, (4) demonstration, (5) evaluation, and (6) communication. The authors also include four possible entry points - (1) problem-centred initiation, (2) objective-centred solution, (3) design and development-centred initiation, and (4) client/context initiation. A brief description of the activities is reported by vom et al. (vom Brocke et al., 2020) and reproduced as follows:

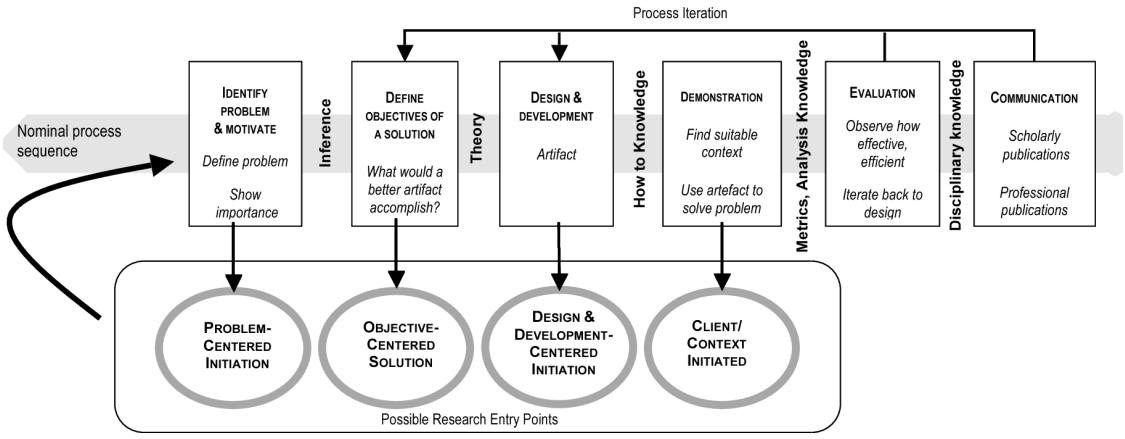


Figure 3-2 DSR Methodology Process Model (Adapted from Peffers et al. 2007)

Activity 1. Problem identification and motivation. This activity defines the specific research problem and justifies the value of a solution. Justifying the value of a solution accomplishes two things: it motivates the researcher and the research audience to pursue the solution and helps the audience appreciate the researcher's understanding of the problem. Resources required for this activity include knowledge of the state of the problem and the importance of its solution.

Activity 2. Define the objectives for a solution. The objectives of a solution can be inferred from the problem definition and knowledge of what is possible and feasible. The objectives can be quantitative, e.g., terms in which a desirable solution would be better than current ones, or qualitative, e.g., a description of how a new artefact is

expected to support solutions to problems not hitherto addressed. The objectives should be inferred rationally from the problem specification.

Activity 3. Design and development. An artefact is created. Conceptually, a DSR artefact can be any designed object in which a research contribution is embedded in the design. This activity includes determining the artefact's desired functionality and its architecture and then creating the actual artefact.

Activity 4. Demonstration. This activity demonstrates the use of the artefact to solve one or more instances of the problem. This could involve its use in experimentation, simulation, case study, proof, or other appropriate activities.

Activity 5. Evaluation. The evaluation measures how well the artefact supports a solution to the problem. This activity involves comparing the objectives of a solution to actual observed results from the use of the artefact in context. Depending on the nature of the problem venue and the artefact, evaluation could take many forms. At the end of this activity, the researchers can decide whether to iterate back to step three to improve the artefact's effectiveness or to continue communication and leave further improvement to subsequent projects.

Activity 6. Communication. Here all aspects of the problem and the designed artefact are communicated to the relevant stakeholders. Appropriate forms of communication are employed depending upon the research goals and the audience, such as practising professionals.

3.4 Design Science Research Methodology for DICE

Presented in the preceding sections, DSR output is in the form of the development of software innovations - artefact. Artefacts can take several forms, such as models, constructs, frameworks, architectures, methods, design principles, and instantiations. March and Smith (1995) published the classification of DSR output as a possible outcome of a research and are described below:

Constructs are the conceptual vocabulary of a problem/solution domain. Constructs arise during the conceptualisation of the problem and are refined throughout the design science research cycle. Since a functional design (artefact) consists of a large number of entities and their relationships, the construct set for a design science research experiment may be larger than the equivalent set for a descriptive (empirical) experiment.

A *model* is "a set of propositions or statements expressing relationships among constructs." The authors identify models with problem and solution statements. They

are proposals for how things should be and presented in terms of what it does, and a theory described in terms of construct relationships.

A *method* is a set of steps (an algorithm or guideline) used to perform a task. "Methods are goal-directed plans for manipulating constructs so that the solution statement model is realised". The problem and solution statement expressed in the construct vocabulary is implicit in a design science research method.

An *instantiation* is the operationalisation of constructs, models, and methods. It is the realisation of the artefact in an environment.

Method, instantiation, construct, and model are found in this thesis; therefore, they fit the research methodology for DICE. The final output of this research work is in the form of a method of deploying distributed simulation on cloud infrastructure. Moreover, other DSR outputs are presented, such as instantiation and constructs produced while developing the final artefact - DICE.

Method: Established above, a method is a set of steps (an algorithm or guideline) used to perform a task. Deploying distributed simulation on the cloud requires the analyst to follow some guidelines on designing, developing, deploying, and executing experimentation. There is none in existence, and this work aims to design and propose two artefacts; (1) Cloud-based distributed simulation development framework, which will be a step-by-step process to develop cloud-compatible federate for execution in a geographically distributed federation running over WAN or the Internet. (2) The deployment architecture to the server as a template on how to organise the computing resources in order to run CBDS successfully. The first research question (RQ1) enquires about "How do we deploy distributed simulation on the cloud?" This method is sufficient enough to address this question.

Construct: Constructs arise during the conceptualisation of the problem and are refined throughout the design science research cycle. This will be used during the model conceptualisation and facilitates the relationships between entities on the deployment architecture and the cloud-based federate development framework. The second research question (RQ2) can be answered using the relationship between components used to define interoperability and issues during experimentation on the cloud.

Model: The proposed deployment architecture is a composition of artefacts that were modelled to solve a problem. This model will define the relationship between the constructs and the method to be applied to solve the problem of interest.

Instantiation: The last research question (RQ3) seeks to understand the factors affecting cloud-based distributed simulation experimentation speed. The two proposals, the CBDS development framework and the deployment architecture, is set to undergo a feasibility test. A suitable case study prototype will run experimentation, collect some results, and analyse for different measurements. This is the ultimate goal of a DSR instantiation output.

3.5 Justification for Choosing Design Science Research

This study aimed to design and propose a development framework, Distributed Simulation Cloud Architecture for Experimentation (DICE) and examine their feasibility and performance; hence, the DSR approach was the most appropriate choice. In the Design Research paradigm, knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artefact; hence it can be called exploring by building. It is inherently a problem-solving process (Vaishnavi and Kuechler, 2015) .

Table 3-1 Design-Science Research Guidelines (Adapted from Sudha et al., 2004)

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

The learning through the building (creation) of an artefact is what defines DSR. Researchers reveal that the DSR cycle maps well with the three different research types -

Design and Development, Exploratory, and Reflective Evaluation, as well as the seven guidelines in Table 3-1 for constructing an artefact which was suggested by Hevner *et al.* (2004). DSR appears to be the most suitable research methodology that can be used to address the research types presented above. In addition to the artefacts, design research offers two more research contributions; (1) Reproducible Knowledge - a novel artefact consisting of the different DSR outputs used to improve the existing knowledge base further. (2) Methodologies and Theories present ways to support the phenomena of interest-based on the development and use of the novel artefact (Purao, 2002). DSR is the implementation of artefacts that could well be used to improve theories and serve as a significant research contribution.

We have established the DSR methodology, which is employed to address all the different types of research types and the research questions posed in chapter one. DSRM is sufficiently supported by the seven guidelines by Von Hevner *et al.* (2004). The idea is to be used for a successful building and evaluation of artefacts, in this case, development framework and DICE, and therefore, provides a strong rationale for its adoption.

Other Methods

In contrast, there are other research methods; quantitative, qualitative, and mixed mode, which combine both the qualitative and quantitative strengths to achieve the given research objectives. In this work, quantitative research is best suited based on philosophical assumptions and deductive research approach. This choice is well defended by Mujis (2011) qualitative research which argues that there is no pre-existing reality while quantitative assumes that there exists a reality about conditions that cannot be influenced by researchers in any way. Furthermore, qualitative research is often used when there is little to no knowledge of a phenomenon. Quantitative research is employed to find the cause and effect of the relationship between variables to either nullify or verify some hypothesis or theory (Creswell, 2002; Yvonne Feilzer, 2010).

Therefore, in this section, the researcher observes from the available literature in chapter two and decides on research methodology. The research community has not represented a cloud architecture for distributed simulation. To achieve the aim of this work, there is a need to produce one. Again, to design and propose the new cloud-based architecture, certain aspects, must be considered, including cloud infrastructure, distributed simulation, and development methodology. Looking at these terms in research leads to architecture development.

Based on the methodological perspective of the research method using a framework, the prominent influencers are already reported in chapter two in detail. According to the established research guidelines and DS domain-specific principles, this study is conducted, and generalisation is not the primary aim. The principal purpose of this work is to design and experiment DICE feasibility. While this can be applied to other related DS-related fields, broader and in-depth work needs to be carried out with success to ascertain its adoption.

3.6 Cloud-Based Simulation Architecture Development Methodology

In the previous sections, DS challenges were identified, and cloud computing potentials were also presented. This work proposes a methodological framework to ease the design, development, and deployment of CBDS for large-scale simulation projects. The CBDS deployment architecture - the DICE is a novel proposal that aims to guide the analysts on how to deploy DS on cloud infrastructure. Experiments in M&S usually begin with "what", a question that the analyst poses, simulates, and analyses results to find answers. As in other study domains, M&S modellers should carefully consider the methodological framework for their system study. In this case, it should serve as a guidance and approach for structuring how CBDS should be performed.

There are a few methodologies for developing a framework (Johnson, 1997) such as one reported by Nance (1987), King *et al.* (2017), Mustapha *et al.* (2010), Santa-Eulaila *et al.* (2011), Dai *et al.* (2014), and one developed by Anastasia and Taylor (2017).

Most of these methods start with a bottom-up approach to identify abstractions. This begins by examining existing solutions to get the basic concepts and tradition, hence the review previously proposed and published solutions including those reported above; DSEEP (IEEE, 2011), Guan, Grande and Boukerche (2019) and Taylor (2018). Figure 4-1 in chapter 4.2 shows the extended DSEEP framework upgraded with cloud implementation components.

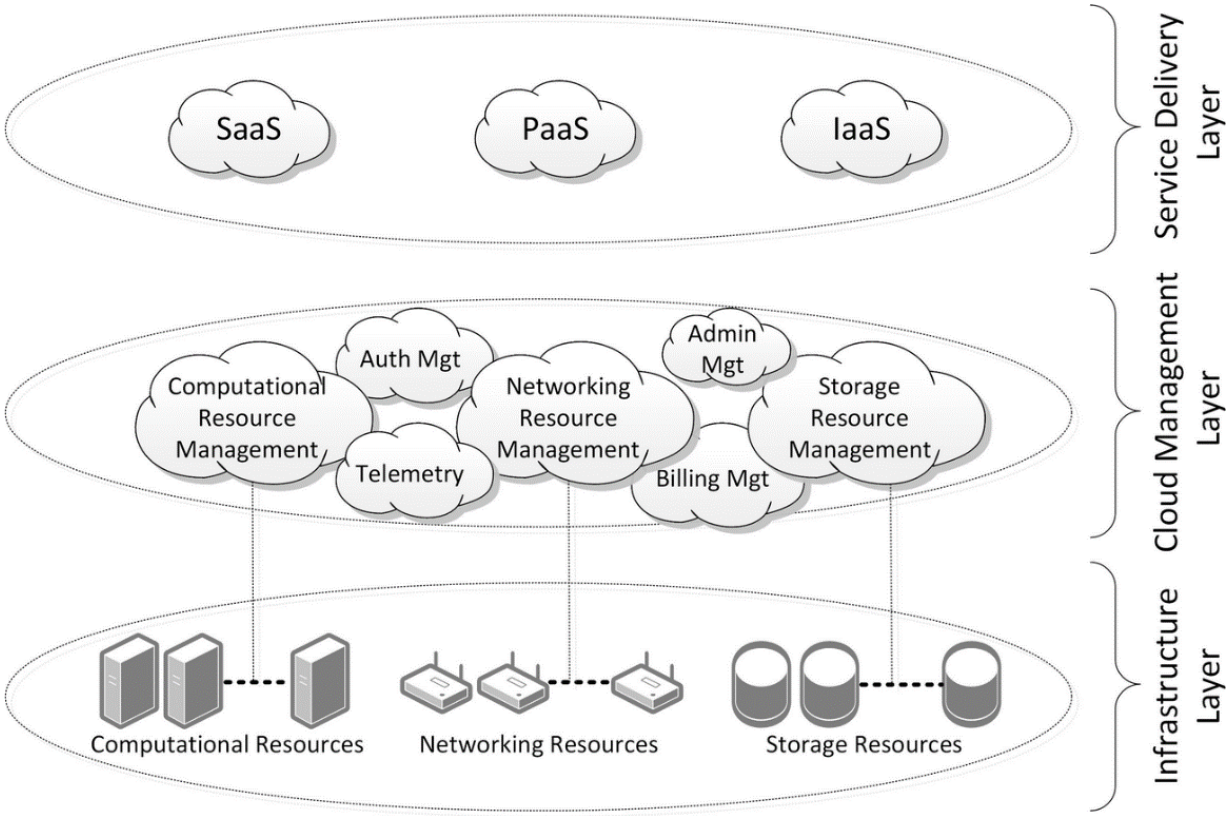


Figure 3-3 Classical Cloud Layered Architecture (Adapted from Dong et al., 2018)

Development Process

The development process involves four phases of inter-related activities shown in Figure 3-4. The phases are analysis, development, prototyping, testing and evaluation. This multiphase process is a continuous circle of developments, adjustments, and refinements.

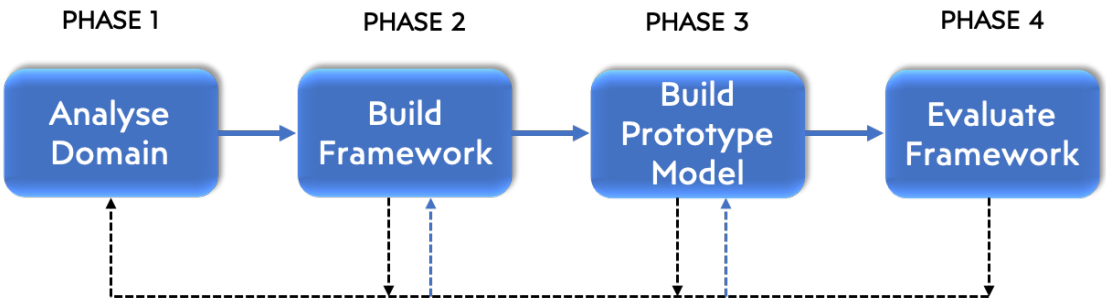


Figure 3-4 Iterative design process used in developing the CBDS Framework

Phase One - analyses the DS and Cloud-Based M&S domains and how DS applications and models are built.

Phase Two - is the actual framework development.

Phase Three - build DS prototype model using the new framework guidelines.

Phase Four - the framework is tested, evaluated and areas of further work are identified and presented.

Both the cloud-compatible federate development framework and the cloud deployment architecture for CBDS follows the same development phases; the testing and evaluation will be designed accordingly. Details of the framework (Figure 4-1) and deployment architecture (Figure 4-3), components, and how they work are given in the next chapter.

3.7 Simulation Model Design in Research

The field of simulation and model design has grown to incorporate existing research methods. Figure 3-5 shows a contribution regarding how generic simulation projects exhibit particular system imitation, system context, and the problem identified to be solved by the simulation (Chan et al., 2015). The authors also argue that the system does not have to be existing or physical; it can be an idea, concept, or proposal. The one mandatory requirement needed to design a simulation project experiment is the "behaviour over time".

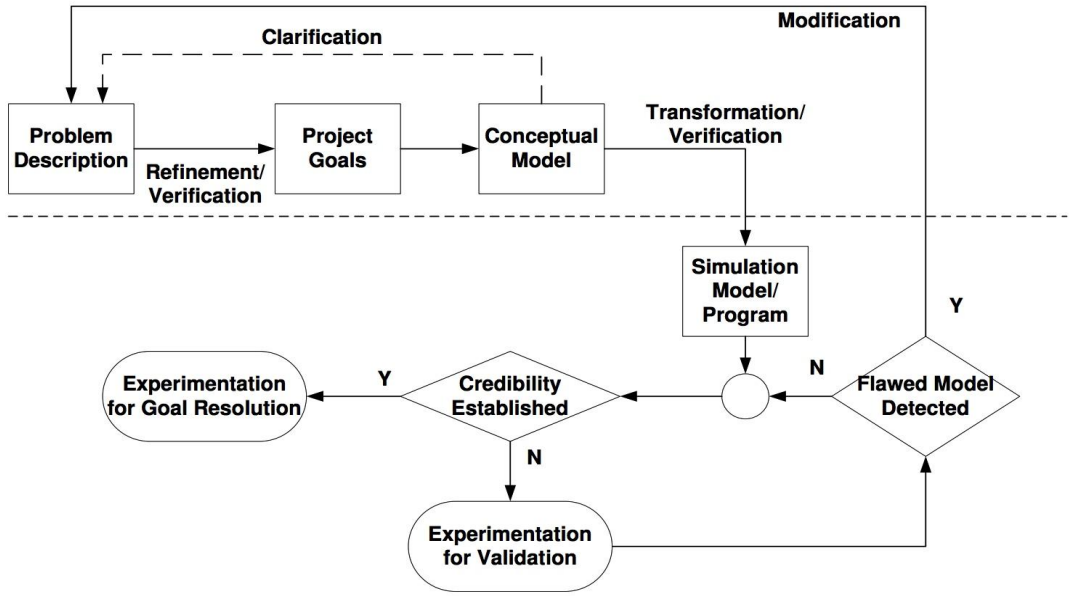


Figure 3-5 States of a modelling and simulation study (Adapted from Chan et al., 2015)

Here, the modelling process starts with the *Problem Description* stage; a crucial task of the modelling & simulation analyst is developing a problem description document. This primary document evolves from the embellishment of the necessary information received from

the user. This depends on the project requirements and team, in this case, using qualitative and quantitative methods of data collection for the SUI.

Finally, *Project Goals* stage the last in the study circle is regarded as the formulation of goals for the simulation project, which turns out to be the first step in the refinement process that will transform the problem description into a conceptual model. It is also believed that achieving some set of clearly defined project goals will coincide with the problem solution. Goals are classically stated in terms of policy options (including details of the experimentation during which these are manipulated) or parameters, and output variables observed during experimentation (Chan et al., 2015).

3.8 Case Study Method

Zainal (2007) states that a case study method enables researchers to examine the data within a specific context carefully. In most cases, a case study method selects a small geographical area or a minimal number of individuals as study subjects. In their true essence, case studies explore and investigate contemporary real-life phenomena through detailed contextual analysis of a limited number of events or conditions and their relationships. From another perspective, a case study is seen "as an empirical inquiry that investigates a contemporary phenomenon within its real-life context; when the boundaries between phenomenon and context are not evident; and in which multiple sources of evidence are used" (Yin, 2014).

In defining a research approach, the first step is choosing a suitable methodology. A decision has been made to use a deductive approach for this work. The fundamental processes involved are illustrated in Figure 3-6. To validate this research with empirical, an Emergency Medical System (EMS) model is used as a case study research method, which allows an in-depth analysis of the identified research problem. Furthermore, Case study provides a means to test whether a proposed theory applies to real-world phenomena. Processes involved in case study methods are shown in Figure 3-6 according to (Yin and Campbell, 2018).

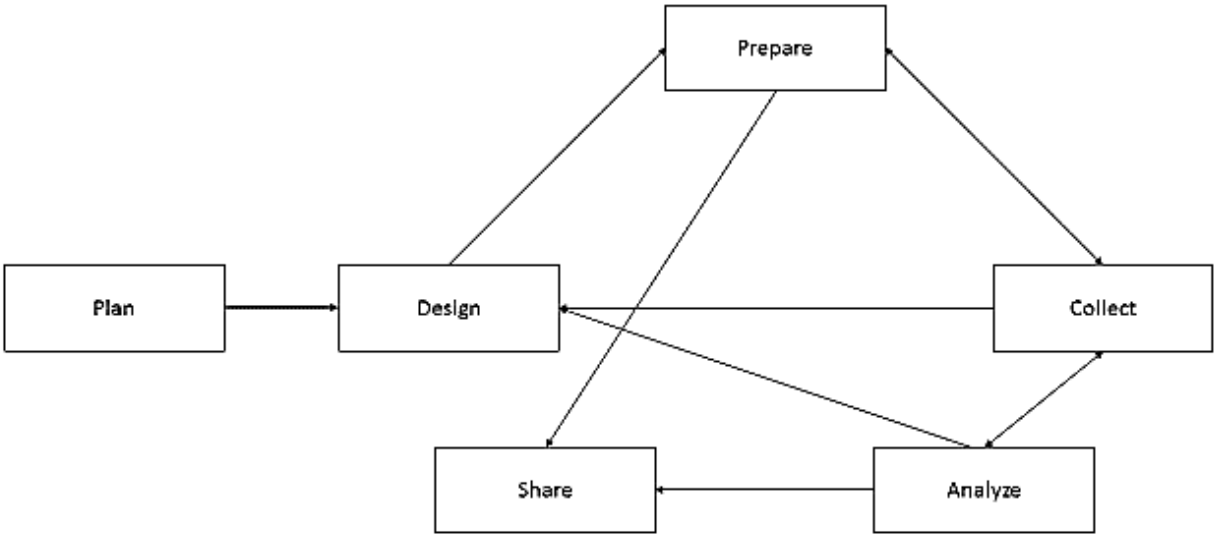


Figure 3-6 Phases in Case Study Research (Adapted from Yiun and Campbell, 2018)

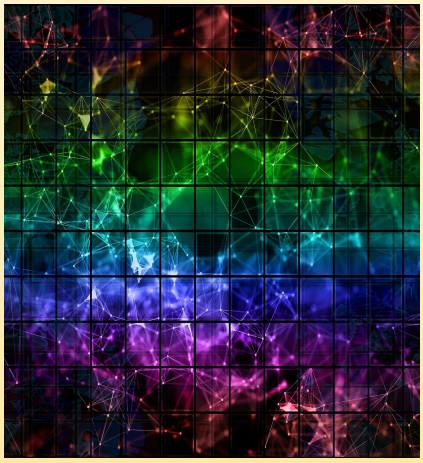
In his book, Case Study Research: Design and Methods, Yin also gave three categories of a case study research approach. Thus, exploratory research explores any phenomenon in the data that serves as a point of interest to the researcher—descriptive, set to describe the natural phenomena that occur within the data in question. Finally, descriptive case studies examine the data strictly both at a surface and profound level to explain the data's phenomena.

3.9 Chapter Recap

This chapter reported some research approaches in M&S. The chosen method is presented - the Design Science Research (DSR) and backed by the published literature in the domain. The method presented an overview and DSR framework. The chapter gave a detailed process for typical DSR activities. The section also shows how the chosen research method designs and produces the development framework and CBDS deployment architecture. The chapter is the second step, after a literature review on how to address the first research question - the RQ1. A justification of presented as the rationale behind choosing DSR for this work.

The section prepared ground for EMS; the case study prototype used to test the proposed architecture's feasibility – the DICE. The data collection method is also presented alongside the research design used to answer posed questions in chapter one. Next, chapter four will discuss the DICE development process, architecture, and technical requirements needed for implementation and testing in chapter five.

This page is intentionally left blank for printing purpose.



CHAPTER FOUR

PROPOSED ARCHITECTURE DEVELOPMENT - DICE

Chapter 4 Proposed Architecture Development - DICE

4.1 Chapter Overview

The preceding section reports the research methodology employed, rationale, justification of choice, discussed the cloud architecture development approach, model design in research, and case study method used. It also analyses the current architecture as practised in cloud-based simulation, which guides the development of the proposed solution to allow easy understanding and adoption by practitioners. This chapter presents the proposed architecture and framework development processes and explains possible implementation schemes used in the following chapter to test, analyse results, evaluate, and validate the proposed architecture's feasibility.

4.2 The Distributed Simulation Cloud Architecture for Experimentation (DICE)

As noted in chapter two, the cloud-based DS aims to link simulation models to form a larger model. There are many attempts to design, develop, and propose cloud solution for simulation modelling and distributed approach such as Falcone *et al.* (2017), D'Angelo and Morzolla (2014), Medel *et al.* (2017), Chaundry *et al.* (2016), and Riley *et al.* (2004).

In the works mentioned above, the cloud-based techniques try to improve the elapsed time simulations for large ABS and DES models by distributing the application to connect and communicate with concurrent Logical Processes (LP) (Taylor, 2018). Some simulation platform's KPIs are usually evaluated based on the execution time and the required resources to complete the simulation run. Additionally, cloud computing platforms with a pay-per-use or pay-as-you-go model have costs attached to resources for a successful run.

The proposed CBDS architecture – DICE also inherits the design principles of cloud computing's "layered cake" in Figure 3-4 in chapter 3.6, as presented in (Dong *et al.*, 2018). The author explains each of the three layers; the service delivery layer, which is visible and used by the users. The middle layer houses the cloud management facility and links the infrastructure, and the service delivery layers. Finally, the bottom is the infrastructure layer that holds other computing resources such as storage, networking, software, and hardware.

Using the method earlier described, the following framework is due to a combination of three methods: the framework development process, Distributed Simulation Engineering and Execution Process (DSEEP), and the simulation DS methodology. The IEEE (2011) published

the DSEEP recommended practice (IEEE Std 1730-2010) which defines the processes and procedures that should be followed by users of distributed simulations to develop and execute their simulations; it is intended as a higher-level framework into which low-level management and systems engineering practices native to user organizations can be integrated and tailored for specific uses. Moreover, the scope in this work is expanded to include the cloud-based aspects of DS experimentations.

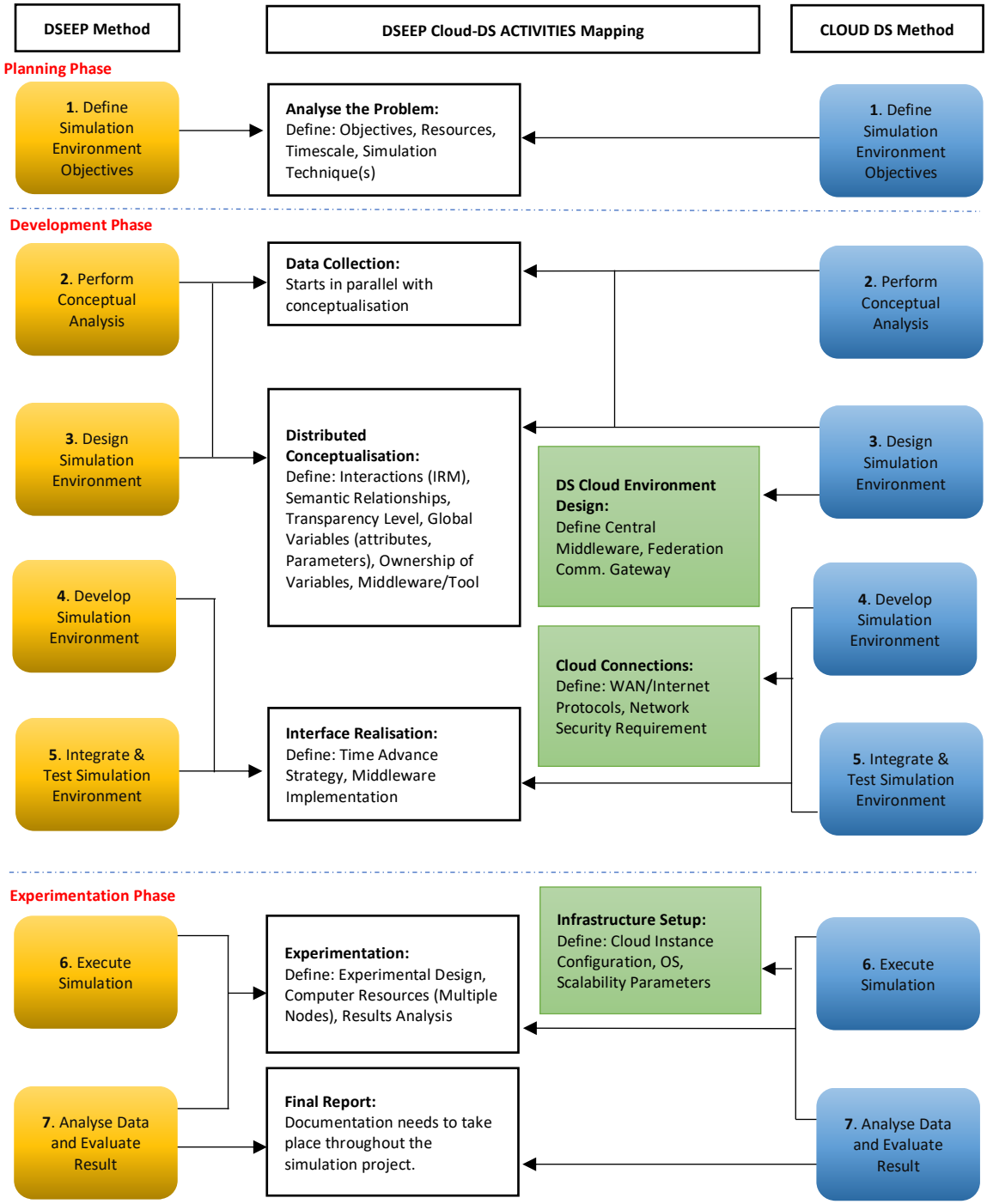


Figure 4-1 Proposed Cloud-Based DS Methodological Framework

The developers of IEEE Std 1730-2010 allow flexibility during implementation. This gives room for the extension of its capability and improvements to adapt to state-of-the-art demand. Figure 4-1 is the proposed methodological framework designed to design and run DS using a cloud environment and its associated technical requirements. Steps three, four, five, and six are the affected sections to accommodate the cloud infrastructure requirements, which will be explained more shortly. In the subsections that follow, the activity description for each framework step is given and broken down into three broader phases – planning, development, and experimentation phases.

4.2.1 Planning Phase

Simulation Project Planning

Step one is where the problem to be analysed is defined. This is the appropriate time to decide with project sponsors whether the simulation is an appropriate analysis method for the system under investigation (SUI).

Furthermore, the analyst and all stakeholders ought to define the project objectives, performance measurement criteria (KPIs), project goal, the needed resources to perform experiments, technology requirements - in the case of DS the environment suitable for the SUI, project timeline, and other requirements.

4.2.2 Development Phase

Perform Distributed Conceptualisation

In the distributed conceptualisation step, the conceptual model of the whole system should be delivered. This involves the component models (federates) and their interactions. At this stage, it is not necessary to include the details of each model. The critical elements are the subsystems that each model will represent and the communication among them. The individual models, therefore, can be represented as black boxes.

The outputs of this step are the design of the high-level components of the SUI. It shows the models representing each part of the system to analyse and the data exchange between the subsystem units. Also, at this step, the type of interaction IRM standards (Taylor *et al.*, 2012) should be defined if the system calls for a hybrid simulation approach. For example, the use of both ABS and DES in one investigation.

Due to the complicated nature of the hybrid DS, the semantic relationship between the combined paradigms must be explicitly identified to avoid conflicting outputs.

For cloud-based DS, there may be a scenario where the data exchange between federates needs to be centralised or otherwise, hence the need to identify the central hub connecting the federation. A middleware implementation tool that supports distributed simulation over a WAN environment and gateway settings for communication between geographically dispersed federate need to be defined at this step.

Build Models (federates)

At this level, if some or all of the models already exist in DS form, these can be recycled. Reusing the model helps avoid duplication of initial effort.

Model Conceptualisation - The modeller at this stage should use IRMs. IRMs define the data accessibility within the federation, between federates and external entities that need to interact with the simulation project via the RTI. The variables are also to be defined, such as global, private, shared, and ownership.

Data Collection - DS projects often require facts and data describing the SUI, which guides the researcher to the successful analysis of the problem to be investigated. This activity is carried out simultaneously with the distributed conceptualisation step, which is inherently informed by the gathered data.

Model Realisation - Realisation involves turning the concept into a computer-readable form using suitable simulation language and tool, as discussed in chapter 2. The by-product, which is the code produced would undergo a validation process.

Define Time Advance Strategy

Here, the project designer will decide which time management is suitable for the SUI. The two main categories are the Time Advanced Request (TAR) used by time-stepped federates and Next Event Request (NER) used in an event-driven federates, which both are applicable depending on the simulation technique. It is used for various federates. There are two synchronisation protocols for federated communications – the optimistic and conservative approach. The former has a recovery mechanism, and the latter does not allow recovery in the event of failure.

Middleware Implementation

In HLA, the central backbone of simulation execution is the RTI. A few of the required components in the RTI are listed below. The middleware is subject to validation and verification after all models and RTI building is complete. The RTI uses components such as FOM, SOM LRC, CRC to coordinate the federation. *The Federation Object Model (FOM)* must be present in all federates and it defines how the interaction takes place in the federation based on objects attributes and transparency level. *The Simulation Object Model (SOM)* defines what and how they can exchange information with other federates in the federation. The individual federate uses the SOM to publish what it wishes others to subscribe to and vice-versa. *Local RTI Components (LRC)* are libraries used by the federates in DS experiments. *Centralised RTI Components (CRC),* an executable or a sort of gateway that interacts with the LRC for the federation execution. *The federate ambassador* is a class which the RTI uses to relay information to all federate *call-back* method. *The RTI Ambassador* is the point through which the program uses API to invoke services using a call to an instance of RTI ambassador connecting the federation.

To run HLA-based DS in the cloud, a ***CBDS Middleware*** with web-enable APIs is required. This is where users define the main RTI responsible for starting and maintaining the federation execution over the cloud environment. All traffic will be routed to this central RTI, and, in turn, it delivers messages to destination federates/federation using pre-configured IP addresses.

CBDS Configuration

Cloud providers usually give infrastructure and allow users to configure the hardware (hypervisor) and software elements of a cloud environment. This ensures that they communicate and inter-operate effectively with various internal and external client services.

Communication Gateway

When there are federates in one region or cloud instance that communicate locally using a multi-cast mode, one should be dedicated to acting as the gateway to the larger (distributed) federation. The federate assigned as the gateway will receive traffic from a distant federate and distribute it to "local" federates and vice-versa. The gateway configuration can be a file loaded while starting the federation's main RTI.

CBDS Connection

CBDS runs on distributed cloud computing resources. Federates can reside in one cloud provider or different providers located in the same region or distant. Connecting these in a WAN mode requires IP addresses that identify each VM or node in the federation. These addresses can be configured as dynamic or permanent, based on what the provider supports. The middleware package will be notified with the various VM addresses to allow them to join the federation and receive and send traffic during simulation execution.

Users also decide and define *the privacy* of data within and outside the cloud environment, such as the public Internet or Virtual Private Networking (VPN). The *security* protocols should also be defined to transport data securely during DS execution over the World Wide Web (WWW) such as Secured Security Layer (SSL), Transport Security Layer (TLS), or other forms of network data security mechanism.

4.2.3 Experimentation Phase

Experimental Design

As in conventional simulation projects, CBDS analysts determine the experimental setup parameters such as the simulation scenarios, input data, amount of runs and the associated random seeds needed for the desired output for analysis. Other simulation experiment concerns include how long the experiment should run, input parameters, initialisation conditions, warm-up periods, length, and resources.

DS Cloud Infrastructure Setup

Some cloud service providers allow users to subscribe and define scalability triggers. They may configure to scale, up or down depending on the executing workload or simulation experiment data. They can define cloud-based VM recourses such as CPU, storage, memory, operating system image, etc. More models join or leave the federation.

Result and Analysis

In many simulation projects, the final report includes a result presentation using statistical and graphical KPI's analysis, sensitivity analysis, and recommended solutions. DS allows the analyst to have insights into how subsystem performance affects the whole system.

4.3 DICE Deployment Architecture

The cloud computing concept gives users on-demand network access to a shared pool of configurable hardware and software resources (Mell and Grance, 2011). Therefore, DICE architecture is designed to fit into this principle. The cloud layered model separates hardware software and other services from which the user chooses during configuration. This is backed by existing solutions published by authors who adopt the existing layering to deploy simulation and DS applications for experimentation. Some examples are presented in chapter two and others like (Calheiros *et al.*, 2011; Nuñez *et al.*, 2011; Liu, Qiu, *et al.*, 2012; Núñez *et al.*, 2012; Rossetti and Chen, 2012; Islam, Shaikh and Sheikh, 2016; Wang and Wainer, 2016; Kousalya, Balakrishnan and Raj, 2017; Salama, Elkhatib and Blair, 2019). The basic requirements to run distributed simulation on the cloud are computing resources, client infrastructures such as FTP, and user interface. This research aims to ease the development of CBDS by non-technical modellers and having fewer layers will make project design and execution relatively less complicated. The more layers, the more expertise, and experience are required to run DS experimentation in the cloud.

Requirements for the new Architecture

Before developing the architecture, let us look at the fundamentals and what is required to design and propose one for this thesis's purpose. Generally, this research recognised several options for deployment, and this work focuses on two prominent example schemes that are explained a little later in this section. To run the identified use cases that align with these example deployments, architecture needs to possess some elements as presented in Figure 4-2 by (Grobauer, Walloschek and Stöcker, 2011).

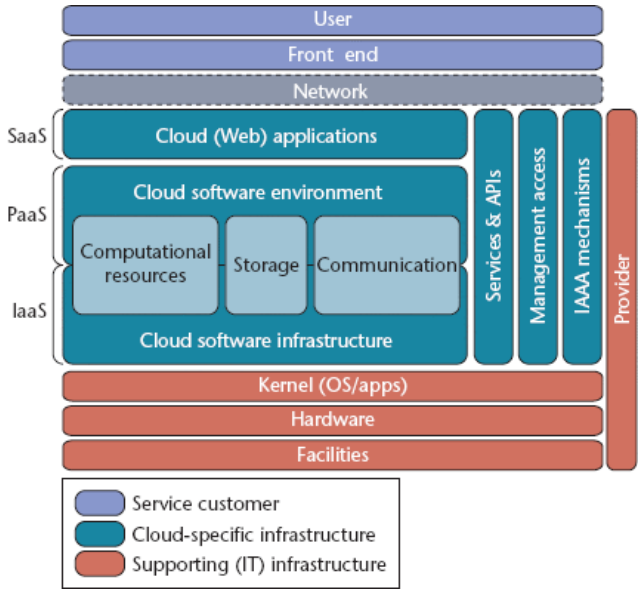


Figure 4-2 Cloud Reference Architecture (Adapted from Grobauer, Walloschek, and Stöcker 2011)

Cloud computing architectures are divided into two parts - front-end and back-end (Jadeja and Modi, 2012) which are connected via a network such as a wide area network or the Internet in many cases. The front-end is where the user application is found and used to access the cloud services - the SaaS model. The backend is the actual on-demand computing resources offered by the provider the IaaS with a pool of storage, servers, and networking, for example. Due to evolution taking place in business organisations in the 1990s, the internet-based ecosystem moved to a horizontal structure from previously vertical, which was more challenging to test, use, and maintain (Seda *et al.*, 2019). Moreover, the need to continuously deliver services leads to migration from monolithic to microservices and service-oriented architecture (Endrei *et al.*, 2004).

Layered architectures are commonly used in designing hardware and software systems. The layering provides flexibility to make changes in one layer without affecting the rest. This makes the idea approachable where the system offers a different level of services and functionalities. Some examples of common layers found in this type of architecture stack are presented in a book (Sheriff, 2006), which comprises the data access layer, business logic layer, web services layer, and user interface layer. Some benefits of this separation are that it enables designers to distribute functions, allows independent layer implementation, and quickly replaces it with a different approach. Furthermore, layered system architectures can be implemented in one-tier, two-tier, three, or more tier depending on the needs, and each has its good and downside trade-offs. Therefore, to develop a layered architecture for cloud-based systems from the above fundamentals, this work considered four requirements and split into sections:

- The resource layer: The layer is where the user configures computing hardware and software resources at the IaaS level. The cloud computing model usually charges users based on the selected resource and "rent" period.
- The cloud access layer: This is an option for the user to choose between public or private cloud setup. Furthermore, these analysts can choose different cloud service providers based on cost, availability, datacentre location, etc.
- The management layer: The is part of the core function where the user can manage the application running on the cloud platform.
- The client access layer: User or client applications are the interfaces that enable the user to access the network configurable computing resources in the cloud.

Other requirements are explained in the DSEEP development framework in chapter two. It is essential to mention that building architecture for cloud ought to consider some more factors such as cost of use and pricing model, general speed factors, cloud platform portability which is the ability to move to federate from one cloud platform to another with fewer complications, and data security on the simulation traffic traversing WAN/Internet environment. All these may be important but are not the focus of this thesis. The idea here is to prove how the principle works, and overall keep the layer stack as simple as possible from the user point of view - in this case, the analyst who are not profoundly technical experts. The proposed architecture is illustrated in Figures 4-3, and versions of services offered are explained concerning this thesis.

These layers above are arranged and designed and brought together to create the layered architecture to bring functions and services that are found missing in chapter two, Table 2-1. Moreover, research papers and books such as (Zhang, Wang and Li, 2019) on the cloud, DS, frameworks, and deployment architecture were examined. This is to ensure that the proposed architecture addresses the current and probably potential future CBDS challenges. Following the established cloud architectural styles, the resulting architecture in Figure 4-3 is composed of four layers: application, distributed simulation management, cloud platform, and VMs layers.

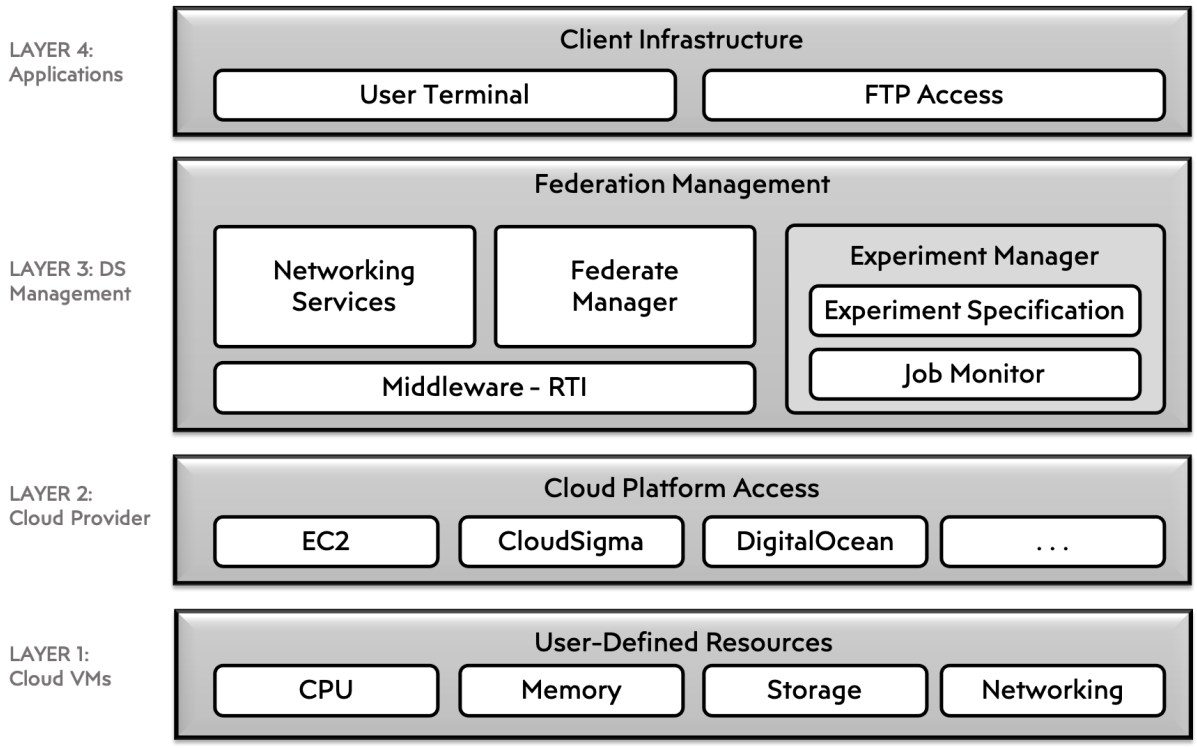


Figure 4-3 DICE deployment architecture with four layers

4.3.1 Layer 4: Application

Client Infrastructure

This is the front-end which contains the client interface applications required to access the cloud computing resources for DS experimentation setup and execution.

User Terminal - This is the application used by a thin or thick client to interact with the cloud services. This can be a GUI-based web browser or command line terminal. Different devices can be used for client access, such as tablets, PCs, phablets, intelligent terminals, etc.

FTP Access - DS experimentation preparation involves uploading a model and data files to the cloud storage and downloading the results output for further analysis. Depending on the environment design, the user uses a file transfer protocol (FTP) to achieve this activity.

4.3.2 Layer 3: DS Management

Federation Management

This layer accommodates the DS management components. It manages the federation setup, network and middleware, and overall experiment management.

Networking - This defines the network environment needed to execute the federation. The gateway is defined here, where all traffic will go through from source to destination federates during execution time.

Federate Manager - This is where individual models will be configured with necessary networking and synchronisation parameters to join the federation. Here, the *RTIambassador* services and *FederateAmbassador CallBack* are defined.

Middleware (RTI) - The Run-Time Infrastructure (RTI) provides services through information exchange standards between federates and synchronisation. The RTI also is used for the overall federation management in a DS project.

Experiment Manager - This subsection allows modellers to submit and launch an experiment through a parameter (federates IP address, cloud host, etc.) passing.

Experiment Specification can be defined and developed using suitable programming languages like Ansible's Playbook to automate the submission of simulation experiment jobs to the cloud.

Job Monitor can be integrated, such as Grafana to monitor experiment execution start and end timings, performance, and other desired KPIs.

4.3.3 Layer 2: Cloud Provider

Cloud Platform Access

Users can choose a cloud provider to use for the DS experiment depending on the simulation requirements defined in steps 4 and 5 of Figure 4-1 above. The different cloud service provider offers various services ranging from payment model and computing resources such a CPU, memory operating system, etc**.**

4.3.4 Layer 1: Cloud Instances (VMs)

User-Defined Resources

In this layer, the user defines the cloud virtual machine instance (VM) configuration needed for the DS experiment. Computing infrastructure resources are subscribed based on the experiment needs and are scalable in many public and private clouds. A basic VM setup may include CPU, Memory, Storage, Networking, and OS.

4.4 CBDS Experimentation Procedure with DICE

To run a cloud-based distributed simulation (CBDS), DICE architecture defines the resource layers that are combined to build a cloud infrastructure. In this architecture, resources are pooled and configured using virtualisation and shared across single or multiple cloud platforms via a network. The components of a CBDS architecture include the Client Infrastructure, Federation Management, Cloud Access, and user-defined resources. These are organised to prepare and run experimentation in three phases: preparation, execution and monitoring, results from analysis and reporting. Details of each phase are as follows.

4.4.1 Preparation Phase

Preparation of the experiment begins with defining, conceptualising, developing, and validating the model of the system under study. The model may be developed with any simulation technique such as ABS, DES, System Dynamics, or Hybrid (a combination of more than one technique). The analyst prepares the inputs scenarios for the experimentation. In this thesis, a hybrid Emergency Medical Service prototype model is

used. It consists of two techniques ABS (the ambulance federate) and DES (hospitals federates), which uses an Interoperability Reference Model (IRM) to interact with one another. More details were given in chapter four.

The analyst uses the top-most application layer to upload the model(s), input files and prepare the job launch script for execution. This comes after deciding on the cloud platforms, computing resources, and networking protocol for the CBDS project. The options available at this layer can be a web browser, FTP client, text-based command terminals, and mobile devices as facilitated by the edge computing concepts. Different cloud services present different characteristics in terms of user access, configure, and use the on-demand resources.

While preparing for the CBDS, the modellers should choose which cloud services provider fits the simulation design requirements. There are many public clouds platforms, each with different characteristics, and those reported by the literature may be preferable to use for the research experimentation and can be benchmarked during results and results analysis. Layer two of the proposed DICE deployment architecture enables analysts to choose their prepared cloud service or services in the case of multiple platforms. Clouds usually provide the user with an interface to configure the on-demand computing resources, such as the processors, memory, storage, and networking. This work uses five cloud services - Amazon Web Services (EC2), Google Cloud Platform, CloudSigma, DigitalOcean, and Scaleway. All of these infrastructures were used to run single and multiple experimentations on single or multiple clouds.

Layer three - the Distributed Simulation Management layer is the main back-end component of the DICE deployment architecture. This is where the middleware is configured for the federation to run on the cloud. The Run-time Infrastructure is defined using the FOM is and SOM. Federates communicates and exchange data during the federation execution; therefore, a network protocol is also defined and configured with necessary IP addresses and port numbers. This communication varies according to the cloud service provider's environment structure and configuration policies, which means the analyst must consider this factor when configuring the networking services for each cloud platform. For example, using poRTIco middleware in CloudSigma, federates may be hosted on different instances and communicate using a multicast network protocol. On CloudSigma, the analyst does not have to specify IP addresses to communicate. Whereas the case is very different with AWS as they do not support multicast at the time

of writing this report, which means the user must create a virtual private cloud (VPC) to configure and communicate between federates on different instances.

Moreover, this layer has an Experiment Manager section where the CBDS can be submitted using a script file written in Ansible. The file contains steps to execute the experimentation, including starting the federation's central component and federates, collecting and putting the outputs in the specified directory for the user to download and analyse. This Manager also monitors the execution and notifies the analyst when their federation crashes or is destroyed. It also logs the simulation starting and ending times for execution time performance metrics.

4.4.2 Execution and Monitoring Phase

After the detailed preparation phase, the actual execution starts. First, the participating clouds are initiated and assigned public IP addresses. Depending on the chosen cloud service providers in the preparation stage, the assigned IP address can be a dynamic address that is lost when the node shut down or restarted or static which is retained until the user releases it back to the pool. This IP is used to route messages exchanged between federates during federation execution and will be explained on one of the implementation approaches.

The launch script (experiment specification) is used under the Experiment Manager sub-layer to submit, start the DS, and collect results in layer three. The script contains parameters essential for the execution, including IP addresses (where applicable), cloud hosts, models, and input/output directories. The script also produces a real-time log showing the status while running, terminated, completed, destroyed, or a federate crashed and exited during the execution. This implementation only indicates the failure at the federation level. The user has to investigate the cause as it may be due to an instance being down or the network failing.

The analyst can integrate advanced job monitoring tools to measure various KPIs such as network traffic between federates, execution time, CPU performance, and storage use. CBDS can be executed on Windows or Linux operating systems, and each has several tools to capture run-time statistics for a different purpose. This research runs on the Ubuntu server version, and the focus is to measure performance and scalability; therefore, the script is used to capture starting and ending times for all experimentations reported in the next chapter. Grafana (https://github.com/grafana/grafana) is one example of an open-source monitoring and

analytics tool that gives the user a detailed interactive visualisation of various components of the data sources.

4.4.3 Preparation Phase

Upon successful experimentation run, the launch scripts detect, record the execution end-time, prepare the outputs to the specified directory, and shut down the nodes (optional). In the current implementation, if the user wants to run more experiments, they have to submit the jobs again as there is no job queue facility at the time of this report. The analyst uses the model outputs and the captured metrics for further analysis according to the CBDS simulation project-set goals.

4.5 DICE Deployment Sequence

The principle of the proposed architecture would allow flexibility during implementation on cloud environment (private/public/hybrid), connection, management, and flow control. With middleware support, this architecture can be used in a variety of ways. The design of the DICE service for CBDS is composed of four decoupled services layers: application (client infrastructure), DS Management (Federation Management), Cloud Provider (Cloud Access), and Cloud VMs (User-Defined Resources). The CBDS provisioning services are depicted by the deployment sequence diagram in Figure 4-4 and described below.

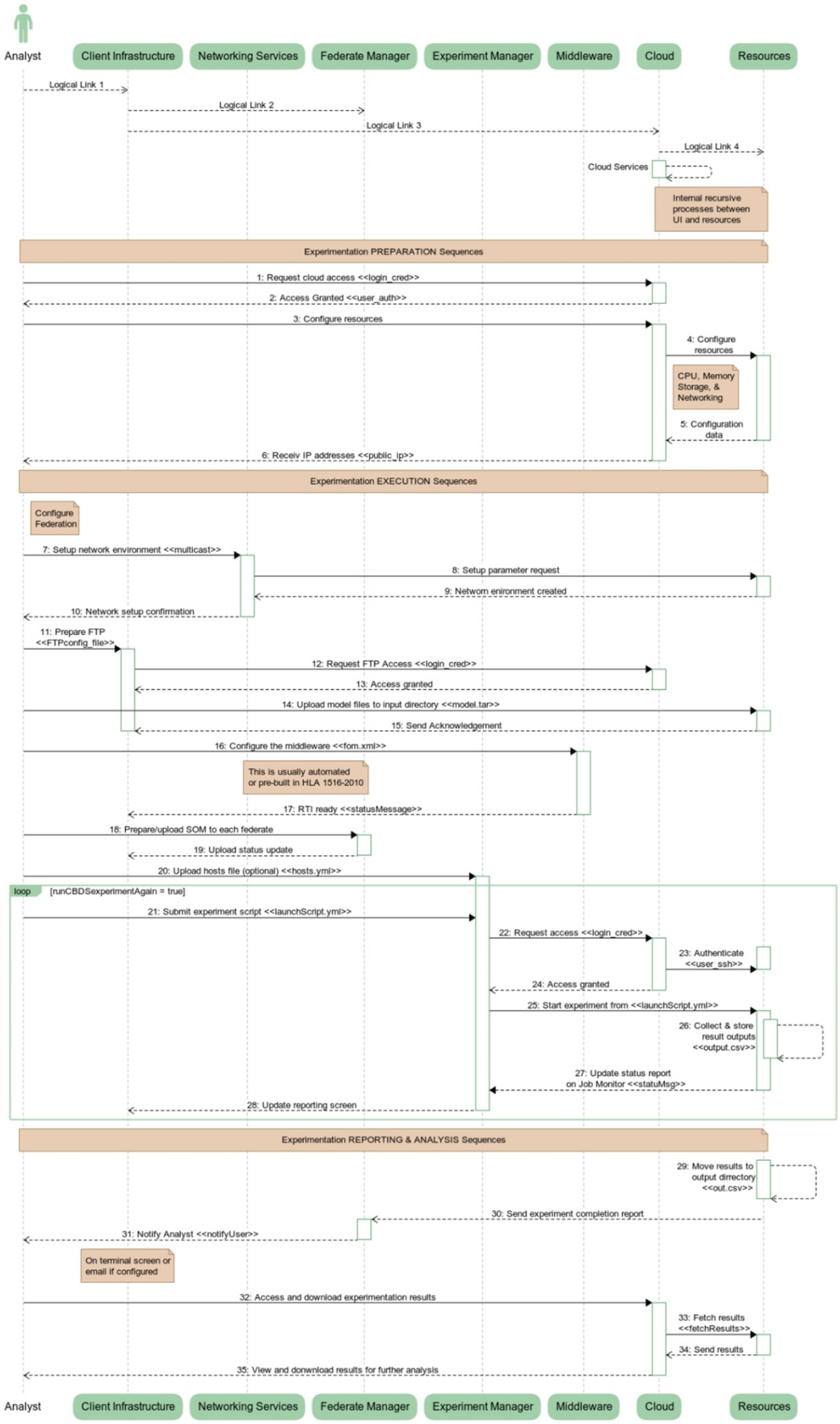


Figure 4-4 DICE Architecture Deployment Sequence - Single Cloud

In DICE architecture, cloud infrastructure is accessed on-demand. DICE takes advantage of Cloud Computing data and elastic resource provisioning to acquire the compute nodes from public or private cloud platforms. As presented in the sequence diagram above, the Client Infrastructure object provides a service interface by Analyst (actor) with Federate Manager and Cloud Platform objects. Computing resources are accessed and configured on-demand through the cloud provider's user interface, which can be a browser, FTP or command-line terminal. The sequence diagram is divided into three parts. The first section is the **Experiment Preparation** that is in charge of setting up the CBDS execution environment, which includes access credentials, configuring resources (CPU, memory, storage and networking), starting and stopping virtual machines throughout the experimentation period. In most cloud services, the resource-associated costs are determined by the computing resources selected and how long the CBDS takes to run.

Next is **Experimentation Execution** starting from sequence number 7: where the Analyst prepares the federation and federates for execution. FOM, OMT, and SOM are some of the configurations at this stage required for the RTI implementation. The Analyst uploads the models and input files (when necessary) and sends the launch script from the Federate Manager sublayer to start the CBDS experimentation. After executing the experiment for the given scenario, the infrastructure sends completion notification to the Analyst. Sequences 21: to 28: can be repetitive when the scenario calls for replications or reruns with different input files. The analyst must submit this manually after each run due to this implementation's lack of queueing facility.

Finally, **Reporting and Analysis** section, the CBDS finishes execution, and the Analyst receives the experiment report from the cloud services and the output results ready for download and further analysis. Depending on the cloud platform and service setup, the computing resources are released back to the pool at the end. As introduced in the previous sections, the computing resources refer to conventional compute services found in the market, such as CloudSigma, Amazon EC2, Google Cloud Platform (GCP), Scaleway, and those used in this research DigitalOcean. The proposed DICE architecture builds on top of these platforms to provide virtual machines and run experiments.

The above explains a series of sequences to prepare, execute and report CBDS experiments using DICE - all for a single platform where the middleware, federation, and federates are hosted on nodes within one cloud. One of the main contributions of this thesis is running CBDS across multiple clouds, and DICE deployment architecture is designed to enable analysts to connect multiple cloud platforms and run experimentation. Connecting

multiple clouds require the modeller to define a central WAN router where all the data traffic passes through from source to destination at runtime.

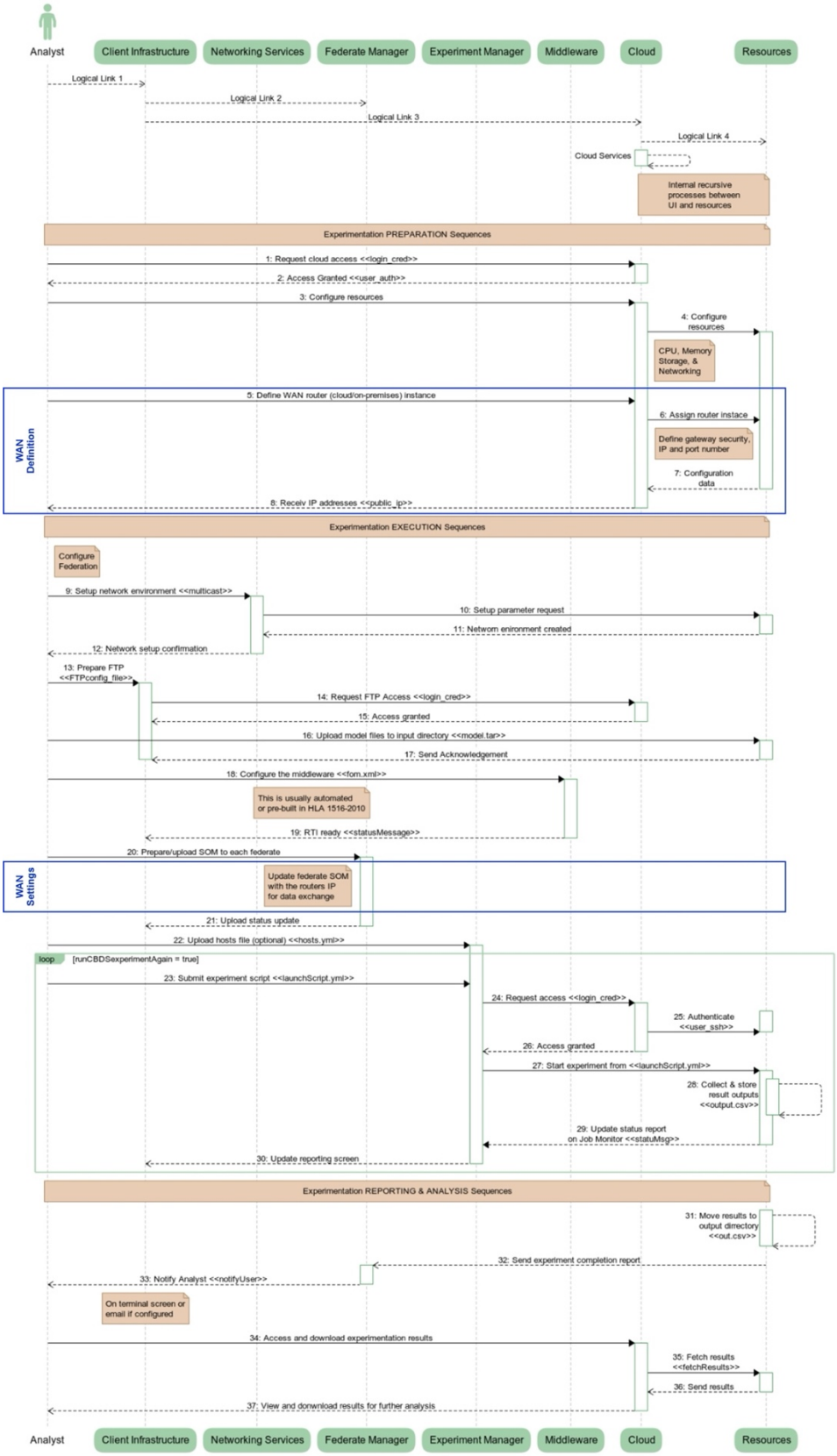


Figure 4-5 DICE Architecture Deployment Sequence - Multiple Clouds

Figure 4-5 is the modified sequence diagram showing the potion where the router configuration is introduced to connect and run multi-cloud CBDS. Specifically, sequence no. 5: 6: 7: and 8: are where the analyst defines the router and gets the selected instance specifications - public IP and port number. Moreover, the router can be on-cloud or off-cloud (on-premises), and in either case, the exact specification is required to connect multiple cloud services. The WAN router settings are updated when configuring the federation and all participating federates. During execution, federates from different cloud platforms interact and exchange simulation data; therefore, all federates require the main router address to forward the messages for relay to the intended destinations. However, some cloud environments may require analysts to define router and gateway even when running experimentations on a single cloud platform example is Amazon EC2 due to its flexible but complex networking services.

4.6 DICE Implementation Approaches

The Cloud computing concept brings about flexibility to users. DICE deployment architecture exploits this opportunity by offering at least six options to deploy CBDS experimentation. It is designed to allow an analyst to run single or multiple experiments on single or multiple cloud instances from single or multiple cloud service platforms (providers). Table 4-1 is a matrix of possible DICE's schemes - approaches to CBDS.

Table 4-1 DICE Deployment Matric - Possible Implementations Approaches

	Single Experiment	Multiple Experiments
Single Cloud	Scheme 1	Scheme 3
Multiple Clouds	Scheme 2a Scheme 2b	Scheme 4a Scheme 4b

Scheme 1: Single Cloud - Single Experiment allows running a single CBDS experiment on a single cloud infrastructure at a time. Example Amazon EC2.

Scheme 2a: Multiple Clouds - Single Experiment is where analysts run a single CBDS experiment on several cloud infrastructures from different providers. Example running one federation with federates hosted on Amazon EC2, CloudSigma, and DigitalOcean cloud

services. The traffic is traversing between clouds using a cloud-based WAN router (a cloud instance equipped with a gateway to relay traffic between the participating cloud platforms)

Scheme 2b: Multiple Clouds - Single Experiment is where analysts run a single CBDS experiment on several cloud infrastructures from different providers. Example running one federation with federates hosted on Amazon EC2, CloudSigma, and DigitalOcean cloud services. In this case, the traffic is routed between the participating cloud platforms via an offline router (a physical device on-premises that receives the incoming traffic and relay them to the destination cloud instance). This may be due to different design decisions by the analyst, such as security or close monitoring purposes. It also presents the opportunity to integrate IoT, Digital Twin devices for Industrial 4.0 connectivity in the CBDS execution circle.

Scheme 3: Single Cloud - Multiple Experiments. This approach enables the modeller to execute multiple federations in parallel on a single infrastructure from a single cloud provider such as CloudSigma.

Scheme 4a: Multiple Clouds - Multiple Experiments. The analyst can run multiple CBDS experiments on several cloud infrastructures from different providers in parallel. For instance, running multiple federations with federates spread across Amazon EC2, CloudSigma, and DigitalOcean cloud services. The traffic is routed between the connected clouds using a cloud-based WAN router as in Scheme 2a above.

Scheme 4b: Multiple Clouds - Multiple Experiments. In this approach, multiple CBDS experiments run on several cloud infrastructures from different cloud providers. For example, running multiple federations with federates hosted on Amazon EC2, CloudSigma, and DigitalOcean cloud services. Here, the traffic exchange between federates is routed between the participating cloud platforms using an offline router as presented in Scheme 2b.

Though there are six different ways to implement CBDS using the proposed DICE architecture, this thesis uses and reported only three approaches from the matrix - Schemes 1, 2a, and 4a. This is due to the cost associated with the cloud infrastructure and the time to complete various experimentation scenarios to measure the performance and scalability of CBDS deployed with DICE. These schemes are presented in detail below and the essential components used on Figures 4-2, 4-3, and 4-4 are;

- **Federate –** A term referring to a simulation model in an HLA-based distributed simulation project.

- **RTI –** Runtime infrastructure is a middleware. It is a fundamental component of HLA-based DS that coordinates federate their operation and data exchange.
- **Configuration File –** It is a file containing the gateway router's IP address and port number for data exchange during federation execution.
- **FEDambassador –** This is a class instance used by the RTI to deliver information to federates using callbacks.
- **RTIambassador –** This is the class through which the federate communicates with the RTI.
- **Router –** An instance is serving as the wide-area network router connecting the distributed federates when the cloud does not support multicasting. It can be implemented on the cloud or on-premises.
- **Arrows –** Indicates the flow of data in and out of components.

4.6.1 SCHEME 1: Single Cloud – Single Experiment

This scheme allows an analyst, depending on the runtime infrastructure, to connect federates directly to one another using a multicast protocol as illustrated by Figure 4-6. Again, various cloud platform service providers have various ways of dealing with network traffic. Users in some clouds such as Amazon EC2 have to use a specialised local gateway configuration file to be able to route traffic even on a single cloud. The configuration file should contain the datagram protocol, queuing and buffer flow control, IP address of the gateway federate, and the federation management rules.

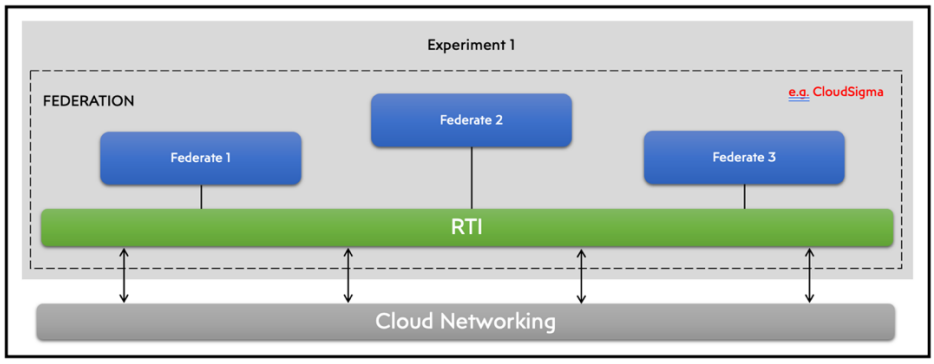


Figure 4-6 Single Cloud – Single Experiment Implementation

4.6.2 SCHEME 2a: Multiple Clouds – Single Experiment

The second approach is designed to centralise all traffic relay to a specific cloud instance via a "WAN router". The router receives incoming packets and forwards them to the intended federate. Figure 4-7 shows how multiple clouds can be used to run a single experiment using one geographically distributed federation. The federation

central component can be initiated by any cloud platform depending on the project design and execution.

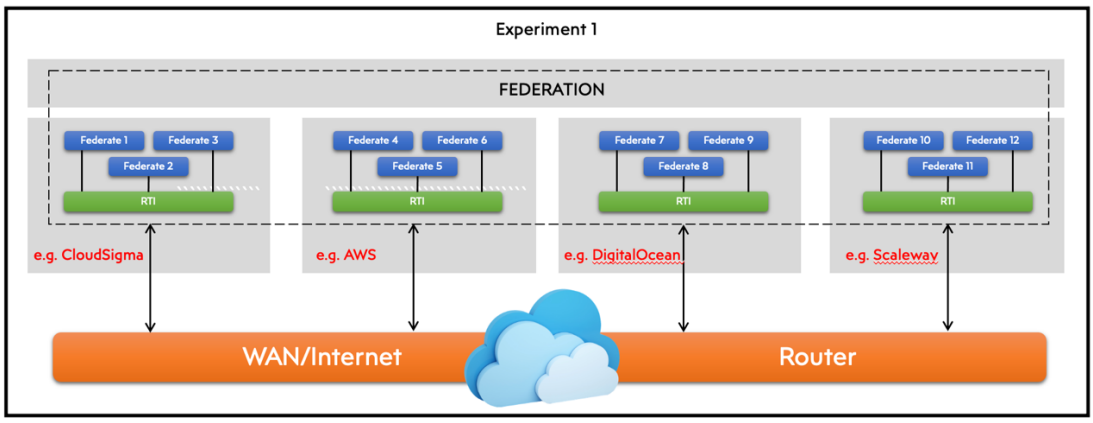


Figure 4-7 Multiple Clouds – Single Experiment Implementation

4.6.3 SCHEME 4a: Multiple Clouds – Multiple Experiments (Parallel)

In this approach, an analyst can run multiple experiments in parallel using multiple cloud platforms. All traffic is relayed to a specified cloud instance via a "WAN router". Figure 4-8 illustrated the experiment design approach with multiple clouds forming multiple geographically distributed federations. Each federation's central component can be initiated from the specified cloud platform depending on the DS design and execution. Moreover, no pattern is followed in the distribution of federates over cloud platforms. The figure only demonstrated a moderately complex experiment design.

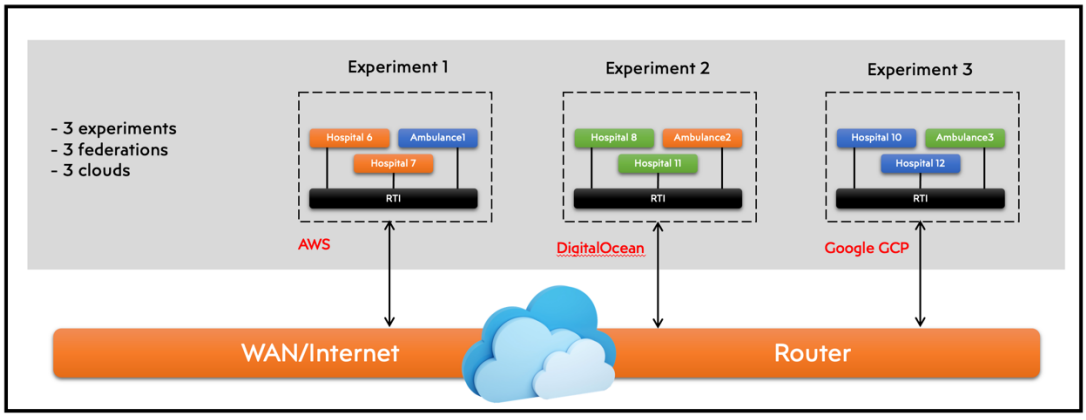


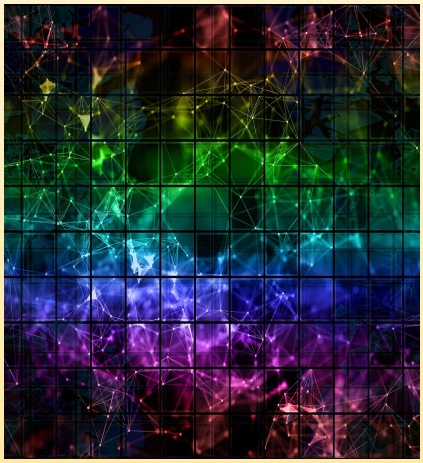
Figure 4-8 Multiple Clouds – Multiple Experiments Implementation

The three schemes presented above are a few possible implementations. Analysts have the architectural flexibility to design a more complex experiment to achieve the set objectives. Due to cost and time constraints, only schemes one and two will be used in the experimental

stage of this thesis presented in the next chapter and subsequent discussions in the 6th chapter – the analysis and evaluation of the results.

4.7 Chapter Recap

We have seen the DICE design process, architecture, and a few deployment schemes for CBDS. The "layered cake" architecture is composed of various services from both cloud and distributed simulation perspectives. The layers and the components involved were explained, and the relationship between them was established. The chapter demonstrates CBDS implementation sequence diagram using the proposed architecture. According to the scenario, analysts have deployment scheme options as to how the cloud-based federation environment is designed to run the experiment. Chapter five takes DICE to the cloud using an EMS prototype which puts the solution to the test. Experimental results were generated and were analysed, discussed, and evaluated in detail in chapter six.



CHAPTER FIVE

DICE IMPLEMENTATION CASE STUDY

Chapter 5 DICE Implementation Case Study

5.1 Chapter Overview

The previous chapter presents the aspects of the proposed framework and architecture development process and possible cloud implementation schemes. It also presents a detailed explanation of various components found in both the CBDS development framework and deployment architecture.

This chapter starts by presenting simulation approaches of ABS and DES, then dives into the experimentation environment setup. The setup includes the cloud infrastructure provider, computing resources, network services, experiment submission, monitoring, and result collection procedure. The following section introduces the case study prototype – the Medical Emergency Service (EMS) and its components comprising an ambulance and hospital accident & emergency (A&E). The interactions between the ambulance, the A&E, the interoperability reference model used, and the time management are all discussed in detail. Moreover, the section presents how the EMS is adapted to the proposed architecture for evaluation. In the end, the reader will find the software tools selected for the experiment based on the research design.

Finally, the chapter reports the EMS model technical specifications, validation, and verification approach, cloud experiment configuration, and execution procedure. Moreover, result collection and performance testing are presented and discussed in detail. A quantitative result is generated, and a comparison is made between the possible implementation schemes from chapter four. This thesis focuses mainly on performance metrics and the technical requirements that affect the execution.

5.2 Simulation Approaches - ABS and DES

As reported earlier in chapter two, section 2.5 and 2.6, the agent-based and discrete event simulations are widely used simulation paradigms. Others include System Dynamics (SD) and PetriNets, but these are not part of this research focus. The hybrid prototype case study of the emergency medical system combines the ABS and DES approaches. The federates are divided into an ambulance service as the central component and accident and emergency department – the hospitals receive severe emergency incidence and walk-in patients. All these will be explained further in the coming sections. Before that, let us

investigate the low-level implementation of the proposed architecture using a befitting case study.

The various layers of architecture provide services, and some of them to the user. To run a distributed simulation experiment on the cloud, the analyst should decide on the type of cloud services needed – private or public, networking environment, and the software tools required for the work.

5.3 Environment Setup

There is a phase in the CBDS architecture implementation that involves the simulation environment setup and submitting the experiment job for execution. The choice of the cloud provider and submission methods are identified in the following sub-sections.

5.3.1 Cloud Infrastructure

Simulation analysis in the cloud provides benefits to analysts in many organisations compared to on-premises expensive computing infrastructure. Cloud computing is attractive due to its virtualisation (Barrett and Kipper, 2010) technology, allowing easy isolation of applications within a shared hardware platform (Menascé and Ngo, 2009). In preparing to test the proposed architecture, there are many aspects considered in choosing the cloud platform; the experiment's objective, testing strategy (as described in section 5.7), infrastructure configuration, provider services, and reliability, and result monitoring.

Layer 3 in the architecture is specified as a cloud provider when it provides the infrastructure to set up the computing resources required for the distributed simulation experimentation. In this research, CloudSigma (https://*CloudSigma.com*), Amazon EC2 (*Amazon Web Services (AWS)*), Scaleway (*Scaleway Cloud services*), Google GCP (*Cloud Computing Services*), and DigitalOcean (*DigitalOcean – The developer cloud*) are the public cloud provider platforms used for testing and evaluation.

5.3.2 Cloud Computing Resources

After the identification of the cloud providers, the next step is configuring the storage, memory, CPU, and networking facility requirements for the experimentation. There is a 10GB storage, 1CPU, 1 GB memory, and one IP address to communicate with the instance in all of the clouds identified for this research. Some cloud providers like DigitalOcean and Scaleway maintain the assigned IP after shutting down for a short

time. Others like Amazon EC2 and CloudSigma, assign dynamic IP every time an instance is shut down or restarts. The federation runs over the Internet/WAN protocol, and the cloud relies on the instance IP addresses to keep the flow of data exchanged during the execution. IP-related failure may cause problems such as federation crashes and deadlock, especially when running multi-cloud experiments where computing nodes span over several geographic locations. Moreover, when an IP is configured to connect to IoT or Industry 4.0 devices, each time an IP fails or is lost due to shut down or server reset, each connected device had to be reconfigured with the new acquired IP to be able to join the federation execution.

5.3.3 Networking Service

For this thesis, a virtual private cloud network environment was set up to manage the participating federates' traffic sent and received. Some implementation schemes and approaches may require a central router to direct traffic from source to destination. In contrast, others, depending on the RTI used, can use the multicast protocol to communicate and exchange data during experimentation execution. The analyst decides the network topology during model design and conceptualisation, which aligns with the system's organisational goal under study.

5.3.4 Experiment Specification (Job Submission)

Another contribution of this thesis is providing a facility to submit DS experimentation jobs to the cloud for execution easily. The process involves defining experiment specifications and a sequence of activities to manage the simulation. DS technology is generally challenging, especially to modellers who are not mainly software developers. Therefore, automating simulation submission and management tasks is believed to encourage using the proposed cloud-based DS.

Ansible's Playbook is chosen to automate repetitive experimentation tasks. Ansible is an open-source automation technology engine. It dramatically improves the consistency, scalability, and reliability of the ICT environment (*Ansible IT Automation*, 2020). The automation can be applied to environments hosted on bare metal servers, the cloud, or other virtualisation platforms. It also automates the configuration of systems and resources ranging from storage, databases, security firewalls, and networks.

This thesis uses and covers a few Ansible script functions sufficient enough to evaluate the CBDS architecture. Ebert *et al*. (2016) believe that Ansible is the easiest to

implement. After all, it does not require installing agents on the client machines because it uses SSH (Secure Shell) to push configurations, which is based on Python. Ansible's configuration is coded in YAML files, thereby reducing the learning curve. This automation tool has recorded successful implementation such as Singh *et al.* (2016), Masek *et al.* (2018), Spiga *et al.* (2018), and Cruz and Casquillho (2019). A script algorism and implementation scenarios are explained in chapter six – the evaluation.

5.4 Testing Schemes and Execution

Six possible implementation approaches with detailed descriptions and figures was reported in chapter four, section 4.6. The DICE was tested against the three schemes due to cost and time constraints: single cloud – single experiment, multiple clouds – single experiment, and multiple clouds – multiple experiment. Each of the schemes will run some federate, and the results were generated, recorded, and analysed accordingly.

5.5 Experiment Monitoring

Another feature in the proposed architecture is experimentation monitoring and logging. This gives useful insights into what is happening with the simulation and computing resources during each run. The user can use Linux internal statistics or install a third-party tool that best suits the objective of the analysis. Here we capture starting and end-time for each run for performance analysis and evaluation purposes.

5.6 Result Collection

In each of the chosen schemes, the experiment result was collected for one month with three replications. The average was used to analyse the performance and scalability using the proposed DICE. Results were discussed after analysis, and this section presents the future research direction for the architecture improvements.

Due to cloud infrastructure cost and experimentation time constraints, this research reported an average of three runs for each scenario as recorded based on the existing literature (Miller *et al.*, 2001; Anagnostou, Nouman and Taylor, 2013; Anagnostou and Taylor, 2017c). Taylor *et al*. (2009) report that there are DS studies by authors who present results for a single run such as Lendermann *et al.* (2003), Riley *et al.* (2004), and Liu, Zou and Ye (2015).

5.7 Client Infrastructure

The topmost layer in the proposed architecture provides an interface between the user and the cloud infrastructure. Here, a browser can be used to access the cloud services provider's platform and set up the on-demand computing resources. FTP clients are used to uploading model data files and download experimentation results. A user command line terminal provides SSH access to the cloud resources for setup and configuration.

Emergency medical services (EMS) is becoming an increasingly well-known operational system used as a benchmark for distributed simulation approaches. With the EMS, running federation involves the development of a specific launch script to run the distributed simulation over different network topologies.

5.8 The Emergency Medical Service (EMS)

A healthcare case study prototype is used - the Emergency Medical Service (EMS), to test the feasibility and evaluate the proposed CBDS architecture. The distributed simulation comprises an *Ambulance* and several *Accidents and Emergency (A&E)* hospital department models (federates). The ambulance component is developed using the ABS approach due to its conditional, dynamic response nature. The A&E federates process-driven, which progresses according to events in time, and therefore, written as DES. The federates use interoperability standards and exchange data during simulation runtime.

Many authors propose and present how the EMS model works such as Pinto, Silva and Young (2015). In their model, a solid line represents the model, and dashed lines show multi-location dispatch where an ambulance can be dispatched from one scene to another when it becomes available to increase response time efficiency. This project makes use of components, interactions, events, protocols, and standards found in medical systems.

As acknowledged in chapter one, the EMS was initially developed by Anastasia Anagnostou (2014), published in OR journal (Anagnostou and Taylor, 2017c), and subsequently published by Nouman, Anagnostou and Taylor (2013), and Chaundhry *et el.* (2016). The base federates, and the middleware is restructured, upgraded for cloud-based deployment, and used in all experimentations. Throughout this research, references and acknowledgment are given where credit is due according to the university and academic ethical requirements.

5.8.1 EMS Interactions

The ambulance federate is the central component of the EMS model, which finds and communicates with DES-based hospital departments within the defined coverage area. The department represented with the accident and emergency (A&E) models receives patients brought by ambulance or walk-ins. An event-driven process continues within the department until the simulation ends. The high-level architecture (HLA) Std. IEEE-1516e (Scrudder *et al.*, 2010), as reported earlier, is a widely used DS standard mainly used in the military. HLA DS standard is used in this research for its data and time synchronisation capability. The runtime infrastructure (RTI) is the centrepiece of HLA, which uses the standardised rule to coordinate information exchange and the interactions between federates, synchronisation, and overall federation management. Figure 5-5 presents the graphical view of interactions using the RTI as the central component.

5.8.2 Interoperability Reference Model (IRM) in EMS

There are enormous interactions that happen between the ambulance, and the A&E federates in this scenario. For example, A&E departments advertise their availability to the ambulance model. This information is kept. When an ambulance wants to transfer the patient from an incident scene, it searches based on the available A&E with necessary treatment facilities and resources. The ambulance federate begin moving the patient for further medical attention. Basically, there are three interactions.

a. **Interaction One -** A&E departments communicate their availability to the ambulance model.
b. **Interaction Two –** The patient (an *agent* in ABS) is transferred from ambulance to DES A&E department, which receives it as an *entity*. A conversion of the patient object from ABS agent to DES entity occurs.
c. **Interaction Three –** After the ambulance model decides which A&E department to take the patient, it notifies that department to reserve the resources to avoid conflicts, delay, or denial.

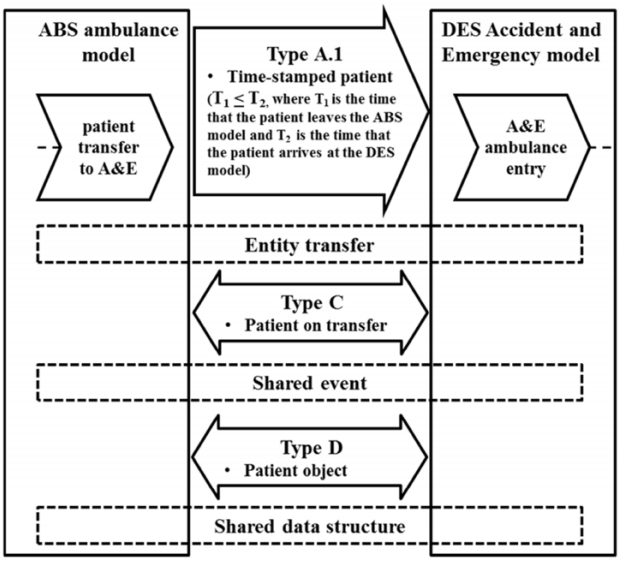


Figure 5-1 IRM used in EMS (Adapted from Anagnostou 2014)

In this study, the IRM facilitates the interactions between the two paradigms used in the EMS hybrid model. The Ambulance Service (ambulance model) is Agent-Based Simulation and several Accident & Emergency Departments (hospital models), which are Discrete Event Simulations. Specifically, the IRM interaction mechanism adopted for EMS is IRM Type (general entity transfer, shared event, shared data structure) or IRM Type (A.1, C, D), and Figure 5-3 shows the IRM model representation.

5.8.3 Data Exchange Protocol and Time Management in EMS

Among the many published DS standards reported in chapter two, HLA protocol is used for time synchronisation and data communication in this work. DS experiments are composed of two or more models (federates) linked together to form a larger simulation called the federation controlled by the RTI middleware of choice.

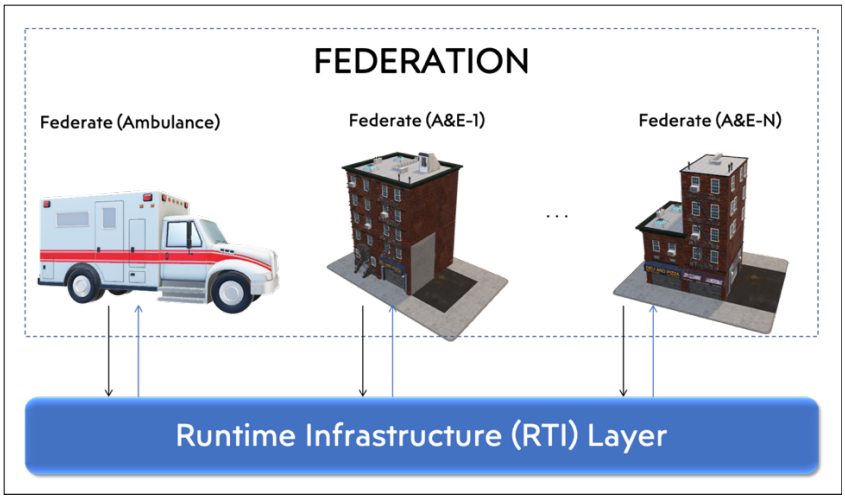


Figure 5-2 DICE HLA conceptualisation (Adapted from Anagnostou 2014)

DICE HLA concept is illustrated in Figure 5-4. The RTI provides federation management services such as data exchange between federates and synchronisation. Both the ambulance (ABS) and the regional hospitals (DES) federates communicates with each other via the RTI as the central controller. Within the federation, several independent hospitals federates all of which communicate with the ambulance model for updates on their availability. More details on the technical functions of RTI will be presented in section 5.5.1 below.

5.9 Adapting EMS to DICE

5.9.1 EMS Model Conceptualisation

Going through the framework implementation, model conceptualisation is a crucial part of the distributed simulation project. It deals with developing the appropriate representation of the real-world domain that applies to the defined problem in the presented scenario. These concepts will be transformed into specific requirements that will later be used during experimental design, testing, execution, results analysis, and evaluation. The EMS conceptualisation in this project comes in two forms, Figure 5-5. The ambulance service federate was developed with ABS and hospital federate in DES.

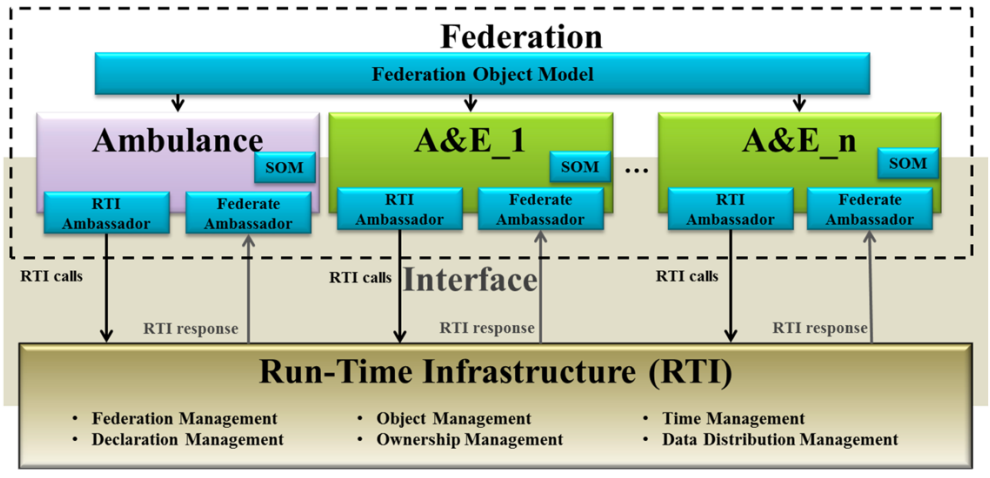


Figure 5-3 Hybrid Distributed EMS Conceptual Model (Simulation Scenario) (Adapted from Anagnostou 2014)

The Ambulance Service Conceptual Model

Emergency ambulance service is usually land-based. Though some scenarios require air ambulance when time is of the essence, or the incident scene is not accessible by road. In any case, land or air, the ambulance service station coordinates calls, vehicle,

and crew deployment based on the reported situation. When an emergency call comes in, the respondent assesses the severity of the accident and decides what vehicle, equipment, and crew to dispatch. The dispatch crew configuration can either be Basic Life Support (BLS) who deals with non-life-threatening cases or Advanced Life Support (ALS) to treat on-scene life-threatening incidents.

When the crew arrives at a scene, they offer the needed services. Depending on their judgement, they may transfer the patients to the hospital for further treatment or release them immediately after the on-site treatment. The ambulance federate has three main periods/states; waiting, service, and response time, as depicted in Figure 5-6, used by the ambulance organisation as the key performance indicators (KPIs). **Waiting** for a state is the time span between when the emergency call comes in, and the attendant finds an available ambulance vehicle. **Service** state is the time when an ambulance vehicle departs the station and ends in the hospital after patient transfer. It may end at the accident scene when no patient is resealed after on-site treatment. **Response time** starts when an emergency call is received and ends when the ambulance arrives at the incident scene.

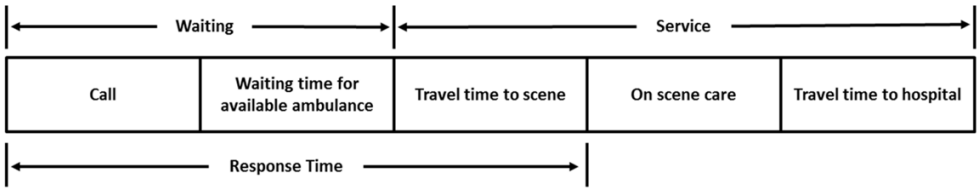


Figure 5-4 Timelines for ambulance service model (Adapted from Fitzsimmons, 1973)

5.10 Justifying the use of EMS

While evaluating the proposed DICE, emergency medical service (EMS) is chosen for having the level of complexity and interoperability needed to test the architecture's feasibility on various cloud infrastructures. EMS is a classic operational research (OR) system that non-technical analysts can use such as medical doctors. I acknowledged that other operational systems might have some level of these characteristics, such as manufacturing or transport systems; the EMS used here have been validated with two simulation paradigms - ABS and DES. This calls for carefully managed object interactions between the two approaches, which put the cloud services to task and significantly affect the result generated from all the experiments conducted. Furthermore, the distributed simulation literature has reported that researchers using these kinds of model to investigate lingering issues and extend knowledge, and for more example of EMS in the literature in addition to those reported in chapter two, Tanika *et al.* (2017), Pinto, Silva and Young (2015), and Yang *et al.* (2019).

5.11 Software Tools

Computer-based modelling and simulation make extensive use of software tools. Many programming languages exist; some are general purpose, while others are domain-specific (Pidd, 1984). One this is clear that there is no one-size-fits-all. Depending on the system to simulate, many factors need to be considered, such as process, entities, interaction, network, data exchange, and security. To test and evaluate DICE, this project uses a well-established DS standard, the HLA Std 1516e. An open-source Java-based simulator and a free and open-source middleware software package are explained in chapter two and more on experimentation-specific details below.

5.11.1 High-Level Architecture (HLA) Distributed Simulation Standard

HLA is one of the matured distributed simulation standards that uses advanced computer technologies to ease simulation development (Ficco *et al.*, 2016). As highlighted in chapters two and four, sections 2.6.3 and 4.4.3, respectively, HLA protocol standard was first conceived and mainly used for military application. As the system gets more complicated, its use is extended and applied in non-military domains. Through HLA, DS allows modellers to develop large simulation model from composing smaller, reusable, independent sub-models. Instead of building a vast monolithic large model from scratch, DS provides an alternative for integrating and interoperating multiple models, each with its own language, operating system, and features.

Federation is the term used for HLA-based distributed simulation, and participating models are called federates. Federates interact with one another using the runtime infrastructure (RTI) middleware that provides services and protocols to manage communication and data exchange within the federation (Falcone, Garro, Taylor, *et al.*, 2017). HLA uses objects, the datasets that are exchanged between federates and events are the interactive communication in the federation (Rainey and Tolk, 2014).

Reusability and interoperability are among the significant challenges in modelling and simulation. Basically, when modellers want to develop large simulation, they have to modify existing code, if available, or develop from scratch, which presents a substantial technical challenge and make simulation project cumbersome. HLA has a specification feature that offers a typical architecture for distributed modelling and simulation. Some HLA implementation produced by some vendors has in-build support for WAN

connectivity, which technically has support for cloud federation execution. In other cases, the analyst may have to program the API to support the WAN protocol for CBDS.

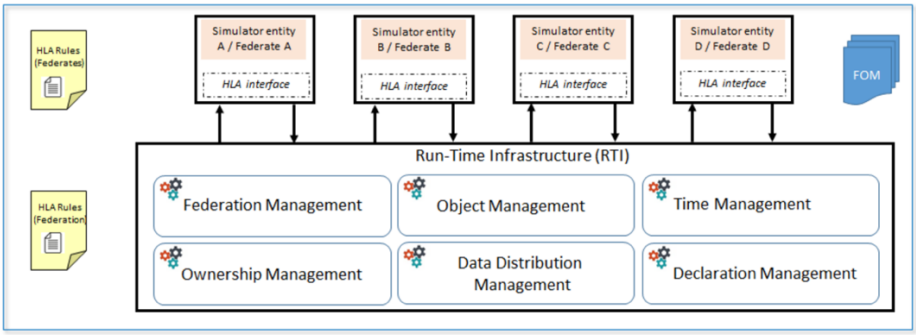


Figure 5-5 HLA Federation Structure with RTI services (Adapted from Gorecki et al., 2018)

A classical HLA provides APIs specification for communication in the federation. Gorecki *et al.* (Gorecki *et al.*, 2018) explain how the RTI manages federation using various services, as illustrated in Figure 5-5. According to the HLA federate interface document (Scrudder *et al.*, 2010), the federation and interaction and the RTI management services are defined. Typically, in an HLA-based DS, there are three components (Huiskamp and Berg, 2016) involved - the HLA Framework and Rules, Federate interface Specification, and object model template (OMT).

5.11.2 Simulator

This project uses an open-source Recursive Porous Agent Simulation Toolkit (RePAST) Symphony (https://repast.github.io) simulator. RePAST is the leading free and open-source large-scale agent-based modelling and simulation library. Users build simulations by incorporating Repast library components into their own programs or using the visual scripting environments (Macal and North, 2006). Its effectiveness in large-scale modelling and simulation of social phenomena has been assessed (Tobias and Hofmann, 2004). Other successful implementation includes Chaudhry et al. (2016), Minson and Theodoropoulos (2008), Garro *et al.* (2015), Crooks (2007), and Anagnostou and Taylor (2017a), and Collier and North (2013).

5.11.3 Middleware

CBDS proposed architecture is a new proposal and requires continued support, research, and development. Therefore, a flexible and modular RTI implementation is highly desirable, and poRTIco opensource RTI fulfils this criterion. Licensed under the Common Developer and Distribution License (CDDL), poRTIco project was initially

developed by Tim Pokorny and Michael Fraser in 2005 and 2007. The project received funding and support from the Australian Defence Simulation Office (ADSO) (*Portico History*, 2008). Since then the literature recorded successful use by authors such as (Tu, Zacharewicz and Chen, 2011; Anagnostou, Nouman and Taylor, 2013; Chaudhry *et al.*, 2016; Akram, Sarfraz and Shoaib, 2019).

PoRTIco is selected for this research to support the defence and IEEE distributed simulation networking standard - the High-Level Architecture (HLA) 1.3 and 1516e (Evolved). The proposed DICE test will run on a cloud. The latest poRTIco v2.1 release has built-in support of wide area network (WAN) bridging capability (*Portico Over a WAN*, 2020). This feature allows for both UDP and multicast data exchange mechanisms.

5.12 Verification and Validation (VV)

To ensure we are building the simulation models the right way, Verification and Validation (VV) activities are essential. Caughlin (1995) describes *verification* as the process of determining whether the model accurately represents the conceptual design, specifications, and behaves as intended. During verification, model outputs are compared with the design specification and expectations. This process is repeated, as illustrated in Figure 5-8 until the computerised model is satisfactory (Sargent, 2013).

On the other hand, the *validation* of activity is used to determine how well a model represents the real-world system it was intended to simulate. Successful validation adds credibility to a model and its output (Robinson, 1997).

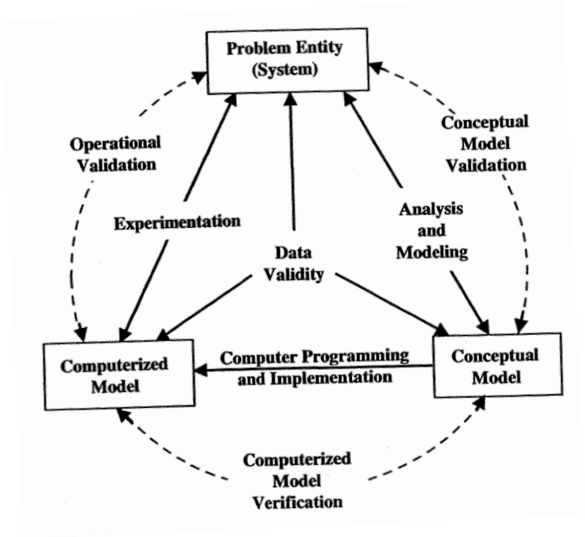


Figure 5-6 Model Development Process with Verification and Validation (Adapted from Sargent, 2013)

EMS Verification - EMS development process includes iterative verification checks in both ambulance and A&E federate conceptual design and translation. The model is interrogated continuously to make sure they conform with the conceived design specifications.

EMS Validation - A test run is conducted in local machines before deploying to the cloud server. This is to make sure it is doing and displaying the correct intended behaviours. This is confirmed in the coming section, during result analysis, where the output corresponds to the NHS UK's targets.

Furthermore, the middleware implementation is verified and validated while programming the models. The poRTIco RTI used is developed and tested to be in working condition. EMS is coded to implement the various HLA and RTI interface APIs. These allow interaction between participating federates during the simulation run. Figure 5-9 shows the sequence of interactions between models in the federation via the RTI serving as the coordination layer during the CBDS experiment.

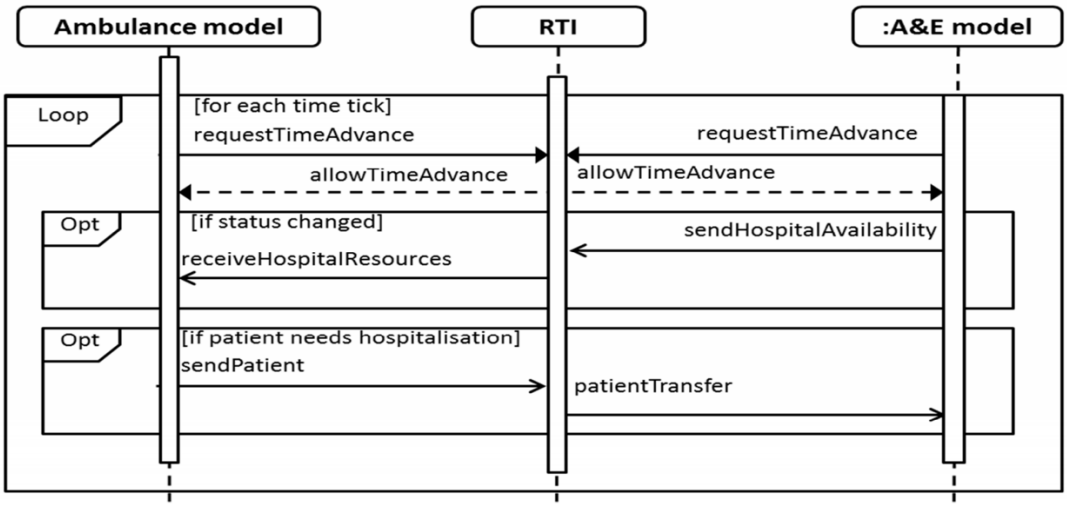


Figure 5-7 Sequence diagram of the interactions using the poRTIco middleware (RTI)

5.13 Experiment Setup

Experimentation executes the simulation runs, collecting results, and using statistics to compare the various configuration scenarios used to check the system performance. In this project, the proposed DICE, as reported earlier in chapters four and five, maybe implemented using three possible schemes. Each implementation has slightly different configuration settings described below.

5.13.1 Cloud Instance and Network Settings

As explained earlier, the proposed architecture is tested for feasibility using two main testing categories – performance of execution time and scalability testing. The second scenario increases the number of federates at each run to upscale the DS. This test observes the scalability behaviours of the proposed architecture. The experiment was executed in a well-controlled cloud computing environment to maintain consistent data exchange and interaction in the federation. Table 5-1 summarises the cloud-based DS experimentation setup.

Table 5-1 CBDS Experiment Settings

Component	**Package/Service**	**Remark**
Simulator	RePAST Symphony v2.1	Opensource Java Simulation Package
Runtime Infrastructure	poRTIco v2.1	Opensource middleware with HLA IEEE 1516e support
DS Standard	High-Level Architecture (HLA) Evolve	IEEE Std. 2010-1516e
Cloud Infrastructure	CloudSigma, Amazon EC2, Scaleway, Google GCP, and DigitalOcean Instances each with. - 10 GB SSD Storage - 1 GHz CPU - 1GBRAM	Public Cloud Platform
Operating System	Linux-Based Ubuntu 18.0.4 LTS and 20.04 LTS	Opensource OS
Programming Runtime	Headless JRE from OpenJDK version 11	Works for both the Simulator and the Middleware
Security	Incoming: SSH, TCP, UDP Outgoing: All Traffic	Only the allowed IP and port numbers in the federation configuration file are allowed
Networking	Wide Area Network (WAN), Virtual LAN (VLAN), Internet	IPv4 Public Addressing

The cloud instances configuration in Table 5-1 shows allocated resources for this experimentation purpose. Each instance contains one federate (a distributed model).

For example, the first run with three federates comprises three cloud instances. Each instance holds one federate during the simulation run.

CloudSigma, Amazon EC2, Google Cloud Platform, DigitalOcean, and Scaleway providers were selected for the test. Because of the number of simulation runs and replications needed for DS, executing large-scale DS experiments on cloud or LAN requires a vast computing resource. The requirements for CBDS may be complicated due to the regional networking involved. CloudSigma infrastructure is used by researchers with success, for example, Anagnostou *et al.* (2019), Kovacs *et al.* (2020), Visti *et al.* (2016), and Kovács, Kacsuk and Emődi (2018).

5.13.2 Execution Procedure

A few logical steps are followed to launch the CBDS experiment in this work. These are.

Step 1: Start cloud instances and get assigned IP addresses.
Step 2: Prepare and upload data files to the input directory of each federate participating in the DS.
Step 3: Create a federation by starting the EMS ambulance sub-model and synchronise participating federates using the IP addresses.
Step 4: Prepare the launch script with parameters such as cloud instance IP address, WAN router gateway configuration, and execution time logs.
Step 5: At the end of each simulation run, download CSV-based results from the model output directory and clear it (optional) for the next run.
Step 6: Perform analysis on the overall results.

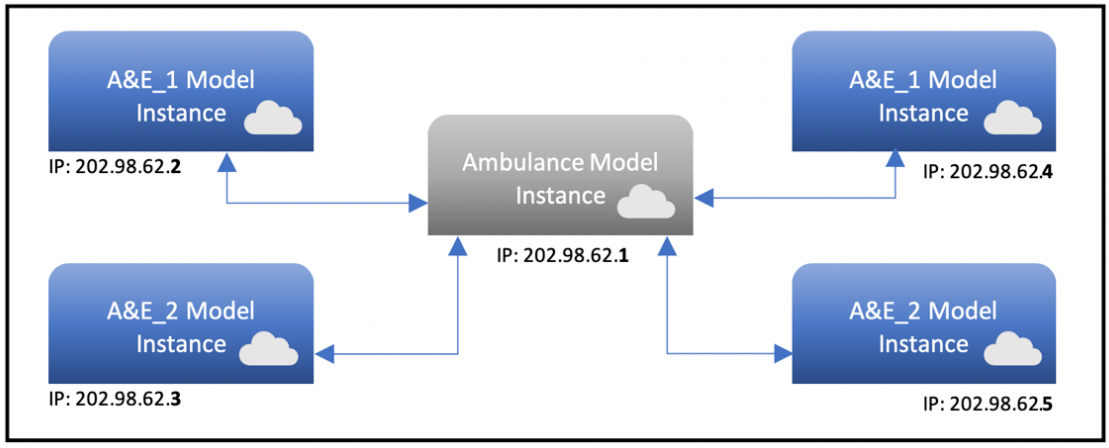


Figure 5-8 Example EMS Cloud Instances Setup with sample IP Addresses

Figure 5-10 shows an example of an EMS federation scenario where each federate is assigned a dynamic IP address provided by CloudSigma. It is worth noting that the IP address is renewed (change) whenever an instance shuts down. To maintain an IP address in many cloud service providers, including CloudSigma, users need to purchase a static or dedicated IP address. Users can assign them to various federates in the CBDS project.

5.14 Experimental Results

The experiment generates results from three replications for each scenario to evaluate the DICE implementation scheme. It is reported that three iterations are a good number to use, which reduces the variance due to the operating system and communications network effects (Taylor *et al.*, 2009). The experimented scenarios were conducted with 17 federates, as earlier demonstrated for performance and scalability testing and analysis. The fifth column – *Ave. Time (minutes)* in Table 5-3 shows that each run during the experiment took an average time between 75.4 and 106.2 minutes as the federates increased. Succeeding sections presents the results along with discussions and findings.

5.14.1 Performance and Scalability

Three types of experiments have been designed to test and observe the performance and scalability of the distributed simulation federates using various cloud computing environments. This is important because the failure of applications due to performance-related issues can be prevented with pre-deployment performance testing (Sarojadevi, 2011). In this project, results were collected from the DICE's three schemes as reported in chapter five. The following graphs present variant results from the three runs for each combination of federates. It is important to note that the execution run times are recorded and reported in minutes; seconds are ignored to have a less complex and more evident analysis. All results are an average of four weeks collection period. Tables showing the values for each run and the calculated average can be found in the appendices.

Figure 5-11 Y-axis represents simulation execution time in minutes. The X-axis shows the number of federates during each run. From the Figure, as the number of federates scales up, the execution time increases. As expected of cloud-based simulations, the execution time difference between the number of federates raises steadily with deviations. This is traced to performance analysis of distributed systems (Teo and Tay,

1996), where the workload may increase communication overheads resulting in a longer simulation elapsed time.

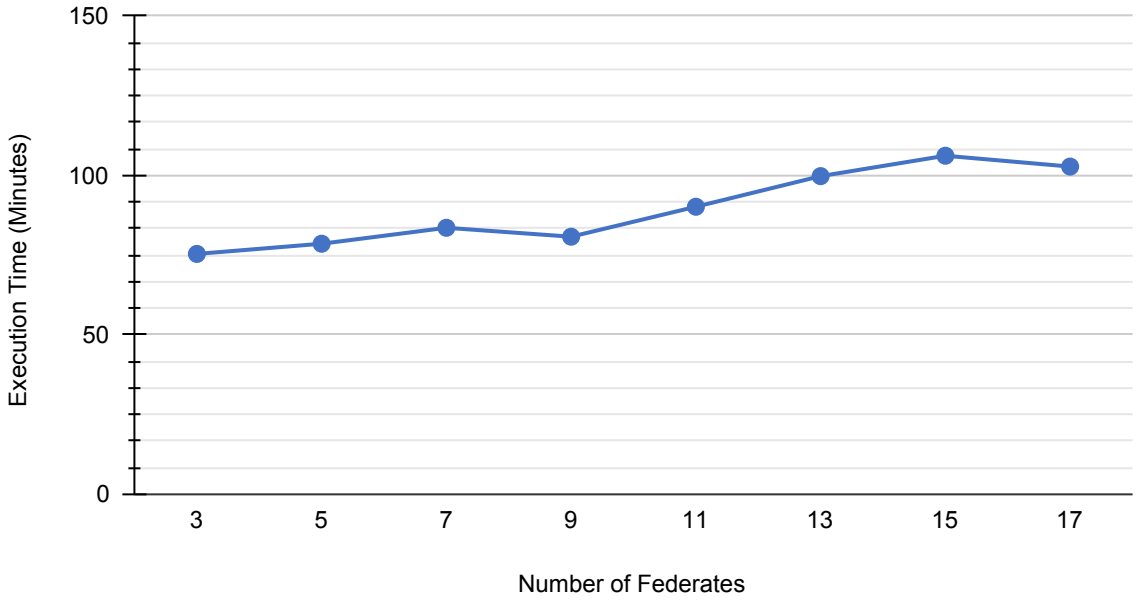


Figure 5-9 Scheme 1: Average of 3 Runs in Minutes

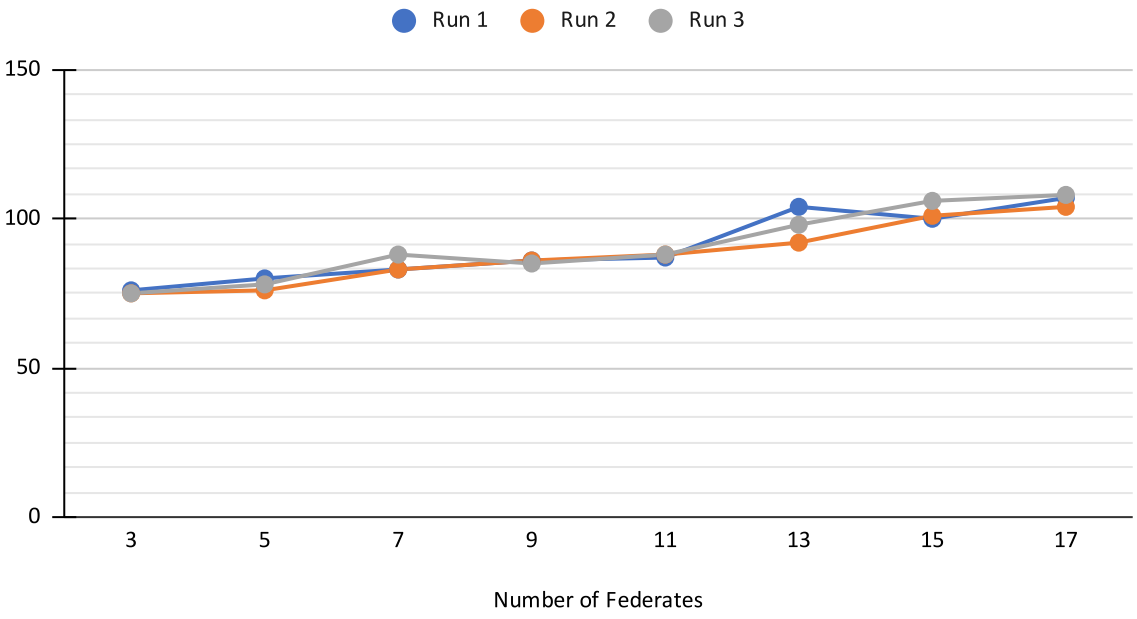


Figure 5-10 Scheme 1: 3 Individual Iterations in Minutes

With this experiment setup, the single federation simulation execution time starts from 75 minutes for three federates and goes up to 108 minutes with 17 federates, i.e., one ambulance and 16 hospitals. Considering the average time plotted

in Figure 5-11, the time difference dropped noticeably. This is visible between seven federates lasting for 83.6 minutes while nine federates averages at 80.8 minutes. Moreover, 15 federates takes 106.2 minutes on average, while 17 federates spent 102.8 minutes. Figure 5-12 compares the three individual iterations (Run 1, Run 2, and Run 3), and it shows an interesting behaviour with 13 federates where all the three runs executed at the different timeframe. Other combinations have closer execution time.

Scheme 2a of the proposed DICE implementation also executes one experiment in a single federation. Here, multiple cloud platforms are connected using an on-cloud Wide Area Network (WAN) router where all participating federates relay datagram traffic from source to destination. Because this scheme employs more than one cloud platform, the runtime starts well over two hours. Precisely, it starts at 216 minutes for three federates and up to 442 minutes for 17 federates. The standard deviation presents a high variability starting with 1.73 minutes for three federates, rise as high as 30.09 and 41.40 minutes for seven federates and drastically fall back to 1.73 minutes for 17 federates.

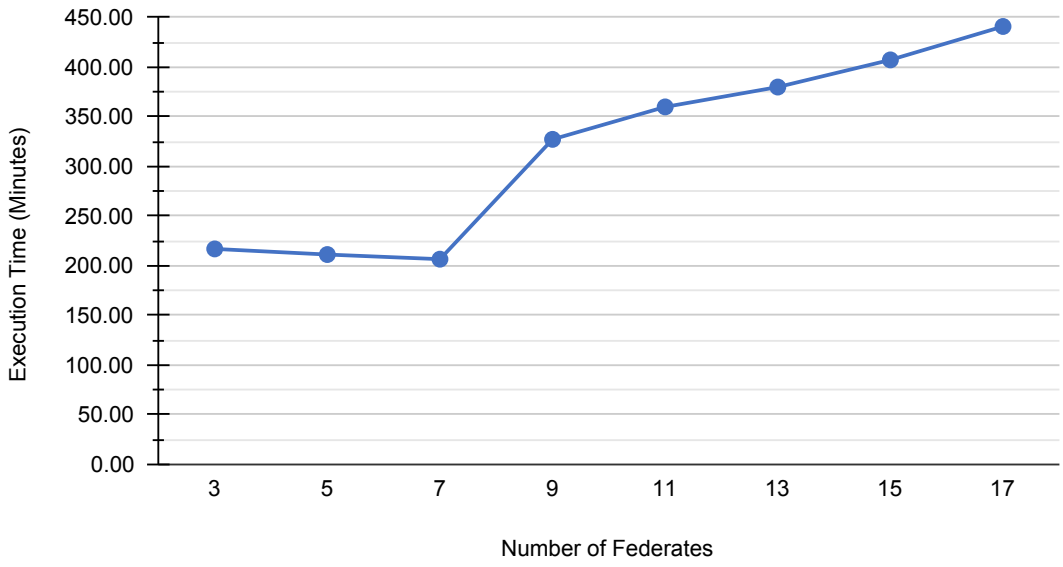


Figure 5-11 Scheme 2a: Average of 3 Runs in Minutes

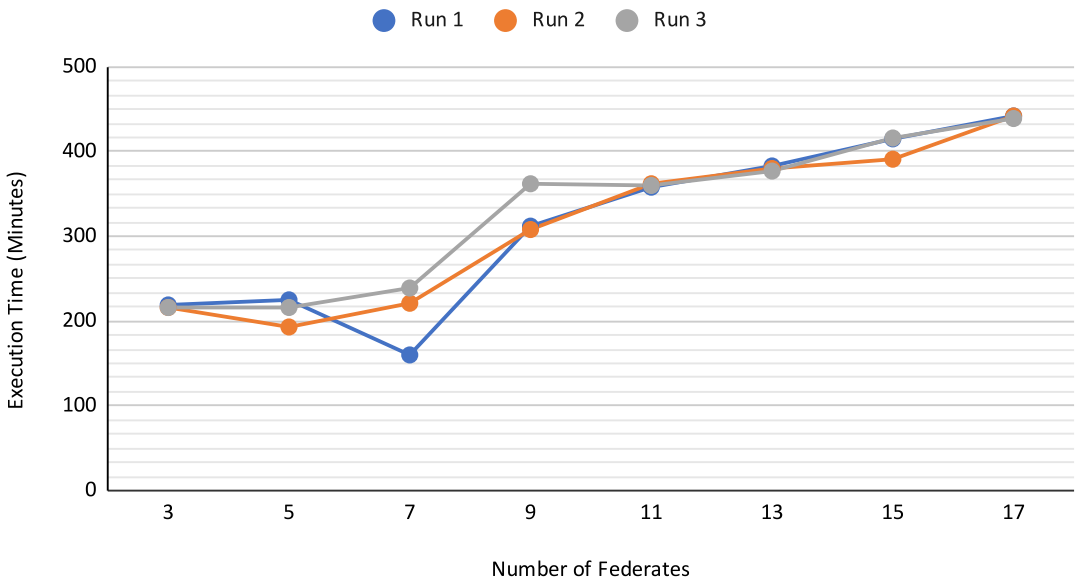


Figure 5-12 Scheme 2a: 3 Individual Iterations in Minutes

In Figures 5-13 and 5-14, like previous figures, Y-axis represents simulation average execution time recorded in minutes while the X-axis shows the number of federates during each scenario runs. As shown in Figure 5-13, the beginning time falls for the first three scenarios and heavily rises between seven and nine federates. The time increases as the number of federates go up. However, looking at the individual iterations in Figure 5-14 reveals that only the first iteration experiences a dramatic fall between the three and five federates. The remaining iterations go up steadily with recorded variations. The behaviours can be caused by different responses from various inter-connected cloud infrastructure.

In contrast, scheme three is designed to compare with the second scheme, which has a traffic router on the cloud. Here, the router is configured locally on a physical machine on a non-dedicated Internet and has no specific resource requirements besides the basic networking setup. Due to the high rate of network failure observed from this scheme, only three scenarios experimented - the three, five, and seven federates combinations. However, the result shows a significant effect of on-cloud and off-cloud routing. The concept considered a DS where the analyst designed to route traffic down to physical infrastructure for management, security, or integration with digital twin systems. The Figure shows that simulation run time starts from 139 minutes for three federates and 195 minutes for seven federates. The deviations between the scenarios were 1.53 minutes on the lower side and 6.43 minutes being the highest.

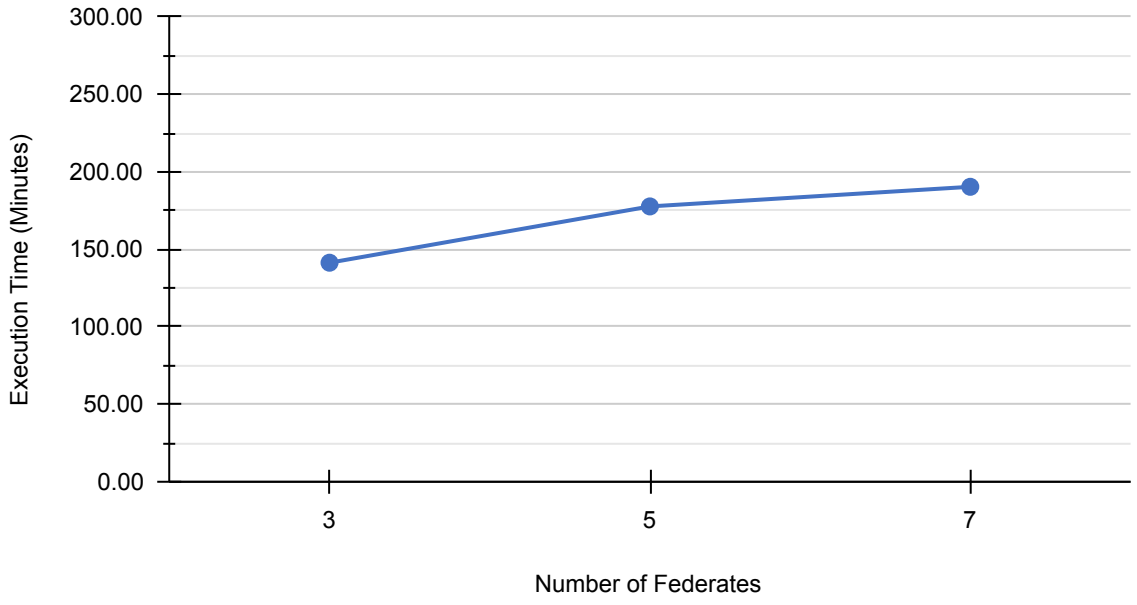


Figure 5-13 Scheme 4a: Average of 3 Runs in Minutes

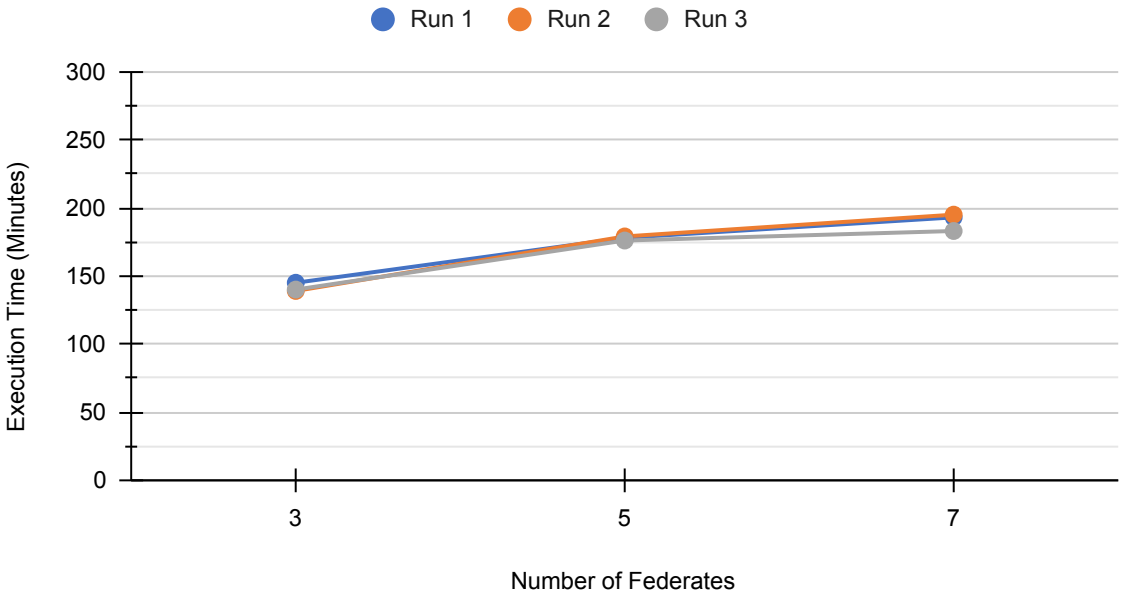


Figure 5-14 Scheme 4a: 3 Individual Iterations in Minutes

It is clear from Figure 5-15; this experiment's execution time goes up from the start as against the previous scheme. However, the standard deviation falls and rises again on the third set of iteration. The three scenarios ran with centralised traffic routing through a physical machine; hence the consistent and steady increases as the number of federates grew. Furthermore, the individual iterations in Figure 5-16 present the consistency from one combination to another. They are almost identical in the execution time on average.

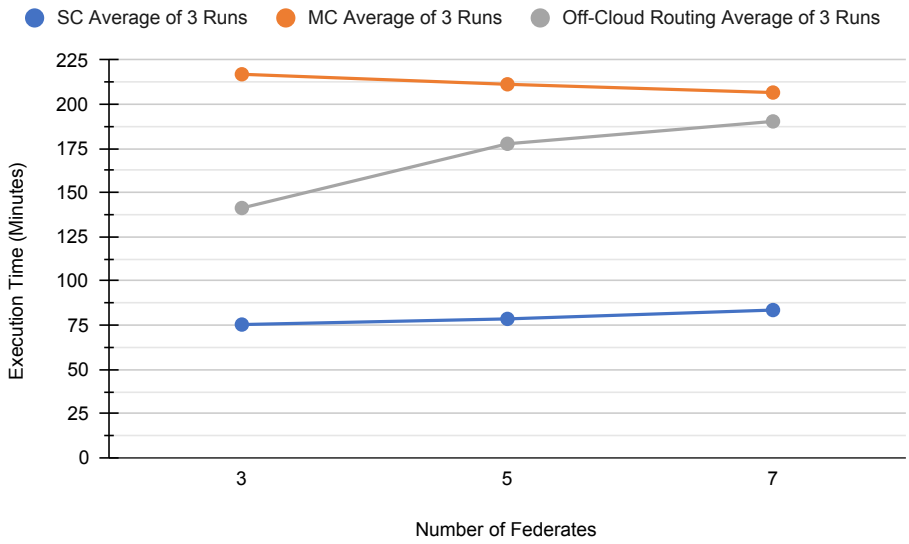


Figure 5-15 Comparison between three scenarios of the three schemes

SC = Single Cloud MC = Multiple Cloud

Going deeper in analysing how the cloud infrastructures perform with different scheme configurations, Figure 5-17 combines and compares the three schemes' average for the first three scenarios, i.e., three, five, and seven federates. The graph in Figure 5-17 illustrates the data showing schemes one and three starts at the beginning and increases as more federates are added. Interestingly schemes two start and the time drops and then rise significantly, as seen in Figure 5-13. This is an interesting finding, and it is elaborated in the discussion chapter.

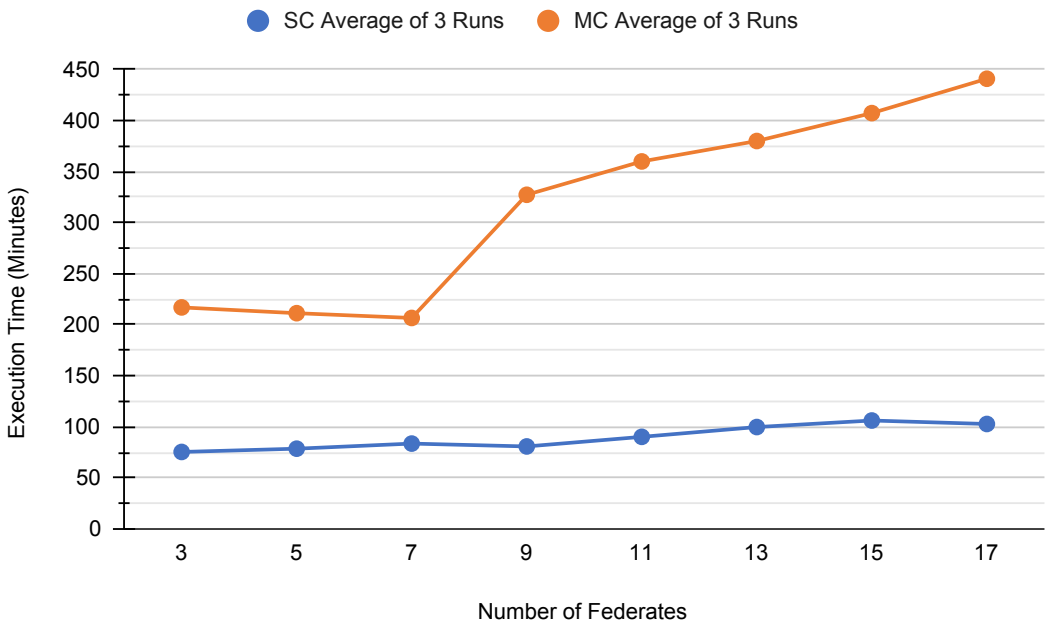


Figure 5-16 Average execution time between schemes one and two

Executing distributed simulation with all federates on nodes within a single cloud platform performs differently from when the experiment is distributed over multiple cloud service providers. Similarly, Y-axis represents simulation average execution time recorded in minutes, while the X-axis shows the number of federates during each scenario. The plotted data in Figure 5-18 visually compare the two schemes where the DS execution time on the single cloud rises from the beginning and continues to go up as more federates are added to each succeeding scenario. Noticeably, the multiple cloud runtime falls initially and then significantly rises between seven and nine federates. It continues to rise steadily as the federate scale up.

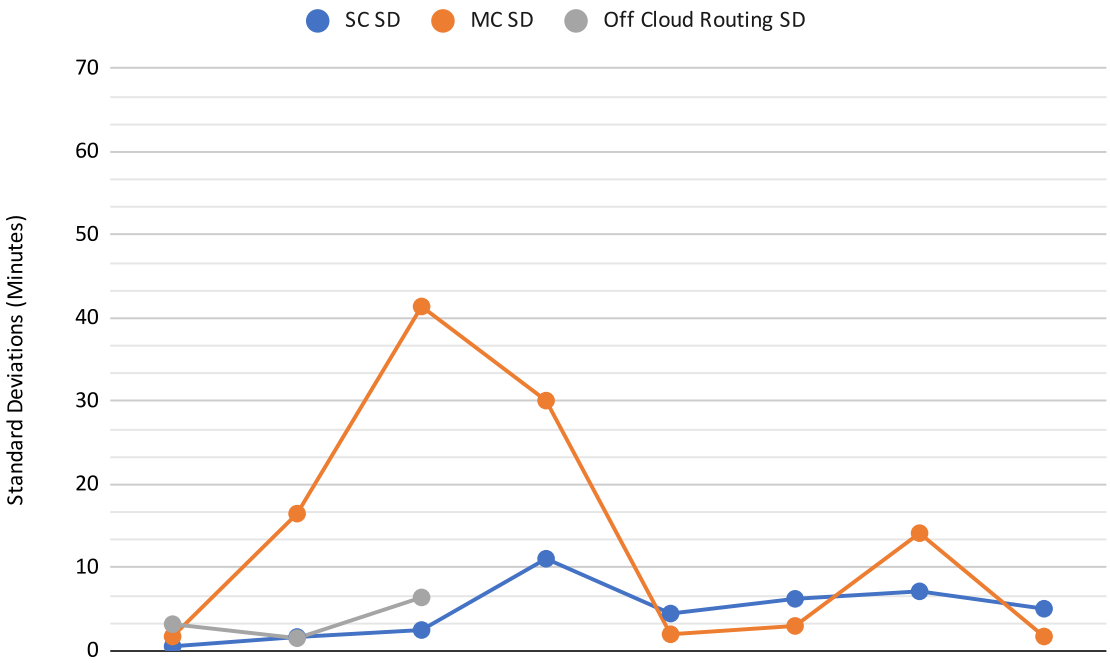


Figure 5-17 Execution time Standard Deviation (SD) for the three schemes

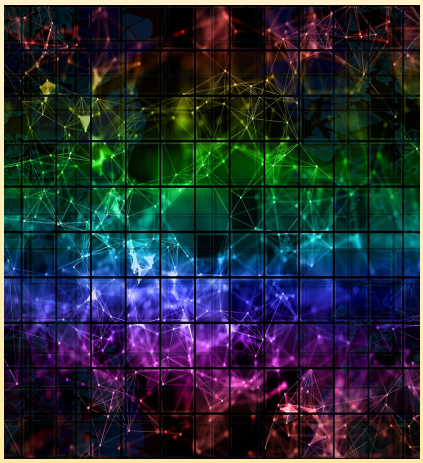
Figure 5-17 presents the standard deviations for the three schemes. The standard deviations are calculated based on the three iterations for each scenario, i.e., three iterations of 3, 5, 7, 9, 11, 13, 15, and 17 federate. Though scheme three shows only up to seven federate, the figure shows the differences in variation between iterations when distributed simulation is executed on single, multiple clouds, or multiple clouds with local on-premises WAN router. Here schemes two and three appeared to be deviating within a close time range while the multiple clouds execution with on-cloud router shows a significant variation in time.

The result presentation above concludes the chapter, and thorough discussions and findings are presented in the next section.

5.15 Chapter Recap

The preceding chapter explains the proposed architecture and framework development process. It also shows the layered architecture, which aims to ease the use of CBDS by both technical and non-technical experts' analysts. This chapter moves further to implement the architecture using the EMS prototype model. The chapter describes in depths the model components, including sub-models, interoperability, interactions, and simulation events. This section also shows how the model is adapted to the cloud-based architecture for experimentation and analysis. Importantly, this chapter presents the experimental results and the technical specification used for the environment. The case study model is explained and adapted to the proposed framework.

The next chapter interprets and discusses the results, presents findings, evaluation, and research limitations.



CHAPTER SIX

DISCUSSION AND EVALUATION

Chapter 6 Discussion and Evaluation

6.1 Chapter Overview

The previous chapter explained the proposed architecture, the design, and the development process. It also presented the experimentation setup, the case study prototype used, and the choice of software tools for the evaluation. Moreover, it explained the Model Realisation stage, where the models to be used for experimentation are defined and configured. Results from the experimentations were processed and presented as-is.

This chapter revisit the research problem and presents the key findings from the result analysis. It also discusses the results in more detail. The sections are organised as to how the results relate to previous research. The findings differ from other studies, how the results and findings confirm some existing theories, and the practical implications and the contributions to the field. Finally, the chapter evaluated the DICE architecture from the research perspective.

6.2 Research Problem and Key Findings

6.2.1 Revisiting the Research Problem

Established in chapter one, DS is a method in operational system analysis that has gained interest due to its claimed benefits, including model reusability and interoperability. DS allows the exploitation of geographically distributed resources such as equipment and people (Fujimoto, 2015b). However, the cost of high-performance computing resources, technical skills, and special training required to design, develop and use DS is an ongoing concern. These are the long-standing challenges that have prevented the broader adoption of parallel and distributed simulation technology (Fujimoto, 2016).

The cloud computing concept offers an alternative approach to address the issues mentioned above using the pay-as-you-go economic model, eliminating considerable investments in the required hardware and software. DS evidently have the potential to benefit M&S. Nevertheless, relatively limited attention has focused on the development framework and deployment architecture to enable analysts to run DS experimentation on the cloud. Therefore, a more in-depth study is needed to understand how modellers will run cloud-based DS and how the cloud platforms will perform with variant parameter inputs. This research has studied, identified the gap from the literature, designed and proposed a development framework and deployment architecture. An emergency

medical services prototype model was developed and used to run various experimentation on selected cloud infrastructures where performance data was collected and presented in the previous chapter.

6.2.2 Key Findings

It is a general belief that Distributed Simulation (DS) development is a complex process and requires expertise with immense courage to undertake. However, a proposed architecture is introduced and experimented with potential benefits for modellers. The cloud-based development methodology guides analysts at every step of the cloud-based distributed simulation (CBDS) implementation - from concept to cloud execution. Almost all the results collected indicated a higher experimentation time. As literature such as (Fujimoto, Malik and Park, 2010) reported, this behaviour is expected and explained why it takes longer in the cloud than running the same model on local machines. Ultimately, the experiment proved it is feasible to study a large-scale DS using cloud infrastructure.

The results indicate that it is feasible to connect and run geographically distributed simulation experiments using cloud infrastructure. Furthermore, the research finds that running a federation on a single cloud performs differently than federation execution on multiple cloud platforms. The significant differences are primarily attributed to how each cloud service provider handles network traffic and the overall communication overheads found on the Internet. These results are explained in detail from the following section onwards.

6.3 CBDS Experimentation Result Summary

The three schemes are executed in line with the performance testing of the distributed EMS simulation model. The first scheme involves two cloud platforms: CloudSigma and Amazon EC2. The former hosts the ambulance model, and the latter holds two hospitals. In scheme 2a, CloudSigma, Google Cloud Platform, and Amazon EC2 were used. Lastly, in scheme 4a, five providers were used: CloudSigma, Amazon EC2, DigitalOcean, Google Cloud Platform, and Scaleway. Each scenario runs thrice and for 30 days simulation time.

Table 6-1 Results summary of Schemes 1, 2a and 4a.

DICE Implementation Approaches	Scheme 1		Scheme 2a		Scheme 4a	
Execution Time	Avg. (mins)	SD	Avg. (mins)	SD	Avg. (mins)	SD
3 federates	75.4	0.55	217.00	1.73	141.33	3.21
5 federates	78.6	1.67	211.33	16.50	177.67	1.53
7 federates	83.6	2.51	206.67	41.40	190.33	6.43
9 federates	80.8	11.08	327.33	30.09		
11 federates	90.2	4.49	360.00	2.00		
13 federates	99.8	6.26	380.00	3.00		
15 federates	106.2	7.16	407.33	14.15		
17 federates	102.8	5.07	441.00	1.73		

The results are shown in Table 6-1 are the average and the standard deviation of three runs. The shadowed grey areas indicate that running the CBDS experiment with an off-the-cloud WAN router was challenging due to network reliability issues with local devices connecting to the internet. The simulation fails after a more extended time. It is observed that the three, five, and seven federates works well because the execution time is not much. However, running with nine or more federates, the execution time gets longer, increasing the change between failure (MTBF). The results also infer that the DS has insignificant variation in execution time as the number of federates increases.

6.4 Discussion

Figures 5-11 to 5-19 in chapter five, section 5.14.1 presented the results of the DICE's three implementation schemes and the standard deviations for each scheme. In each scenario, the hybrid Emergency Medical Service (EMS) model is used. The EMS is comprised

of an ambulance service as an ABS sub-model interacting with several accidents and emergency hospitals as DES sub-models. The detailed activities performed during the federation execution were discussed earlier. Experiments are performed on public cloud infrastructures; CloudSigma, DigitalOcean, Scaleway, and Google Cloud Platform. Ansible automation tool was used to monitor and log the initiation and execution time. The results in the tables show individual scenario iteration time, average execution time and standard deviation. For this research, only the average time is used to evaluate the performance as depicted using the line-graph figures. Three additional graphs combine the three schemes' average time, the second compares scheme one and two, and the third one presents the standard deviations of the three schemes.

In line with the objectives, the results collected and analysed shows that modellers can design, develop, and run distributed simulation experimentation using cloud infrastructures. Presented in the previous chapter, the results are plotted using line-based statistical graphs, and the figures gave the data visualisation of how the cloud raise and fall performance using different scenarios and environment setup. The first research question in this study is concerned with how cloud services can deploy and run DS. Based on background review; this research has identified the technical components required to design and develop a 'cloud-aware federate (a distributed sub-model). Using an established method, these components were used effectively in creating a DS development framework, which guides the analyst on the necessary (iterative) steps required to develop and successfully run a distributed simulation experimentation project on cloud infrastructure. Overall, the CBDS aims to save time and investment on the high-performance computing resources needed to analyse medium and large-scale operational systems. Figures 5-11 and 5-12 indicated that as the federation workload and/or federates scales up, the execution time is affected with variant standard deviations.

The deployment schemes presented in chapter four calls for different environmental setups for the experimentation stage of this research. For example, scheme one, single cloud - single experiment, all federates reside on a single cloud platform. The regions may differ, but the infrastructure has the same architecture and behaviours unless the user configured otherwise. In this case, virtual machines were created on the CloudSigma platform and configured to route traffic using a gateway federate. The participating federates interact with the interoperability reference model (IRM) presented in the proceeding chapters. The data exchange was not affected by running experimentation through the wide area network/Internet, but the speed is affected by the internet protocols and communication overheads supported by the literature. This addressed the second research question seeking

to know the factors affecting the interoperability amongst distributed models in a cloud-based federation.

From scheme two, multiple clouds - single experiment, Figures 5-13 and 5-14 illustrated the data for average three scenarios iterations and individual scenarios iterations, respectively. The reliability of the result can be seen from the consistent pattern among the combined average and individual execution times measured in minutes. However, the significant differences between a scenario with seven and nine federates is believed to have been caused by several incoming requests and load balancing (Fayoumi and Arabia, 2011), resource allocation and release timing (Losup *et al.*, 2009) and the number of I/O operations affecting the network (Mei *et al.*, 2013) which are among the established cloud performance evaluation criteria.

The third research question seeks to answer what factors are affecting the cloud-based DS experimentation speed. This poser directly relates to many factors uncovered during the experiment design. These include the availability of computing resources at a given point in time, network configuration, workload, scalability, and VM location. Also, authors such as (Khanghahi and Ravanmehr, 2013; Khalid, Abdullah and Rashid, 2016) in their publication, lists the following as potential factors affecting cloud performance; security, recovery, service level agreement, network bandwidth, memory capacity, buffer capacity, disk capacity, fault tolerance, and a number of users. Likewise, the cost is often contributing to cloud performance; even though this was not studied extensively, it was only realised during the purchase/subscription of cloud services used during experimentations.

This research, therefore, established that performance and speed of a cloud-based DS can be negatively affected by overheads, and the literature confirms that contention in virtual machines operating on a shared infrastructure brings noticeable performance overhead (Xu *et al.*, 2014). Furthermore, cloud-based DS can benefit analysts more when executing a specialised high-performance computing infrastructure (HPCI). Nowadays, cloud service providers begin to introduce HPC instances (Liu *et al.*, 2012) specialised for high-performance applications such as CBDS applications, for example, Amazon's HPC (AWS *High Performance Computing (HPC)*), Google Cloud Platform HPC (Google *High Performance Computing (HPC)*), Oracle Cloud Infrastructure HPC (Oarcle *High Performance Computing (HPC)*), and Azure HPC (Microsoft *High-performance computing*). However, it is beyond the scope of this study to deploy and perform experimentations using this kind of infrastructure to evaluate DICE or the DS development framework. Nonetheless, DICE architecture is

designed with flexibility and technical capability to deploy DS on any IaaS, though performance may depend on many factors as indicated above.

Finally, cloud performance and experimentation speed in cloud-based distributed simulation project is a critical issue. As established in this study, it depends on many factors. The objective of this study is to initiate a deep enquiry into the problems. For now, this work has designed and proposed a deployment architecture - DICE, which open doors for further research in the factors affecting general simulation and DS on cloud environments.

6.5 Results Implication

The various experiments carried out in this research provides new insight (cloud performance and scalability based on the experimentation scenarios) into the use of cloud infrastructures to deploy geographically distributed simulation federates. This includes how the cloud computing nodes behave during DS experimentation. It also uncovers the critical components, networking, middleware, gateway, router, etc., that are required to set up CBDS infrastructure for large-scale operational system analysis. Moreover, the empirical research has practical implications and has contributed to the cloud-based distributed simulation (CBDS) approach and focussed more on analysing operational research systems by less technical modellers. The principal contributions of this thesis are:

- It proposes a scalable CBDS deployment architecture - the **Di**stributed Simulation **C**loud Architecture for **E**xperimentation (**DICE**). DICE becomes the foundation of this thesis research, which provides technical specifications and guides analyst on how to deploy DS on various cloud platforms.
- It introduced a cloud-compatible distributed simulation federates development framework, which has origin in the Distributed Simulation and Engineering and Execution Process (DSEEP), an IEEE standard.
- In chapter five, section 5.16.1 presented the experimentation results, this work exploits and demonstrated how network communication and interference affect execution time and overall cloud performance.

6.6 Evaluation

The proposed architecture - DICE was born out of a gap identified from the literature review and has been conceptualised, designed, and developed. To evaluate and subsequently release it to the research arena, validation is required. It was carefully carried out using a befitting prototype case study - the EMS. DICE maps the research objectives and

questions. Furthermore, a new CBDS development methodology was introduced and used to develop the prototype. The two proposals development framework and the CBDS deployment architecture are open to external evaluation, where practitioners who are the primary target end-user could explore more perspective based on their use cases.

The CBDS architecture in this research follows the layered cake model of a cloud computing concept. The approach separates the application, the underlying cloud infrastructure and resource management complexity. This was explained in chapter four. The following phases are applied to EMS prototype development and execution with the modified established process guide.

When compared with DSEEP earlier, the framework reported three main development phases: planning, development, and experimentation. The ***planning phase*** is the same for almost all approaches, including stand-alone, LAN-based and Cloud-Based DS. The distributed simulation project planning stage involves problem definition by analysts and whether DS is a suitable analysis method. This part is used to evaluate the RQ1, which enquires about deploying DS on the cloud. The implementation and running of the EMS using the proposed architecture expose the components needed to deploy CBDS. To define the problem, the objective must be clearly specified by the client which can be internal or external to the organisation responsible for the study's execution Ülgen, Johnsonbaugh and Klungle (2000) and BK and Ezhil (2019).

The CBDS ***development phase*** slightly differs from LAN-Based DS and is hugely distinct from stand-alone simulation projects. In DS, federates are built from scratch or modified (existing models) to be able to run independently and have the interoperability to be linked together and exchange data in a federation. One of the fundamental differences is in the communication mode between the LAN and Cloud environments. In local networks, DS federates can exchange traffic using the best-effort multicast communication mode. The RTI will relay them from the source to intended destinations. For cloud-based DS, the federation must have a router or a gateway to direct inbound and outgoing TCP packets and UDPs. This calls for having to deal with IP addresses and port numbers at each level of communication during federation execution. This phase contributes to RQ2 and RQ3. The first aims to uncover the factors affecting interoperability, and the latter inquire about CBDS execution speed. Communication and the internet overhead are the factors affecting the overall simulation execution performance. During the validation and verification stage of development, IP address and port numbers were checked in each federate. The middleware implementation can send and receive traffic using the configured addresses. Network and security settings for

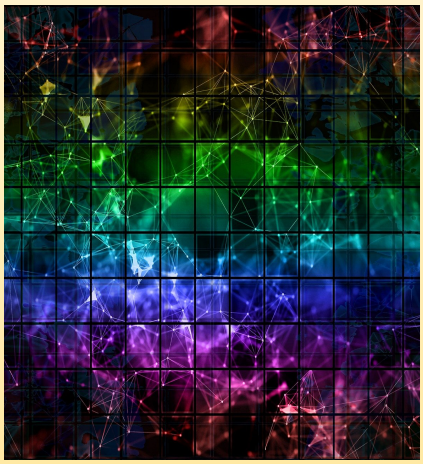
federates residing on different cloud platforms may vary based on the models' complexity. They may also vary on how they are linked together to form a more extensive cloud-based DS simulation. For example, Amazon EC2 and CloudSigma.

This thesis's proposed ***execution phase*** also introduced a few extra steps where federates configuration and federation settings are passed as a parameter in a predefined launch script. The popular Ansible Playbook is used to submit a job to multiple cloud platforms selected by the analyst. In this thesis, DICE implementation has three options, as explained in previous chapters. Traffic can traverse through a cloud-based router or configure on a local node for security and other project design objectives. Because federates on the Internet communicates, connection reliability is a component to consider and factor into the environment design.

Overall, the framework provides an alternative to developing simple and complex CBDS to study large operational systems in health, manufacturing, engineering, and military domains. Indeed, the proposed EMS has been pre-developed using DSEEP recommended practice, and in this work, DSEEP has been extended to implement CBDS. This study has been performed using the framework to investigate the performance and scalability of the cloud. The hybrid model adapted to the framework was used to experiment with different scenarios on single and multiple cloud infrastructure. These runs have shown that both the framework and the deployment architecture are feasible and presents opportunities for more research work. This is the first and is open to extension and improvements by research communities and industry to the author's knowledge.

6.7 Chapter Recap

This chapter began with an overview of the research problem and the key findings from the experimentation. It has presented an in-depth discussion of the results shown in the previous chapter. Moreover, the result implication and how that contributed to the existing knowledge was explained. Finally, the proposed development and deployment architecture was evaluated and compared with its initial design objective to answer the research questions (RQ1, RQ2, and RQ3). The next chapter is the last one, which concludes the research, highlights issues found in the process, and gives potential future work on DICE.



CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

Chapter 7 Conclusions and Future Work

7.1 Chapter Overview

Chapter six reviewed the research problem, discussed the experimentation results, presented the findings and the evaluation discussion of the proposed framework and architecture.

Chapter seven concludes the research journey. It has subsections that give a summary of the entire research—then followed by a part dedicated to showing how the research questions are addressed throughout the previous chapters—furthermore, this part recaps the research contribution, limitations, and possible future work area. A reflections section is added to narrate the author's experience throughout the project's lifetime exceeding three years due to the rippling effects of the mysterious COVID-19 pandemic, which took the world by surprise.

7.2 Summary of the Thesis

The research community is continually searching for ways and means to improve large-scale systems using suitable technology. M&S is a tool that allows them to analyse systems behaviour over time, analyse the result, and present options to organisational management for decision-making. However, modellers are confronted with the discouraging challenge of using DS as an analysis tool of choice for reasons such as it being too technical and steep learning curves. This thesis is conceived and prepared to take on the challenge of investigating how to make it adaptable even by non-technical analysts.

The author begins by asking; *why the need for cloud-based distributed simulation in operational research?* After reviewing related literature in the field, the answer indicates *evolution*, *advancements in technology,* and by extension, *economics*. Researchers uses M&S as a promising system analysis method, and today, organisations are becoming more sophisticated with complex horizontal integrations. This means a conventional M&S may not suit large-scale system simulation analysis. The research community introduced DS, which provides an alternative to the traditional computer simulation approach where single analyst, runs a single simulation model on a single processor. Unfortunately, DS comes with its challenge, that require interoperability amongst interoperating models, which may be developed with different simulation technique and significant amount of computing resources - which equals money, and additionally, it is complex and cumbersome. As information and communications technology advances, the cloud computing concept developed, and it

provides an enticing alternative to traditional DS. Cloud computing offer pay-as-you-go network access to configurable computing resources where users pay for what they use or rent for a certain period of time. This was a relief to the huge investment usually required for conventional DS.

Up to the time of writing this project report, there is no known methodology, framework, or guide on how to design, build, and deploy cloud-based DS. This was identified in the literature and therefore becomes a motivating factor to investigate and propose a feasible solution at least to M&S communities in the first instance. An architecture is designed and proposed in the research work – A **DI**stributed Simulation **C**loud Architecture for **E**xperimentation (**DICE**). A CBDS methodology was designed. A prototype EMS model was used to test and evaluate the proposal. The experimental result is encouraging. Though it did not significantly speed up experimentation, the author believes it could pave the way to upgrades and improvements through research by traditional DS and stand-alone simulations using the cloud infrastructure.

The test results showed that the cloud-based DS experiment execution time increases against the local area network (LAN)-based DS and single computer foun in the literature. In this work, the proposed technique solves some of the challenges non-technical modellers face due to software and hardware engineering involved in simulation projects. One of this thesis's contributions is the DICE, which aims to extend the distributed simulation engineering and execution process (DSEEP) concept. This means, from running DS on a local environment to the distributed cloud infrastructures. It is a step-by-step guide to the non-technical modellers in the M&S community. It will allow them to focus more on the analysis rather than the underlying technical complexities involved in the design and development processes.

7.3 Addressing the Research Questions

To bridge the literature gap, this research aims to investigate cloud-based federate development framework and multi-cloud deployment architecture for Distributed Simulation (DS). Further, research questions (RQs) were formulated to achieve the said aim: How can you deploy distributed simulation on the cloud? What are the factors affecting the interoperability of distributed simulation on the cloud? What are the factors affecting cloud-based distributed simulation experimentation speed?

A set of objectives was then decided to answer the questions, and these objectives form the main parts of this thesis. The following is a repeated list from the first chapter, section 1.6.

Objective 1: To review the literature and uncover the theoretical perspectives on cloud, distributed simulation, and the challenges of the use of modelling and simulation in operational research.

Objective 2: Identify a suitable methodology to apply to address the research questions, which will help achieve the thesis aim.

Objective 3: Design and develop a cloud-based methodological framework for distributed simulation of a large-scale system.

Objective 4: To use a prototype EMS model to implement and test the proposed architecture's feasibility.

Objective 5: To evaluate the architecture via experimentation results analysis with an in-depth discussion.

The objectives were used. They adequately address the research questions as discussed and mapped into chapters two, three, four, five, and six, respectively. Below is a summary of the answers.

RQ1 - How can you deploy distributed simulation on the cloud?

Modellers are used to following an established development framework; none exist for cloud-based. This thesis proposes both development framework and CBDS deployment architecture in chapter four sections 4.2 and 4.3, respectively. Both were used in the design, development, and testing of the prototype case study to evaluate the proposed solution.

RQ2 - What are the factors affecting the interoperability of distributed simulation on the cloud?

Cloud computing resources configuration, networking, security settings are factors identified to affect extending large-scale simulation and how the model involved can interoperate and exchange simulation information. Besides, the middleware implementation must have the facility for distributing TCP/UDP data over a WAN and the Internet.

RQ3 - What are the factors affecting cloud-based distributed simulation experimentation speed?

Authors in the -cloud-based domain are working already on this issue and have published some findings of what could affect cloud-based DS speed. For example, (Buyya *et al.*, 2009; Mehmi, Verma, & Sangal, 2017; Visti *et al.*, 2016). This has been confirmed in this thesis when analysing results. When designing CBDS, modellers

should take the communication and network protocols into account when deciding simulation execution time.

7.4 Research Contribution

The main contribution of this research to the field of M&S is **A DIstributed Simulation Cloud Architecture for Experimentation (DICE)** designed and proposed to ease the conceptualising, design, building, deployment, and execution of CBDS by non-technical analyst and by extension, other domain modellers. A prototype distributed and complex hybrid emergency medical service model was used to test its feasibility. The precise steps in the framework make it easy to follow and iterate sub-activities until the development is complete, and the experiment is successful. Until this time, this is the only framework and methodology for developing Cloud-Based Distributed Simulation (CBDS).

Furthermore, CBDS experimentation architecture is proposed and tested. Job submission to the cloud environment can be daunting even for the experienced technical user. This research produced a simple, configurable script to submit CBDS experiment jobs to single or multiple clouds through a single user terminal for easy monitoring and control. For the first time, this research has connected and run five different cloud infrastructures (CloudSigma, Amazon EC2, Scaleway, DigitalOcean, and Google Cloud Platform) to conduct CBDS experimentation. Playbook from Ansible was used to design the launch of the script in this thesis. With a dedicated experimentation management section of the proposed layered architecture, other server automation engines and or scripts can be used according to the project requirements and developer choice.

Before this work, the author could not find an authoritative definition of Cloud-Based Distributed Simulation (CBDS). It is a relatively new field of research compared to the widely studied M&S. As reported in the first chapter, section 1.2, this research has given a definitive meaning to the new cloud-based distributed simulation concept for this research and possibly future ones. The CBDS defined and used it as;

> ***A technique that enables the execution of multiple distributed simulations run across multiple, on-demand, and configurable cloud infrastructure, platforms, and software for the user to use as a service, over WAN or the Internet.***

7.5 Research Challenges

The challenges faced during this research were mostly technical. For example, when deploying the distributed simulation federation in the cloud, many commercial, public cloud providers were selected. These include Amazon EC2, CloudSigma, Microsoft Azure, IBM Cloud, Scaleway, Google Cloud Platform, and DigitalOcean. Of the list, only five have a relatively straightforward configuration at the infrastructure level. Others have a super complex setup resource and networking facilities, which may defeat the goal of this research of making it easier for the analysts. This requires the author to spend significant time studying them in detail and then design CBDS experiment to run from them.

Another obstacle is the cost attached to cloud computing resources. EMS scenarios to test the proposed architecture require an enormous amount of time to run the desired average of five runs for each scenario multiplied by two architecture deployment schemes. This comes as a financial constraint to the author as a research student.

Historically, the experimentation stage of this thesis was affected by the COVID-19 pandemic breakout. The national lockdown imposed, as a result, barred access to simulation labs for several months. This is an indirect effect on the overall research.

7.6 Research Limitations

Like many Ph.D. theses, this has its limitations, and the following are some of the shortfalls identified at the end of this research.

- The architecture is evaluated using a single prototype case study, which is the EMS model. The hybrid model has two components; the ambulance service model developed using ABS simulation paradigm. The other is the A&E department which is a process-driven DES model. A different simulation paradigm was not tested, such as System Dynamics in combination with the rest.
- The prototype model uses data for the London coverage area only, which may not provide enough ground for generalising its findings. This is a case study-related limitation.
- The literature underpinning the research is based on M&S that focuses more on non-technical modellers with no software engineering experience.
- In the current implementation of CBDS using the proposed framework and deployment architecture, if one instance is down, there is no recovery mechanism put in place to restart or prevent the failed model from affecting the federation run.

Moreover, where a dynamic IP address is used for the cloud infrastructure, the IP is lost when the instance is turned off. A new IP is assigned in most providers, which means reconfiguring the launch script and RTI middleware components to work correctly for the next experimentation.

- When configuring the virtual machines on the cloud, region and physical location are not considered, however, that did not affect the main objective of this research.
- Experiments are submitted manually with minimal automation during execution and logging.

7.7 Research Future Work

The author believes this is a novel endeavour in terms of Cloud-Based DS. Therefore, it may serve as a steppingstone to many feature enhancements and to improve upon what is presented here. Firstly, this work was carried out entirely headless (command line terminal) with no graphical user interface (GUI). Future work can bring the complexity under the hood and provide a more intuitive, user-friendly interface with a touch of modern user experience (UX) principle. Mobile cloud computing is gaining significance (Shiraz *et al.*, 2012; Amoretti, Grazioli and Zanichelli, 2015; Bahwaireth *et al.*, 2016) and bringing CBDS capability using mobile devices is an attractive feature.

For flexibility, accessibility, and reproducibility, this research uses open-source software tools - simulator, middleware, and programming language to develop the model and deploy it for testing. Interoperability with commercial packages can do excellent future research. It will attract the M&S research community to embrace the concept if they know CBDS can interoperate with commercial packages or even with legacy systems.

Lastly, all the issues raised in sections 7.5 and 7.6 above need to be addressed. A recovery mechanism for a failed experiment execution will save cost as cloud instances are "rented", and the cost adds as the clock ticks.

7.8 Reflections

There are many aspects that this thesis should have covered. For example, rather than sending input parameters through command line interface to the cloud instance, a visual interface would make a great way of understanding the simulation run and processes involved. Another thing is the deployment on multiple clouds to form a regional federation, making it a broader geographically distributed environment. However, there was a constrained financial limitation on the author's part. The time needed to explore the various cloud providers'

technical configurations and adapt the DICE prototype for multi-cloud testing was also limited. The three years is not enough for that. Unfortunately, the COVID-19 lockdown bedevils some effort to cover these missing aspects of this ambitious project.

7.9 Chapter Recap

This chapter culminates the research report for this thesis. It revisited the proposed architecture, the project's aim, how the research questions were addressed, objectives used in the process, and research contributions. The section also presents the research limitation, issues identified and possible future work to expand the proposed solution even beyond M&S domain consumption. Let us review the thesis as a whole from the beginning.

Chapter one introduces the reader to a high-level overview of the work submitted. It begins with an introduction to the research background, context, motivation, and the questions this thesis is out to address. The aims and objectives are presented as a vehicle to design, execute and complete the research. Furthermore, this chapter also gives a brief overview of the succeeding sections. Chapter two reviews the recently published research in Distributed Simulation, Distributed Simulation, and Cloud-based Simulation and identifies the gap in the literature.

Moreover, the section gives the reader history and general simulation concepts, types of modelling, world views, approaches, and experimentation. It introduces some essential aspects of CBDS; the high-level architecture (HLA), time in simulation, and time management (synchronisation). The chapter also analyses, and reports simulation methodologies related to this thesis from both on-premises and cloud infrastructures. Then relates how that relates to the M&S research communities of practice.

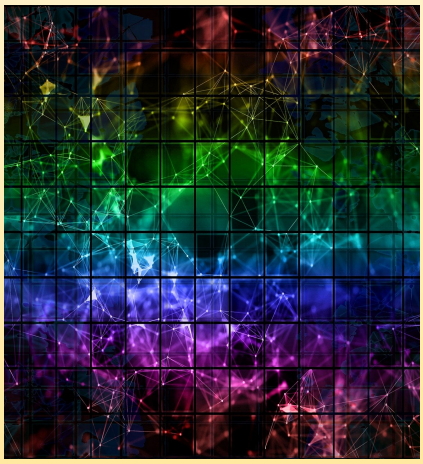
Chapter three of the thesis states the research design approach and offers possible alternatives to address the questions posed from the academic perspective. It also describes data collection and experimentation tools, methods of result analysis, and justifications for the chosen methods. The chapter explains the cloud architecture development approach taken in this work and the case study method adapted to implement and evaluate the proposed framework and architecture. Chapter four presents the proposed architecture and framework development processes and explains possible implementation schemes used in the following chapter to test, analyse results, evaluate, and validate the proposed architecture's feasibility. Chapter five presents the simulation approaches of ABS and DES, then dives into the experimentation environment setup. The setup includes the cloud infrastructure provider, computing resources, network services, experiment submission, monitoring, and result

collection procedure. The following section introduces the case study prototype – the Medical Emergency Service (EMS) and its components comprising an ambulance and hospital accident & emergency (A&E). The interactions between the ambulance, the A&E, the interoperability reference model used, and the time management are discussed in detail. Moreover, the section presents how the EMS is adapted to the proposed architecture for evaluation. In the end, the reader will find the software tools selected for the experiment based on the research design.

Chapter six revisits the research problem and presents the key findings from the result analysis. It also discusses the results in more detail. The sections are organised as to how the results relate to previous research. The findings differ from other studies, how the results and findings confirm some existing theories, and the practical implications and the contributions to the field. Finally, the chapter evaluated the DICE architecture from the research perspective.

Each chapter above is presented with sub-sections and aims to achieve the set objectives. The external examiner thoroughly reviewed the thesis and gave more than 200 comments on how to improve the thesis. This help make the arguments in work more convincing and more pleasant to read.

This page is intentionally left blank for printing purpose.



REFERENCES

BIBLIOGRAPHY

References

Abar, S., Theodoropoulos, G.K., Lemarinier, P., O'Hare, G.M.P., (2017). Agent Based Modelling And Simulation Tools: A Review Of The State-Of-Art Software. *Computer Science Review*. Vol. 24, 13–33. DOI: Https://Doi.Org/10.1016/J.COSREV.2017.03.001

Akram, A., Sarfraz, M.S., Shoaib, U., (2019). HLA Run Time Infrastructure: A Comparative Study. *Mehran University Research Journal of Engineering and Technology.* Vol. 38, 961–972. DOI: Https://Doi.Org/10.22581/Muet1982.1904.09

Aliaga, M., & Gunderson, B. (2006). Interactive Statistics. Upper Saddle River, N.J: Pearson Prentice Hall.

Alturki, A., Gable, G.G., Bandara, W., (2011). A Design Science Research Roadmap, *In: Lecture Notes In Computer Science (Including Subseries Lecture Notes In Artificial Intelligence And Lecture Notes In Bioinformatics)*. Springer, Berlin, Heidelberg, Pp. 107–123. DOI: Https://Doi.Org/10.1007/978-3-642-20633-7_8

Amazon Web Services (AWS) [WWW Document], N.D. URL Https://Aws.Amazon.Com/ (Accessed 2.13.20).

Amoretti, M., Grazioli, A., Zanichelli, F., 2015. A Modeling And Simulation Framework For Mobile Cloud Computing. *Simulation Modelling Practice and Theory.* Vol. 58, Part 2, 140-156 DOI: Https://Doi.Org/10.1016/J.Simpat.2015.05.004

Anagnostou, A., (2014). Thesis "A Distributed Simulation Methodology For Large-Scale Hybrid Modelling And Simulation Of Emergency Medical Services". *Department Of Computer Science, Brunel University London, United Kingdom.*

Anagnostou, A., Nouman, A., Taylor, S.J.E., (2013). Distributed Hybrid Agent-Based Discrete Event Emergency Medical Services Simulation, *In: Proceedings Of The 2013 Winter Simulation Conference - Simulation: Making Decisions In A Complex World*, . Pp. 1625–1636. DOI: Https://Doi.Org/10.1109/WSC.2013.6721545

Anagnostou, A., Taylor, S.J.E., (2017). A Distributed Simulation Methodological Framework

For OR/MS Applications. *Simulation Modelling Practice And Theory.* Vol. 70, 101–119. DOI: Https://Doi.Org/10.1016/J.Simpat.2016.10.007

Anagnostou, A., Taylor, S.J.E., Tijjani Abubakar, N., Kiss, T., Deslauriers, J., Gesmier, G., Terstyanszky, G., Kacsuk, P., Kovacs, J., (2019). Towards A Deadline-Based Simulation Experimentation Framework Using Micro-Services Auto-Scaling Approach, *In: Proceedings – 2019 Winter Simulation Conference*. Institute Of Electrical And Electronics Engineers Inc., Pp. 2749–2758. DOI: Https://Doi.Org/10.1109/WSC40007.2019.9004882

Ansible IT Automation [Https://Www.Ansible.Com/], (Accessed 8.13.20).

Anu, M., (1997). Introduction To Modeling And Simulation, In: S. Andradóttir, K.J. Haaly, D.H. Withers, B.L. Nelson (Eds.), *In: Proceedings – 1997 Winter Simulation Conference*. Pp. 7–13.

Azevedo, T., Rossetti, R.J.F., Barbosa, J.G., (2015). Densifying The Sparse Cloud Simsaas: The Need Of A Synergy Among Agent-Directed Simulation, Simsaas And HLA, *In: 5th International Conference On Simulation And Modeling Methodologies, Technologies And Applications* · SIMULTECH 2015. Colmar, France. DOI: Https://Doi.Org/10.5220/0005542801720177

B.K., J., S. Ezhil, S., (2019). A Performance On Simulation With Methodologies. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*. Vol. 8:7, 1166–1170.

Eduard Babulak and Ming Wang (August 18th 2010). Discrete Event Simulation: State of the Art, Discrete Event Simulations, Aitor Goti, IntechOpen, DOI: https://doi.org/10.5772/9894

Bahwaireth, K., Tawalbeh, L., Benkhelifa, E., Jararweh, Y., Tawalbeh, M.A., (2016). Experimental Comparison Of Simulation Tools For Efficient Cloud And Mobile Cloud Computing Applications. *EURASIP Journal on Information Security* Vol. 2016, 1–14. DOI: Https://Doi.Org/10.1186/S13635-016-0039-Y

Balachandran, A., (2000). Introduction To Simulation And Modeling: Historical Perspective [WWW Document]. Univ. Houst. URL

Http://Www.Uh.Edu/~Lcr3600/Simulation/Historical.Html (Accessed 10.13.18).

Balci, O., (2012). A Life Cycle For Modeling And Simulation. *Simulation: Principles of Advanced and Distributed Simulation (PADS)* Vol. 88(7), 870–883. DOI: Https://Doi.Org/10.1177/0037549712438469

Banks, C.M., (2010). Introduction To Modeling And Simulation, *In: Modeling And Simulation Fundamentals: Theoretical Underpinnings And Practical Domains*. John Wiley And Sons, Pp. 1–24. DOI: Https://Doi.Org/10.1002/9780470590621.Ch1

Banks, Carson, Nelson, Nicol, (2013). Introduction To Simulation. *In Proceedings Of The 2013 Winter Simulation Conference: Simulation: Making Decisions In A Complex World*. IEEE Press, 291–305.

Banks, J., Carson, J.S., Barry, I.I., Nelson, L., Nicol, D.M., (2013). Discrete-Event System Simulation, 5th Ed. Pearson (Intl).

Banks, J., Carson, J.S., Nelson, B.L., Nicol, D.M., (2000). Discrete-Event System Simulation (3rd Edition), 3rd Ed. Prentice Hall.

Barbosa, F.P., Charão, A.S., (2012). Impact Of Pay-As-You-Go Cloud Platforms On Software Pricing And Development: A Review And Case Study, *In: Lecture Notes In Computer Science (Including Subseries Lecture Notes In Artificial Intelligence And Lecture Notes In Bioinformatics)*. Springer, Berlin, Heidelberg, Pp. 404–417. DOI: Https://Doi.Org/10.1007/978-3-642-31128-4_30

Barrett, D., Kipper, G., (2010). How Virtualization Happens, *In: Virtualization And Forensics.* Elsevier, Pp. 3–24. DOI: Https://Doi.Org/10.1016/B978-1-59749-557-8.00001-1

Barton, R.R., (2004). Designing Simulation Experiments, *In: Proceedings Of The 2004 Winter Simulation Conference*. IEEE, Pp. 69–75. DOI: Https://Doi.Org/10.1109/WSC.2004.1371304

Baxter, P., Jack, S.J., (2008). Qualitative Case Study Methodology: Study Design And Implementation For Novice Researchers. *The Qualitative Report 13* (2008): 544-559.

Block, J., (2018). Hybrid Agent-Based Modeling (HABM)—A Framework For Combining

Agent-Based Modeling And Simulation, Discrete Event Simulation, And System Dynamics. Springer, Cham, Pp. 603–608. DOI: Https://Doi.Org/10.1007/978-3-319-89920-6_80

Khanzode V.V., (1995), Research Methodology: Technique & Trends, New Delhi: APH Publishing Corporation, p. 35 24. Ibid 25. Borwankar, op.cit., p.46

Brailsford, S.C., Eldabi, T., Kunc, M., Mustafee, N., Osorio, A.F., 2019. Hybrid Simulation Modelling In Operational Research: A State-Of-The-Art Review. *European Journal of Operational Research.* Vol. 278(3), 721-737 DOI: Https://Doi.Org/10.1016/J.Ejor.2018.10.025

Brito, A. V., Costa, L.F.S., Bucher, H., Sander, O., Becker, J., Oliveira, H., Melcher, E.U.K., (2016). A Distributed Simulation Platform Using HLA For Complex Embedded Systems Design, *In: Proceedings - 2015 IEEE/ACM 19th International Symposium On Distributed Simulation And Real Time Applications, DS-RT 2015*. Institute Of Electrical And Electronics Engineers Inc., Pp. 195–202. DOI: Https://Doi.Org/10.1109/DS-RT.2015.16

R. E. Bryant. 1977. SIMULATION OF PACKET COMMUNICATION ARCHITECTURE COMPUTER SYSTEMS. *Technical Report.* Massachusetts Institute of Technology, USA.

Bryman, A., Bell, E., (2015). Business Research Methods. Oxford University Press, Oxford.

Buora, G.B., Giusti, C., Barbina, M., (2014). Taking Advantages Of Modern Distributed Infrastructures In Modelling And Simulation, *In: Lecture Notes In Computer Science (Including Subseries Lecture Notes In Artificial Intelligence And Lecture Notes In Bioinformatics)*. Springer Verlag, Pp. 67–76. DOI: Https://Doi.Org/10.1007/978-3-319-13823-7_7

Burks, A.W., Burks, A.R., (1981). The ENIAC: First General-Purpose Electronic Computer. Annals of the History of Computing. Vol. 3(4), 310–389. DOI: Https://Doi.Org/10.1109/MAHC.1981.10043

Buyya, R., Ranjan, R., Calheiros, R.N., (2009). Modeling And Simulation Of Scalable Cloud Computing Environments And The Cloudsim Toolkit: Challenges And Opportunities, *In:*

2009 International Conference On High Performance Computing & Simulation. IEEE, Pp. 1–11. DOI: Https://Doi.Org/10.1109/HPCSIM.2009.5192685

Byrne, J., Heavey, C., Byrne, P.J., (2010). A Review Of Web-Based Simulation And Supporting Tools. Simulation Modelling Practice and Theory. Vol. 18(3), 253–276. DOI: Https://Doi.Org/10.1016/J.Simpat.2009.09.013

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R., (2011). Cloudsim: A Toolkit For Modeling And Simulation Of Cloud Computing Environments And Evaluation Of Resource Provisioning Algorithms. Software Practice and Experience. Vol. 41(1), 23–50. DOI: Https://Doi.Org/10.1002/Spe.995

Carillo, M., Cordasco, G., Serrapica, F., Scarano, V., Spagnuolo, C., Szufel, P., (2018). Distributed Simulation Optimization And Parameter Exploration Framework For The Cloud. *Simulation Modelling Practice and Theory*. Vol. 83, 108–123. DOI: Https://Doi.Org/10.1016/J.Simpat.2017.12.005

Carothers, C.D., Perumalla, K.S., (2010). On Deciding Between Conservative And Optimistic Approaches On Massively Parallel Platforms, *In: Proceedings – 2010 Winter Simulation Conference (WSC)*. Pp. 678–687. DOI: Https://Doi.Org/10.1109/WSC.2010.5679119

Carson, J.S., (1993). Modeling And Simulation Worldviews, In: G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (Eds.). *In: Proceedings of 1993 Winter Simulation Conference (WSC).* IEEE, Los Angeles, CA, USA, Pp. 18–23. DOI: Https://Doi.Org/10.1109/WSC.1993.718024

Caughlin, D., (1995). Verification, Validation, And Accreditation (VV&Amp;A) Of Models And Simulations Through Reduced Order Metamodels, *In: Proceedings* of *1995 Winter Simulation Conference Proceedings (WSC).* IEEE, Pp. 1405–1412. DOI: Https://Doi.Org/10.1109/WSC.1995.479054

Chan, V., Pegden, D., (2017). The History Of Simulation Modeling, In: W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, E. Page (Eds.). *In: Proceedings of 2017 Winter Simulation Conference (WSC)*. IEEE Press Piscataway, NJ, USA, Las Vegas, Nevada, P. 18.

Chan, K. V, Moon, I., K Roeder, T.M., Macal, C., Rossetti, M.D., Robinson Gilbert Arbez

Louis Birta, S.G., Tolk Gerd Wagner, A., (2015). Conceptual Modeling: Definition, Purpose And Benefits, *In: Proceedings of 2015 Winter Simulation Conference (WSC)*. IEEE Xplore, Huntington Beach, CA, USA, Pp. 2812–2826. DOI: Https://Doi.Org/10.1109/WSC.2015.7408386

Chandy, K.M., Misra, J., (1981). Asynchronous Distributed Simulation Via A Sequence Of Parallel Computations. *Communications of the ACM.* Vol. 24(4), 198–206. DOI: Https://Doi.Org/10.1145/358598.358613

Chaudhry, N.R., Nouman, A., Anagnostou, A., Taylor, S.J.E., (2016). Investigating WS-PGRADE Workflows For Cloud-Based Distributed Simulation, *In: Proceedings – 2016 Winter Simulation Conference.* Institute Of Electrical And Electronics Engineers Inc., Pp. 3180–3181. DOI: Https://Doi.Org/10.1109/WSC.2015.7408459

Chavez, J.D.J., Ramirez, A.I., Dinavahi, V., Iravani, R., Martinez, J.A., Jatskevich, J., Chang, G.W., (2010). Interfacing Techniques For Time-Domain And Frequency-Domain Simulation Methods. IEEE Transactions on Power Delivery. Vol. 25(3), 1796-1807. DOI: Https://Doi.Org/10.1109/TPWRD.2009.2037152

Cloud Computing Services [Https://Cloud.Google.Com/] (Accessed 1.25.21).

Cloudsigma.Com [Https://Www.Cloudsigma.Com/] (Accessed 2.13.20).

Collier, N., North, M., (2013). Parallel Agent-Based Simulation With Repast For High Performance Computing. Simulation - Special Issue: Advancing Sim Theory and Practice With Distributed Computing: Part 1. Vol. 89(10), 1215–1235. DOI: Https://Doi.Org/10.1177/0037549712462620

Creswell, J., (1966). Research Design: Qualitative, Quantitative Approaches, And Mixed Methods Approaches. Thousand Oaks, Calif: Sage.

Creswell, J. (2002). Educational Research: Planning, Conducting, And Evaluating Quantitative And Qualitative Research. Upper Saddle River, NJ: Merrill Prentice Hall.

Crooks, A.T., (2007). The Repast Simulation/Modelling System For Geospatial Simulation, *In: Agent-Based Models For Spatial Systems In Social Sciences & Economic Science With Heterogeneous Interacting Agents (ABM–S4–ESHIA)*. Agelonde, La Londe Les

Maures, France.

Crum, W.R., Berry, E., Ridgway, J.P., Sivananthan, U.M., Tan, L.B., Smith, M.A., (1998). Frequency-Domain Simulation Of MR Tagging. *Journal of Magnetic Resonance Imaging.* Vol. 8(5), 1040–1050. DOI: Https://Doi.Org/10.1002/Jmri.1880080507

Cruz, R.S., Casquilho, M., (2019). Distributed Computing, *In: Proceedings - Iberian Conference On Information Systems And Technologies, CISTI*. IEEE Computer Society. DOI: Https://Doi.Org/10.23919/CISTI.2019.8760827

D'Angelo, G., (2014). Parallel And Distributed Simulation From Many Cores To The Public Cloud (Extended Version), *In: Proceedings International Conference On High Performance Computing And Simulation*. Istanbul, Turkey. DOI: Https://Doi.Org/10.1109/Hpcsim.2011.5999802

D'Angelo, G., Bracuto, M., (2009). Distributed Simulation Of Large-Scale And Detailed Models. International Journal of Simulation and Process Modelling (IJSPM). Vol. 5(2), 120–131. DOI: Https://Doi.Org/10.1504/IJSPM.2009.028625

D'Angelo, G., Marzolla, M., (2014). New Trends In Parallel And Distributed Simulation: From Many-Cores To Cloud Computing. *Simulation Modelling Practice and Theory*. Vol. 49, 320-335. DOI: Https://Doi.Org/10.1016/J.Simpat.2014.06.007

Dai, H., Lin, J., Long, Q., (2014). A Fractal Perspective-Based Methodological Framework For Supply Chain Modelling And Distributed Simulation With Multi-Agent System. *International Journal of Production Research*. Vol. 52(22), 6819–6840. DOI: Https://Doi.Org/10.1080/00207543.2014.919414

de Lara, J., Guerra, E., Boronat, A. et al. (2014). Domain-specific discrete event modelling and simulation using graph transformation. *Software and Systems Modeling.* Vol. 13, 209–238. https://doi.org/10.1007/s10270-012-0242-3

Delen, D., Demirkan, H., (2013). Data, Information And Analytics As Services. *Decision Support Systems.* Vol. 55(1), 359-363. DOI: Https://Doi.Org/10.1016/J.Dss.2012.05.044

Digitalocean – The Developer Cloud [Https://Www.Digitalocean.Com/] (Accessed 2.28.20).

Dikaiakos, M.D., Katsaros, D., Mehra, P., Pallis, G., Vakali, A., (2009). Cloud Computing: Distributed Internet Computing For IT And Scientific Research. *IEEE Internet Computing.* Vol. 13(5), 10-13. DOI: Https://Doi.Org/10.1109/MIC.2009.103

Djanatliev, A., Kolominsky-Rabas, P., Hofmann, B.M., Aisenbrey, A., German, R., (2014). System Dynamics And Agent-Based Simulation For Prospective Health Technology Assessments*, *In: Proceedings - Advances In Intelligent Systems And Computing.* Springer Verlag, Pp. 85–96. DOI: Https://Doi.Org/10.1007/978-3-319-03581-9_6

Dodig-Crnkovic, G., (2002). Scientific Methods In Computer Science. *In: Proceedings Of The Conference For The Promotion Of Research In IT At New Universities And At University Colleges In Sweden*, Skövde, Suecia

Dong, D., Xiong, H., Castañe, G.G., Morrison, J.P., (2018). Cloud Architectures And Management Approaches, *In: Proceedings - Heterogeneity, High Performance Computing, Self-Organization And The Cloud*. Springer International Publishing, Pp. 31–61. DOI: Https://Doi.Org/10.1007/978-3-319-76038-4_2

Donohue, J.M., (1994). Experimental Designs For Simulation, *In: Proceedings Of 1994 Winter Simulation Conference*. Pp. 200–206. DOI: Https://Doi.Org/10.1109/WSC.1994.717123

Dubiel, B., Tsimhoni, O., (2005). Integrating Agent Based Modeling Into A Discrete Event Simulation, *In: Proceedings – 2005 Winter Simulation Conference*. Pp. 1029–1037. DOI: Https://Doi.Org/10.1109/WSC.2005.1574355

Dunleavy, P. (2003). How To Plan, Draft, Write And Finish A Doctoral Thesis Or Dissertation. Palgrave Macmillan.

Ebert, C., Gallardo, G., Hernantes, J., Serrano, N., (2016). Devops. IEEE Software. Vol. 33(3), 94–100. DOI: Https://Doi.Org/10.1109/MS.2016.68

Eldabi, T., Irani, Z., Paul, R.J., Love, P.E.D., (2002). Quantitative And Qualitative Decision-Making Methods In Simulation Modelling. Management Decision. Vol. 40(1), 64–73. DOI: Https://Doi.Org/10.1108/00251740210413370

Endrei, M., & International Business Machines Corporation. (2004). Patterns: Service-

oriented architecture and web services. *"This edition applies to IBM WebSphere Application Server base V5.1, IBM WebSphere Application Server Network Deployment V5.0.2.4, IBM WebSphere MQ V5.3, and IBM WebSphere Studio Application Developer V5.1.1, for use with IBM AIX 5.1, Red Hat Linux Advanced Server V2.1, and Microsoft Windows 2000."*. United States: IBM Corp., International Technical Support Organization.

England NHS Statistics, (2011). Ambulance Quality Indicators Data 2011-12. London.

Falcone, A., Garro, A., Taylor, S.J.E., Anagnostou, A., 2017b. Simplifying The Development Of HLA-Based Distributed Simulations With The HLA Development Kit Software Framework (DKF), *In: Proceedings - 2017 IEEE/ACM 21st International Symposium On Distributed Simulation And Real Time Applications, DS-RT 2017*. Institute Of Electrical And Electronics Engineers Inc., Pp. 1–2. DOI: Https://Doi.Org/10.1109/DISTRA.2017.8167691

Farkas, Z., Hajnal, Á., Kacsuk, P., (2014). WS-PGRADE/Guse And Clouds, *In: Proceedings of Science Gateways For Distributed Computing Infrastructures*. Springer International Publishing, Cham, Pp. 97–109. DOI: Https://Doi.Org/10.1007/978-3-319-11268-8_7

Fayoumi, A., Arabia, S., 2011. Performance Evaluation Of A Cloud Based Load Balancer Severing Pareto Traffic. *Journal of Theoretical and Applied Information Technology*. Islamabad. Vol. 32(1), 28-34.

Ficco, M., Avolio, G., Palmieri, F., Castiglione, A., (2016). An HLA-Based Framework For Simulation Of Large-Scale Critical Systems. *Concurrency and Computation: Practice and Experiences.* Vol. 28(2), 400–419. DOI: Https://Doi.Org/10.1002/Cpe.3472

Fishman, G.S., (1973). Concepts And Methods In Discrete Event Digital Simulation [By] George S. Fishman. Wiley, New York.

Fitzsimmons, J.A., (1973). A Methodology For Emergency Ambulance Deployment. Management Science. Vol. 19(6), 627–636. DOI: Https://Doi.Org/10.1287/Mnsc.19.6.627

Fujimoto, R., (2015a). Parallel And Distributed Simulation, *In: Proceedings of 2015 Winter Simulation Conference (WSC)*. IEEE, Pp. 45–59. DOI:

Https://Doi.Org/10.1109/WSC.2015.7408152

Fujimoto, R., (2015b). Parallel And Distributed Simulation, In: L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, And M. D. Rossetti, Eds. (Ed.), *In: Proceedings of 2015 Winter Simulation Conference*. IEEE, Pp. 45–59.

Fujimoto, R.M., (2016). Research Challenges In Parallel And Distributed Simulation. ACM Transactions on Modeling and Computer Simulation. Vol. 26(4), 1–29. DOI: Https://Doi.Org/10.1145/2866577

Fujimoto, R.M., (2003). Proceedings Of The 35th Conference On Winter Simulation Driving Innovation., *In: Proceedings Of The 2003 Conference On Winter Simulation: Driving Innovation.* 2003 Winter Simulation Conference (WSC), New Orleans, Louisiana, Pp. 124–134.

Fujimoto, R.M., (2001). Parallel And Distributed Simulation Systems. *In: Proceedings Of The 2001 Winter Simulation Conference (WSC).* Atlanta, GA, Pp. 147–157.

Fujimoto, R.M., (2000). Parallel And Distributed Simulation Systems. Wiley Interscience, 2000.

Fujimoto, R.M., (1998). Time Management In The High Level Architecture. *Simulation*. Vol. 71(6), 388–400. DOI: Https://Doi.Org/10.1177/003754979807100604

Fujimoto, R.M., (1990). Parallel Discrete Event Simulation. *Communications of the ACM.* Vol. 33(10), 30–53. DOI: Https://Doi.Org/10.1145/84537.84545

Fujimoto, R.M., (1989). Performance Measurements Of Distributed Simulation Strategies. *Transactions of the Society for Computer Simulation*. Vol. 6(2), 89–132.

Fujimoto, R.M., Malik, A.W., Park, A., (2010). Parallel And Distributed Simulation In The Cloud. *SCS Modeling And Simulation Magazine, Society For Modeling And Simulation*. Vol. 3, 1-10.

Garrido, J.M., (1999). Practical Process Simulation Using Object-Oriented Techniques And C++. Artech House, Boston.

Garro, A., Falcone, A., Chaudhry, N.R., Salah, O.-A., Anagnostou, A., Taylor, S.J.E., (2015). A Prototype HLA Development Kit, *In: Proceedings Of The 3rd ACM Conference On SIGSIM-Principles Of Advanced Discrete Simulation - SIGSIM-PADS '15*. ACM Press, New York, New York, USA, Pp. 45–46. DOI: Https://Doi.Org/10.1145/2769458.2769489

Gibson, J., Rondeau, R., Eveleigh, D., Tan, Q., (2012). Benefits And Challenges Of Three Cloud Computing Service Models, *In: Proceedings of the 2012 Fourth International Conference On Computational Aspects Of Social Networks (Cason)*. IEEE, Pp. 198–205. DOI: Https://Doi.Org/10.1109/Cason.2012.6412402

Goldsman, D., Nance, R.E., Wilson, J.R., Stewart, H.M., (2009). A BRIEF HISTORY OF SIMULATION. M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, R. G. Ingalls (Eds.), *In: Proceedings of the Winter Simulation Conference.* Pp. 310–313.

Gorecki, S., Bouanan, Y., Zacharewicz, G., Ribault, Judicaël, Perry, N., Ribault, Judicael, (2018). Integrating HLA-Based Distributed Simulation For Management Science And BPMN. INCOM.

Gottdank, T., (2014). Introduction To The WS-PGRADE/Guse Science Gateway Framework, *In: Science Gateways For Distributed Computing Infrastructures*. Springer International Publishing, Cham, Pp. 19–32. DOI: Https://Doi.Org/10.1007/978-3-319-11268-8_2

Grobauer, B., Walloschek, T., Stöcker, E., (2011). Understanding Cloud Computing Vulnerabilities. *IEEE Security & Privacy*. Vol. 9(2), 50–57. DOI: Https://Doi.Org/10.1109/MSP.2010.115

Guan, S., De Grande, R.E., Boukerche, A., (2019). A Multi-Layered Scheme For Distributed Simulations On The Cloud Environment. IEEE Transactions on Cloud Computing. Vol. 7(1), 5–18. DOI: Https://Doi.Org/10.1109/TCC.2015.2453945

High-Performance Computing – HPC | Microsoft Azure [Https://Azure.Microsoft.Com/En-Gb/Solutions/High-Performance-Computing/] (Accessed 5.16.21).

High Performance Computing (HPC) | AWS [Https://Aws.Amazon.Com/Hpc/] (Accessed 5.16.21).

High Performance Computing (HPC) | Oracle United Kingdom

[Https://Www.Oracle.Com/Uk/Cloud/Hpc/] (Accessed 5.16.21).

High Performance Computing (HPC) Solutions | Google Cloud [Https://Cloud.Google.Com/Solutions/Hpc] (Accessed 5.16.21).

The Portico Project [Http://Www.Porticoproject.Org/] (Accessed 2.4.20).

Repast Suite [Https://Repast.Github.Io/Docs.Html] (Accessed 10.5.18).

Huiskamp, W., Van Den Berg, T., (2016). Federated Simulations, *In: Studies In Systems, Decision And Control*. Springer International Publishing, Pp. 109–137. DOI: Https://Doi.Org/10.1007/978-3-319-51043-9_6

Hwangbo, S., Lee, K., (2016). Cloud Services For Modeling And Simulation: A Simulation Of A Chemical Gasdiffusion In The Cloud, *In: 2016 IEEE/ACM 20th International Symposium On Distributed Simulation And Real Time Applications (DS-RT)*. IEEE, Pp. 187–188. DOI: Https://Doi.Org/10.1109/DS-RT.2016.26

IEEE, (2011). IEEE Recommended Practice For Distributed Simulation Engineering And Execution Process (DSEEP). *IEEE Std 1730-2010 (Revision IEEE Std 1516.3-2003) 2010*, 1–79. DOI: Https://Doi.Org/10.1109/IEEESTD.2011.5706287

IEEE Recommended Practice For High Level Architecture (HLA) Federation Development And Execution Process (FEDEP), (2003). *IEEE Std 1516.3-2003 0_1-32*. DOI: Https://Doi.Org/10.1109/IEEESTD.2003.94251

Irv Lustig, Brenda Dietrich, C.J. And C.D., (2010). The Analytics Journey | Analytics Magazine [WWW Document]. Anal. Mag. URL Http://Analytics-Magazine.Org/The-Analytics-Journey/ (Accessed 8.29.18).

Islam, N., Shaikh, Z.A., Sheikh, G.S., (2016). Towards Cloud Based Mobile Ad Hoc Network Simulation, *In: Proceedings of the 6th International Conference On Computing, Communications And Networking Technologies, ICCCNT 2015*. Institute Of Electrical And Electronics Engineers Inc. DOI: Https://Doi.Org/10.1109/ICCCNT.2015.7395170

J Will M, B., C Fransoo, J., (2002). Operations Management Research Methodologies Using Quantitative Modeling. *International Journal of Operations & Production Management.*

Vol. 22(2), 241–264. DOI: Https://Doi.Org/10.1108/01443570210414338

Jadeja, Y., Modi, K., (2012). Cloud Computing - Concepts, Architecture And Challenges, *In: Proceedings of the 2012 International Conference On Computing, Electronics And Electrical Technologies (ICCEET)*. Pp. 877–880. DOI: Https://Doi.Org/10.1109/ICCEET.2012.6203873

Jefferson, D., Sowizral, H.A., (1982). Fast Concurrent Simulation Using The Time Warp Mechanism: Part I, Local Control.

Jefferson, D.R., (1985). Virtual Time. ACM Transactions on Programming Languages and Systems. Vol. 7(3), 404–425. DOI: Https://Doi.Org/10.1145/3916.3988

Jeffrey S., S., Bruce, W., Jennifer, P., Nathan, D., (2007). A Proposed Open System Architecture For Modeling And Simulation (OSAMS), *In: Proceedings of the Fall Simulation Interoperability Workshop*. SISO, Florida, Pp. 616–637.

Jha, V., Bagrodia, R.L., (1994). A Unified Framework For Conservative And Optimistic Distributed Simulation, *In: Proceedings Of The Eighth Workshop On Parallel And Distributed Simulation - PADS '94*. ACM Press, New York, USA, Pp. 12–19. DOI: Https://Doi.Org/10.1145/182478.182480

Johnson, H.E., Tolk, A., (2013). Evaluating The Applicability Of Cloud Computing Enterprises In Support Of The Next Generation Of Modeling And Simulation Architectures. ANSS 2013.

Johnson, M.E., Jackman, J., 1989. Infinitesimal Perturbation Analysis: A Tool For Simulation. The Journal of the Operational Research Society. Vol. 40(3), 243-254. DOI: Https://Doi.Org/10.2307/2583338

Johnson, R.E., (1997). Frameworks = (Components + Patterns). *Communications of the ACM*. Vol. 40(10), 39–42. DOI: Https://Doi.Org/10.1145/262793.262799

Katsaliaki, K., Brailsford, S.C., (2007). Using Simulation To Improve The Blood Supply Chain, *In: Journal Of The Operational Research Society*. Palgrave Macmillan Ltd., Pp. 219–227. DOI: Https://Doi.Org/10.1057/Palgrave.Jors.2602195

Kelay, T., Chan, K.L., Ako, E., Yasin, M., Costopoulos, C., Gold, M., Kneebone, R.K., Malik, I.S., Bello, F., (2017). Distributed Simulation As A Modelling Tool For The Development Of A Simulation-Based Training Programme For Cardiovascular Specialties. Advances in Simulation. Vol. 2, 1-13. DOI: Https://Doi.Org/10.1186/S41077-017-0049-Y

Kelton, D. W., Barton, R.R., (2003). Experimental Design For Simulation, *In: Proceedings Of The 35th Conference On Winter Simulation: Driving Innovation*. 2003 Winter Simulation Conference, New Orleans, Louisiana, Pp. 59–65.

Khalid, R., Abdullah, T., Rashid, I., (2016). Performance Degradation Factors In Cloud Computing. *International Journal of Scientific & Engineering Research*. Vol. 7(11), 384–394.

Khanghahi, N., Ravanmehr, R., (2013). Cloud Computing Performance Evaluation: Issues And Challenges. *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*. Vol. 3(5), 29–41. DOI: Https://Doi.Org/10.5121/Ijccsa.2013.3503

King, D.W., Hodson, D.D., Peterson, G.L., (2017). The Role Of Simulation Frameworks In Relation To Experiments, *In: Proceedings Of The 2017 Winter Simulation Conference, WSC '17*. IEEE Press.

Kiran, D.R., (2019). Forecasting, *In: Proceedings of the Production Planning And Control*. Elsevier, Pp. 141–156. DOI: Https://Doi.Org/10.1016/B978-0-12-818364-9.00010-X

Kousalya, G., Balakrishnan, P., Pethuru Raj, C., (2017). *Workflow Modeling And Simulation Techniques*. Springer, Cham, Pp. 85–101. DOI: Https://Doi.Org/10.1007/978-3-319-56982-6_5

Kovacs, J., Kiss, T., Taylor, S.J.E., Farkas, A, Anagnostou, A., Pattison, G, Emodi, M, Kite, S., Petry, J, Snookes, G, Kacsuk, P. and Lovas, R. 2020. Industry Simulation Gateway on a Scalable Cloud. Gesing, S., Taylor, I. and Barclay, I (ed.) *12th International Workshop on Science Gateways*. On-line 10 - 11 Jun 2020 CEUR Workshop Proceedings.

Kovács, J., Kacsuk, P., Emődi, M., (2018). Deploying Docker Swarm Cluster On Hybrid Clouds Using Occopus. *Advances in Engineering Software*. Vol. 125, 136–145. DOI: Https://Doi.Org/10.1016/J.Advengsoft.2018.08.001

Kuljis, J., Paul, R.J., (2001). An Appraisal Of Web-Based Simulation: Whither We Wander? Advances in Engineering Software. Vol. 9(1-2), 37–54. DOI: Https://Doi.Org/10.1016/S0928-4869(01)00032-5

LAS Coverage, (2020). London Ambulance Service Coverage [Https://Www.Londonambulance.Nhs.Uk/About-Us/Where-We-Are/] (Accessed 8.19.20).

Law, A.M., (2015). Simulation Modeling And Analysis, FIFTH EDITION.

Law, A.M., Kelton, W.D., (1991). Simulation Modeling And Analysis, 2nd Ed. Mcgraw-Hill.

Lendermann, P., Julka, N., Gan, B.P., Chen, D., Mcginnis, L.F., Mcginnis, J.P., (2003). Distributed Supply Chain Simulation As A Decision Support Tool For The Semiconductor Industry. Simulation: Modeling and Analysis of Semiconductor Manufacturing. Vol. 79(3), 126–138. DOI: Https://Doi.Org/10.1177/0037549703255635

Leong, T.K., Ali, B.M., Prakash, V., Nordin, N.K., (2000). Prototype Of Web-Based Simulation Environment: Using CGI And Javascript, *In: Proceedings of the IEEE Region 10 Annual International Conference*. IEEE Region 10 International Conference TENCON. DOI: Https://Doi.Org/10.1109/Tencon.2000.893689

Liu, X., He, Q., Qiu, X., Chen, B., Huang, K., (2012a). Cloud-Based Computer Simulation: Towards Planting Existing Simulation Software Into The Cloud. Simulation Modelling Practice and Theory. Vol. 26, 135–150. DOI: Https://Doi.Org/10.1016/J.Simpat.2012.05.001

Liu, X., Qiu, X., Chen, B., Huang, K., (2012b). Cloud-Based Simulation: The State-Of-The-Art Computer Simulation Paradigm, *In: Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop On Principles Of Advanced And Distributed Simulation*. IEEE, Pp. 71–74. DOI: Https://Doi.Org/10.1109/PADS.2012.11

Liu, Z., Zou, H., Ye, W., (2015). Simulation Runner: A Cloud-Based Parallel And Distributed HPC Platform, *In: Proceedings - 2015 IEEE 8th International Conference On Cloud Computing, CLOUD 2015*. Institute Of Electrical And Electronics Engineers Inc., Pp. 885–892. DOI: Https://Doi.Org/10.1109/CLOUD.2015.121

Lubicz, M., Mielczarek, B., (1987). Simulation Modelling Of Emergency Medical Services. European Journal of Operational Research. Vol. 29(2), 178–185. DOI: Https://Doi.Org/10.1016/0377-2217(87)90107-X

Lucas, T.W., Kelton, W.D., Sánchez, P.J., Sanchez, S.M., Anderson, B.L., (2015). Changing The Paradigm: Simulation, Now A Method Of First Resort. Naval Research Logistics (NRL). Vol. 62(4), 293–303. DOI: Https://Doi.Org/10.1002/Nav.21628

Luo, J., Hong, L.J., Nelson, B.L., Wu, Y., (2015). Fully Sequential Procedures For Large-Scale Ranking-And-Selection Problems In Parallel Computing Environments. *Operations Research*. Vol. 63(5), 1177–1194. DOI: Https://Doi.Org/10.1287/Opre.2015.1413

Macal, C.M., North, M.J., (2011). Introductory Tutorial: Agent-Based Modeling And Simulation, *In: Proceedings og the 2011 Winter Simulations Conference (WSC)*.

Macal, C.M., North, M.J., (2010). Tutorial On Agent-Based Modelling And Simulation. *Journal of Simulation*. Vol. 4(3), 151–162. DOI: Https://Doi.Org/10.1057/Jos.2010.3

Macal, C.M., North, M.J., (2006). Tutorial On Agent-Based Modeling And Simulation Part 2: How To Model With Agents, *In: Proceedings – 2006 Winter Simulation Conference (WSC)*. Pp. 73–83. DOI: Https://Doi.Org/10.1109/WSC.2006.323040

MAK RTI - VT MAK [Https://Www.Mak.Com/Products/Link/Mak-Rti] (Accessed 2.4.20).

Mani Chandy, K., Misra, J., (1979). Distributed Simulation: A Case Study In Design And Verification Of Distributed Programs. IEEE Transactions on Software Engineering. Vol. SE-5(5), 440–452. DOI: Https://Doi.Org/10.1109/TSE.1979.230182

Masek, P., Stusek, M., Krejci, J., Zeman, K., Pokorny, J., Kudlacek, M., (2018). Unleashing Full Potential Of Ansible Framework: University Labs Administration, *In: Proceedings of Conference Of Open Innovation Association, FRUCT*. IEEE Computer Society, Pp. 144–150. DOI: Https://Doi.Org/10.23919/FRUCT.2018.8468270

Mccall, M., Murray, B., (2010). IEEE 1278 Distributed Interactive Simulation (DIS). Application Protocols," *In IEEE P1278.1/D15, April 2010*, Vol., No., Pp.1-707, 11 Aug.

2010.

Medel, V., Arronategui, U., Bañares, J.Á., Colom, J.M., (2017). Distributed Simulation Of Complex And Scalable Systems: From Models To The Cloud, *In: Lecture Notes In Computer Science (Including Subseries Lecture Notes In Artificial Intelligence And Lecture Notes In Bioinformatics)*. Springer Verlag, Pp. 304–318. DOI: Https://Doi.Org/10.1007/978-3-319-61920-0_22

Mehmi, S., Verma, H.K., Sangal, A.L., (2017). Simulation Modeling Of Cloud Computing For Smart Grid Using Cloudsim. Journal of Electrical Systems and Information Technology. Vol. 4(1), 159–172. DOI: Https://Doi.Org/10.1016/J.JESIT.2016.10.004

Mei, Y., Liu, L., Pu, X., Sivathanu, S., Dong, X., (2013). Performance Analysis Of Network I/O Workloads In Virtualized Data Centers. IEEE Transactions on Services Computing. Vol. 6(1), 48–63. DOI: Https://Doi.Org/10.1109/TSC.2011.36

Mell, P., Grance, T., (2011). The NIST Definition Of Cloud Computing Recommendations Of The National Institute Of Standards And Technology. *Gaithersburg*. DOI: Https://Doi.Org/10.6028/NIST.SP.800-145

Menascé, D.A., Ngo, P., (2009). Understanding Cloud Computing: Experimentation And Capacity Planning. *In: Proceedings of the 2009 Computer Measurement Group Conference*, Dallas, TX, Dec. 7-11, 2009.

Mihai, D.C., Valentin, C., Legrand, I.C., (2011). Simulation Framework For Modeling Large-Scale Distributed Systems. *International Conference On Control Systems And Computer Science (CSCS-14)*, Ed. Politehnica Press, Bucharest, Romania, Pp. 145-149

Miller, J.A., Fishwick, P.A., Taylor, S.J.E., Benjamin, P., Szymanski, B., (2001). Research And Commercial Opportunities In Web-Based Simulation. Simul. Simulation Practice and Theory. Vol. 9(1-2), 55–72. DOI: Https://Doi.Org/10.1016/S0928-4869(01)00035-0

Minson, R., Theodoropoulos, G.K., (2008). Distributing Repast Agent-Based Simulations With HLA. Concurrency and Computation: Practice and Experience. Vol. 20(10), 1225–1256. DOI: Https://Doi.Org/10.1002/Cpe.1280

Mishra, D.S.B., Alok, D.S., (2017). Handbook Of Research Methodology. RZ 94, Sector - 6, Dwarka, New Delhi - 110075 Shubham Vihar, Mangla, Bilaspur, Chhattisgarh - 495001

Misra, J., (1986). Distributed Discrete-Event Simulation. ACM Computing Surveys. Vol. 18(1), 39–65. DOI: Https://Doi.Org/10.1145/6462.6485

Mistry, K.K., Lazaridis, P.I., Zaharis, Z.D., Akinsolu, M.O., Liu, B., Xenos, T.D., Glover, I.A., Prasad, R., (2019). Time And Frequency Domain Simulation, Measurement And Optimization Of Log-Periodic Antennas. *Wireless Personal Communications*. Vol. 107, 771–783. DOI: Https://Doi.Org/10.1007/S11277-019-06299-W

Mohannad, E., Ayash, M., (2013). Research Methodologies In Computer Science And Information Systems. *Computer Science*. 2014;2014:1-4.

Möller, B., Garro, A., Falcone, A., Crues, E.Z., Dexter, D.E., (2016). Promoting A-Priori Interoperability Of HLA-Based Simulations In The Space Domain: The SISO Space Reference FOM Initiative, *In: 2016 IEEE/ACM 20th International Symposium On Distributed Simulation And Real Time Applications (DS-RT)*. IEEE, Pp. 100–107. DOI: Https://Doi.Org/10.1109/DS-RT.2016.15

Morgan, G., Smircich, L., (1980). The Case For Qualitative Research. The Academy of Management Review. Vol. 5(4), 491. DOI: Https://Doi.Org/10.2307/257453

Muijs, D., (2011). Doing Quantitative Research In Education With SPSS. (2nd Ed.). SAGE Publications Ltd

Müller-Bloch, C., Kranz, J., (2015). A Framework For Rigorously Identifying Research Gaps In Qualitative Literature Reviews. *In: proceedings of the 36th International Conference on Information Systems (2015). 1-19.*

Muller, D., (2011). Automod[tm] - Providing Simulation Solutions For Over 25 Years, *In: Proceedings – 2011 Winter Simulation Conference (WSC)*. Pp. 39–51. DOI: Https://Doi.Org/10.1109/WSC.2011.6147738

Mustapha, K., Tranvouez, E., Espinasse, B., Ferrarini, A., (2010). An Organization-Oriented Methodological Framework For Agent-Based Supply Chain Simulation, *In: 2010 4th International Conference On Research Challenges In Information Science -*

Proceedings, RCIS 2010. IEEE Computer Society, Pp. 353–364. DOI: Https://Doi.Org/10.1109/Rcis.2010.5507395

Nance, R., (1987). The Conical Methodology: A Framework For Simulation Model Development. Annals of Operations Research. Vol. 53, 1–45. DOI: Https://Doi.Org/10.1007/BF02136825

Nance, R.E., (1993). A History Of Discrete Event Simulation Programming Languages. *ACM SIGPLAN Notices*. Vol. 28(3), 149–175. DOI: Https://Doi.Org/10.1145/155360.155368

NATO, (2015). Modelling And Simulation As A Service: New Concepts And Service-Oriented Architectures.

North, M.J., Macal, C.M., (2007). Managing Business Complexity: Discovering Strategic Solutions With Agent-Based Modeling And Simulation. Oxford University Press, Oxford, U.K. DOI: Https://Doi.Org/10.1093/Acprof:Oso/9780195172119.001.0001

Noseworthy, J.R., (2011). Providing Interoperable Real-Time Data Communication With TENA, *In: Proceedings - IEEE Military Communications Conference MILCOM*. Pp. 1598–1603. DOI: Https://Doi.Org/10.1109/MILCOM.2011.6127537

Nouman, A., Anagnostou, A., Taylor, S.J.E., (2013). Developing A Distributed Agent-Based And DES Simulation Using Portico And Repast, *In: Proceedings - IEEE International Symposium On Distributed Simulation And Real-Time Applications*. IEEE Computer Society, Pp. 97–104. DOI: Https://Doi.Org/10.1109/DS-RT.2013.18

Nuñez, A., Vázquez-Poletti, J.L., Caminero, A.C., Carretero, J., Llorente, I.M., (2011). Design Of A New Cloud Computing Simulation Platform, *In: Lecture Notes In Computer Science (Including Subseries Lecture Notes In Artificial Intelligence And Lecture Notes In Bioinformatics)*. Springer, Berlin, Heidelberg, Pp. 582–593. DOI: Https://Doi.Org/10.1007/978-3-642-21931-3_45

Núñez, A., Vázquez-Poletti, Jose L, Caminero, Agustin C, Castañé, Gabriel G, Carretero, Jesus, Llorente, Ignacio M, Vázquez-Poletti, J L, Llorente, I M, Caminero, A C, Castañé, G G, Carretero, J, (2012). Icancloud: A Flexible And Scalable Cloud Infrastructure Simulator. Journal of Grid Computing. Vol. 10, 185–209. DOI: Https://Doi.Org/10.1007/S10723-012-9208-5

Nutaro, J., Sarjoughian, H., (2004). Design Of Distributed Simulation Environments: A Unified System-Theoretic And Logical Processes Approach. *Simulation*. Vol. 80(11), 577–589. DOI: Https://Doi.Org/10.1177/0037549704050919

O'Keefe, R.M., (2016). Experimental Behavioural Research In Operational Research: What We Know And What We Might Come To Know. *European Journal of Operational Research*. Vol. 249(3), 899–907. DOI: Https://Doi.Org/10.1016/J.Ejor.2015.09.027

Ong, S.E., Foster, L.J., Mann, M., (2003). Mass Spectrometric-Based Approaches In Quantitative Proteomics. *Methods* 29(2), 124–130. DOI: Https://Doi.Org/10.1016/S1046-2023(02)00303-1

Onggo, B.S., Selviaridis, K., (2017). On The Business Models Of Cloud-Based Modelling And Simulation For Decision Support, *Working Paper*. Lancaster University, Lancaster.

Park, A., Fujimoto, R.M., Perumalla, K.S., (2004). Conservative Synchronization Of Large-Scale Network Simulations, *In: Proceedings - Workshop On Parallel And Distributed Simulation*. Pp. 153–161. DOI: Https://Doi.Org/10.1109/PADS.2004.1301296

Pedrielli, G., Sacco, M., Terkaj, W., Tolio, T., (2012). An HLA-Based Distributed Simulation For Networked Manufacturing Systems Analysis. Journal of Simulation. Vol. 6(4), 237–252. DOI: Https://Doi.Org/10.1057/Jos.2012.6

Pegden, C.D., (2010). Advanced Tutorial: Overview Of Simulation World Views, In: B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, E. Yücesan (Eds.), *Proceedings Of The 2010 Winter Simulation Conference*. IEEE, Pp. 210–215. DOI: Https://Doi.Org/10.1109/WSC.2010.5679161

Petty, M., Weisel, E., (2003). Basis For A Theory Of Semantic Composability, In: Mason SJ, Hill R (Eds.), *Proceedings Of The Spring Simulation Interoperability Workshop. Simulation Interoperability Standards Organization*, Orlando, USA.

Pidd, M., (1984). Computer Simulation In Management Science. Wiley.

Pinto, L.R., Silva, P.M.S., Young, T.P., (2015). A Generic Method To Develop Simulation Models For Ambulance Systems. Simulation Modelling Practice and Theory. Vol. 51,

170–183. DOI: Https://Doi.Org/10.1016/J.Simpat.2014.12.001

Pitch Technologies – Pitch Prti – A Certified HLA RTI [Http://Www.Pitchtechnologies.Com/Products/Prti/] (Accessed 8.29.18).

Pitt, M., Monks, T., Crowe, S., Vasilakis, C., 2016. Systems Modelling And Simulation In Health Service Design, Delivery And Decision Making. *BMJ Qual Saf*. Vol. 25, 38-45. DOI: Https://Doi.Org/10.1136/Bmjqs-2015-004430

Portico History [Http://Portico.Openlvc.Org/Index.Php?Title=Portico_History] (Accessed 8.13.20).

Portico Over A WAN [://Timpokorny.Github.Io/Public/Documentation/User/Wan.Html] (Accessed 8.13.20).

Prochazka, D., Hodicky, J., (2017). Modelling And Simulation As A Service And Concept Development And Experimentation, *In: ICMT 2017 - 6th International Conference On Military Technologies*. Institute Of Electrical And Electronics Engineers Inc., Pp. 721–727. DOI: Https://Doi.Org/10.1109/MILTECHS.2017.7988851

Qaisar, E.J., (2012). Introduction To Cloud Computing For Developers: Key Concepts, The Players And Their Offerings, *In: 2012 IEEE TCF Information Technology Professional Conference*, TCF Pro IT 2012. DOI: Https://Doi.Org/10.1109/Tcfproit.2012.6221131

Rainey, L.B., Tolk, A., (2014). Modeling And Simulation Support For System Of Systems Engineering Applications. Wiley. DOI: Https://Doi.Org/10.1002/9781118501757

Rajaei, H., Alotaibi, F., Jamalian, S., (201. A Dististributed Simulation Platform For Cloud Computing, *In: Proceedings Of The 20th Communications & Networking Symposium, CNS '17*. Society For Computer Simulation International, San Diego, CA, USA.

Rao, D.M., Wilsey, P.A., (2000). Dynamic Component Substitution In Web-Based Simulation, *In: 2000 Winter Simulation Conference Proceedings*. Pp. 1840–1848. DOI: Https://Doi.Org/10.1109/Wsc.2000.899177

Render, B., Stair, R.M., Hanna, M.E., (2009). Quantitative Analysis For Management. Pearson Prentice Hall, Upper Saddle River, NJ.

Rice, S. V., Markowitz, H.M., Marjanski, A., Bailey, S.M., (2005). The SIMSCRIPT III Programming Language For Modular Object-Oriented Simulation, *In: Proceedings – 2005 Winter Simulation Conference.* Pp. 621–630. DOI: Https://Doi.Org/10.1109/WSC.2005.1574302

Riley, G.F., Ammar, M.H., Fujimoto, R.M., Park, A., Perumalla, K., Xu, D., (2004). A Federated Approach To Distributed Network Simulation. ACM Transactions on Modeling and Computer Simulation. Vol. 14(2), 116–148. DOI: Https://Doi.Org/10.1145/985793.985795

Rixon, A., Moglia, M., Burn, S., (2005). Bottom-Up Approaches To Building Agent-Based Models: Discussing The Need For A Platform, *In: Proceedings CABM-HEMA-SMAGET 2005 Conference: Joint Conference on Multi-Agent Modelling for Environmental Management*.

Rizvi, S.S., (2013). A Logical Process Simulation Model For Conservative Distributed Simulation Systems. *International journal of simulation modelling*. Vol. 12(2), 69–81. DOI: Https://Doi.Org/10.2507/IJSIMM12(2)1.224

Robinson, S., (2005). Distributed Simulation And Simulation Practice. *Simulation*. Vol. 81(1), 5–13. DOI: Https://Doi.Org/10.1177/0037549705052327

Robinson, S., (2001). Soft With A Hard Centre: Discrete-Event Simulation In Facilitation. Journal of the Operational Research Society. Vol. 52(8), 905–915. DOI: Https://Doi.Org/10.1057/Palgrave.Jors.2601158

Robinson, S., (1997). Simulation Model Verification And Validation, *In: Proceedings Of The 29th Conference On Winter Simulation - WSC '97*. ACM Press, New York, New York, USA, Pp. 53–59. DOI: Https://Doi.Org/10.1145/268437.268448

Robinson, S., Taylor, S.J.E., (2008). Celebrating 50 Years Of Simulation Software. Journal of Simulation. Vol. 2(3), 127. DOI: Https://Doi.Org/10.1057/Jos.2008.16

Rohrer, M.W., (2002). Automod Tutorial [Simulation Package]. *Institute Of Electrical And Electronics Engineers (IEEE)*, Pp. 170–176. DOI: Https://Doi.Org/10.1109/Wsc.2000.899713

Rossetti, M.D., Chen, Y., (2012). A Cloud Computing Architecture For Supply Chain Network Simulation, In: Proceedings – 2012 Winter Simulation Conference. C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds. DOI: Https://Doi.Org/10.1109/WSC.2012.6465196

Salama, M., Elkhatib, Y., Blair, G., (2019). Iotnetsim: A Modelling And Simulation Platform For End-To-End Iot Services And Networking, *In: UCC 2019 - Proceedings Of The 12th IEEE/ACM International Conference On Utility And Cloud Computing*. Association For Computing Machinery, Inc, New York, NY, USA, Pp. 251–261. DOI: Https://Doi.Org/10.1145/3344341.3368820

Santa-Eulalia, L.A., Halladjian, G., D'Amours, S., Frayret, J.M., (2011). Integrated Methodological Frameworks For Modeling Agent-Based Advanced Supply Chain Planning Systems: A Systematic Literature Review. Journal of Industrial Engineering and Management. Vol. 4(4), 624-668. DOI: Https://Doi.Org/10.3926/Jiem.326

Sargent, R.G., (2013). An Introduction To Verification And Validation Of Simulation Models, *In: Proceedings Of The 2013 Winter Simulation Conference - Simulation: Making Decisions In A Complex World*, WSC 2013. Pp. 321–327. DOI: Https://Doi.Org/10.1109/WSC.2013.6721430

Sarojadevi, H. (2012). Performance Testing: Methodologies and Tools. *Journal of Information Engineering and Applications*, 1(5), 53-61.

Scaleway Cloud Services [Https://Www.Scaleway.Com/En/] (Accessed 1.25.21).

Scrudder, R., Saunders, R., Möller, B., Morse, K.L., (2010). IEEE Standard For Modeling And Simulation (M\ Amp;S) High Level Architecture (HLA)-- Federate Interface Specification. *IEEE Std 1516.1-2010 (Revision IEEE Std 1516.1-2000)* 1–378. DOI: Https://Doi.Org/10.1109/IEEESTD.2010.5557728

Seda, P., Masek, P., Sedova, J., Seda, M., Krejci, J., Hosek, J., (2019). Efficient Architecture Design For Software As A Service In Cloud Environments, *In: Proceedings of the International Congress On Ultra Modern Telecommunications And Control Systems And Workshops*. IEEE Computer Society. DOI: Https://Doi.Org/10.1109/ICUMT.2018.8631237

Serna, M., Sevillano, F., Beltrán, M., Guzmán, A., (2010). Defining The Entity Transfer Interoperability Reference Model For Military Applications, *In: Proceedings of the Spring Simulation Multiconference 2010, Springsim'10*. ACM Press, New York, New York, USA, P. 1. DOI: Https://Doi.Org/10.1145/1878537.1878560

Shanthikumar, J.G., Sargent, R.G., (1983). UNIFYING VIEW OF HYBRID SIMULATION/ANALYTIC MODELS AND MODELING. Operations Research. Vol. 31(6), 1030–1052. DOI: Https://Doi.Org/10.1287/Opre.31.6.1030

Shekhar, S., Abdel-Aziz, H., Walker, M., Caglar, F., Gokhale, A., Koutsoukos, X., (2016). A Simulation As A Service Cloud Middleware. Annals of Telecommunications. Vol. 71, 93–108. DOI: Https://Doi.Org/10.1007/S12243-015-0475-6

Sheriff, P. D. (2006). Fundamentals of N-Tier: With examples in C++ and VB. NET. Place of publication not identified: PDSA.

Shiraz, M., Gani, A., Khokhar, R.H., Ahmed, E., (2012). An Extendable Simulation Framework For Modeling Application Processing Potentials Of Smart Mobile Devices For Mobile Cloud Computing, *In: Proceedings - 10th International Conference On Frontiers Of Information Technology*, FIT 2012. Pp. 331–336. DOI: Https://Doi.Org/10.1109/FIT.2012.66

Singh, N.K., Thakur, S., Chaurasiya, H., Nagdev, H., (2016). Automated Provisioning Of Application In IAAS Cloud Using Ansible Configuration Management, *In: Proceedings On 2015 1st International Conference On Next Generation Computing Technologies, NGCT 2015*. Institute Of Electrical And Electronics Engineers Inc., Pp. 81–85. DOI: Https://Doi.Org/10.1109/NGCT.2015.7375087

Sokolowski, J.A., Durak, U., Mustafee, N., Tolk, A., (2019). Summer Of Simulation : 50 Years Of Seminal Computer Simulation Research. Springer; 1st ed. 2019.

Spiga, D., Antonacci, M., Boccali, T., Costantini, A., Ciangottini, D., Donvito, G., Duma, C., Duranti, M., Formato, V., Gaido, L., Michelotto, D., Salomoni, D., Tracolli, M., (2018). DODAS: How To Effectively Exploit Heterogeneous Clouds For Scientific Computations*, In: Proceedings Of Science*. Sissa Medialab Srl, P. 024. DOI: Https://Doi.Org/10.22323/1.327.0024

St-Pierre, N.R., (2001). Invited Review. Integrating Quantitative Findings From Multiple Studies Using Mixed Model Methodology. Journal of Dairy Science. Vol. 84(4), 741-755. DOI: Https://Doi.Org/10.3168/Jds.S0022-0302(01)74530-4

Straßburger, S., (2006). Overview About The High Level Architecture For Modelling And Simulation And Recent Developments. In: Simulation News Europe Special Issue Parallel And Distributed Simulation Methods And Environments, 5-14.

Strassburger, S., Schulze, T., Fujimoto, R., (2008). Future Trends In Distributed Simulation And Distributed Virtual Environments: Results Of A Peer Study, *In: Proceedings – 2008 Winter Simulation Conference (WSC)*. Pp. 777–785. DOI: Https://Doi.Org/10.1109/WSC.2008.4736140

Su, S., Shih, C.-L., (2003). Modeling An Emergency Medical Services System Using Computer Simulation. *International Journal of Medical Informatics*. Vol. 72(1-3), 57–72. DOI: Https://Doi.Org/10.1016/J.Ijmedinf.2003.08.003

Tang, Y., Perumalla, K.S., Fujimoto, R.M., Karimabadi, H., Driscoll, J., Omelchenko, Y., (2005). Optimistic Parallel Discrete Event Simulations Of Physical Systems Using Reverse Computation, *In: Proceedings - Workshop On Principles Of Advanced And Distributed Simulation, PADS*. Pp. 26–35. DOI: Https://Doi.Org/10.1109/Pads.2005.16

Taylor, S.J.E., (2018). Distributed Simulation: State-Of-The-Art And Potential For Operational Research. *European Journal of Operational Research*. Vol. 273(1), 1–19. DOI: Https://Doi.Org/10.1016/J.Ejor.2018.04.032

Taylor, S.J.E., Anagnostou, A., (2014). Cloud Computing For Modelling & Simulation. *Operational Research Society*, Pp. 96–107.

Taylor, S. J.E., Eldabi, T., Riley, G., Paul, R.J., Pidd, M., (2009). Simulation Modelling Is 50 Do We Need A Reality Check? *Journal of the Operational Research Society*. Vol. 60(1), 69-82. DOI: Https://Doi.Org/10.1057/Jors.2008.196

Taylor, S.J.E., Kiss, T., Anagnostou, A., Terstyanszky, G., Kacsuk, P., Costes, J., Fantini, N., (2018). The Cloudsme Simulation Platform And Its Applications: A Generic Multi-Cloud Platform For Developing And Executing Commercial Cloud-Based Simulations.

Future Generation Computer Systems. Vol. 88, 524–539. DOI: Https://Doi.Org/10.1016/J.Future.2018.06.006

Taylor, Simon J. E., Mustafee, N., Turner, S.J., Pan, K., Strassburger, S., (2009). Commercial-Off-The-Shelf Simulation Package Interoperability: Issues And Futures, *In: Proceedings Of The 2009 Winter Simulation Conference (WSC)*. IEEE, Pp. 203–215. DOI: Https://Doi.Org/10.1109/WSC.2009.5429326

Taylor, S.J.E., Revagar, N., Chambers, J., Yero, M., Anagnostou, A., Nouman, A., Chaudhry, N.R., Elfrey, P.R., (2014). Simulation Exploration Experience: A Distributed Hybrid Simulation Of A Lunar Mining Operation, *In: Proceedings - IEEE International Symposium On Distributed Simulation And Real-Time Applications, DS-RT*. Institute Of Electrical And Electronics Engineers Inc., Pp. 107–112. DOI: Https://Doi.Org/10.1109/DS-RT.2014.21

Taylor, S.J.E., Turner, S.J., Strassburger, S., Mustafee, N., (2012). Bridging The Gap: A Standards-Based Approach To OR/MS Distributed Simulation. *ACM Transactions on Modeling and Computer Simulation*. Vol. 22(4), 1-23. DOI: Https://Doi.Org/10.1145/2379810.2379811

Teo, Y.M., Tay, S.C., (1996). Performance Analysis Of Parallel Simulation On Distributed Systems. Distributed Systems Engineering. Vol. 3(1), 20–31. DOI: Https://Doi.Org/10.1088/0967-1846/3/1/004

Tobias, R., Hofmann, C., (2004). Evaluation Of Free Java-Libraries For Social-Scientific Agent Based Simulation. Journal of Artificial Societies and Social Simulation, 7(1), 1-29. DOI: Https://Doi.Org/10.5167/Uzh-115438

Tocher, K.D., (1963). The Art Of Simulation, Electrical Engineering Series. English Universities Press.

Tolk, A., Muguira, J.A., (2003). The Levels Of Conceptual Interoperability Model, *In: Proceedings of the Fall Simulation Interoperability Workshop 2003*. Orlando, Florida.

Topçu, O., Durak, U., Oğuztüzün, H., Yilmaz, L., (2016). Distributed Simulation : A Model Driven Engineering Approach. Springer; 1st ed. 2016.

Tsai, W.-T., Li, W., Sarjoughian, H., Shao, Q., (2011). Simsaas: Simulation Software-As-A-Service, *In: Proceedings Of The 44th Annual Simulation Symposium, ANSS '11.*

Society For Computer Simulation International, San Diego, CA, USA, Pp. 77–86.

Tu, Z., Zacharewicz, G., Chen, D., (2011). Developing A Web-Enabled HLA Federate Based On Portico RTI, *In: Proceedings – 2011 Winter Simulation Conference*. Pp. 2289–2301. DOI: Https://Doi.Org/10.1109/WSC.2011.6147940

Turner Katherine Morse, S.J., Bailey, G., Beeker, E., Lightfoot Saker Solutions Malcolm Low, J., Mccall, J., Rabe Fraunhofer IPK Frank Riddick, M., Saville Nohorizon Marco Schumann Fraunhofer IFF Steffen Strassburger, J., Sturrock Simio LLC Simon Jetaylor, D., Turner, S., Waller, A., (2010). The SISO CSPI PDG Standard For COTS Simulation Package Interoperability Reference Models. *In: Proceedings of the 2008 Summer Computer Simulation Conference (SCSC)*. 1-10.

Ubuntu 18.04.4 LTS (Bionic Beaver) [Http://Releases.Ubuntu.Com/18.04/] (Accessed 2.17.20).

Ülgen, O., Johnsonbaugh, B., Klungle, R., (2000). Simulation Methodology -- A Practitioner's Perspective Michigan Simulation User Group Technical Committee On Simulation Methodology.

Vee, V.-Y., Hsu, W.-J., (1999). Parallel Discrete Event Simulation: *A Survey*. Singapore.

Viana, J., Brailsford, S.C., Harindra, V., Harper, P.R., (2014). Combining Discrete-Event Simulation And System Dynamics In A Healthcare Setting: A Composite Model For Chlamydia Infection. *European Journal of Operational Research*. Vol. 237(1), 196–206. DOI: Https://Doi.Org/10.1016/J.Ejor.2014.02.052

Visti, H., Kiss, T., Terstyanszky, G., Gesmier, G. and Winter, S. 2016. MiCADO – Towards a Microservice-based Cloud Application-level Dynamic Orchestrator. Gesing, S. and Krüger, J. (ed.) *8th International Workshop on Science Gateways, IWSG 2016*. Rome, Italy 08 - 10 Jun 2016 CEUR Workshop Proceedings.

Wang, H., Zheng, Y., Zhao, M., (2013). A Framework For Integrating Discrete Event Simulation With Agent-Based Modeling, *In: Proceedings Of 2013 6th International Conference On Information Management, Innovation Management And Industrial Engineering, ICIII 2013*. Pp. 176–180. DOI: Https://Doi.Org/10.1109/ICIII.2013.6703542

Wang, S., Roshanaei, V., Aleman, D., Urbach, D., (2016). A Discrete Event Simulation Evaluation Of Distributed Operating Room Scheduling. *IIE Transactions on Healthcare*

Systems Engineering. Vol. 6(4), 236–245. DOI: Https://Doi.Org/10.1080/19488300.2016.1226994

Wang, S., Wainer, G., (2016). Modeling And Simulation As A Service Architecture For Deploying Resources In The Cloud. *International Journal of Modeling, Simulation, and Scientific Computing*. Vol. 7(1), 1-35. DOI: Https://Doi.Org/10.1142/S1793962316410026

Wang, Wenguang, Tolk, A., Wang, Weiping, (2009). The Levels Of Conceptual Interoperability Model: Applying Systems Engineering Principles To M&Amp;S, *In: Proceedings Of The 2009 Spring Simulation Multiconference, Springsim '09. Society For Computer Simulation International*, San Diego, CA, USA.

Wei, B., Knowles, A., Silva, C., Mounce, C., Enem, A., (2018). When Asteroids Attack The Moon: Design And Implementation Of An STK-Based Satellite Communication Simulation For The NASA-Led Simulation Exploration Experience, *In: Proceedings of the Advances In Intelligent Systems And Computing*. Springer Verlag, Pp. 73–79. DOI: Https://Doi.Org/10.1007/978-3-319-54978-1_10

Whitman, L., Huff, B., Palaniswamy, S., (1998). Commercial Simulation Over The Web, *In: Proceedings of the 1998 Winter Simulation Conference Proceedings*. IEEE, Pp. 335–339. DOI: Https://Doi.Org/10.1109/Wsc.1998.745005

Wilcox, P.A., Burger, A.G., Hoare, P., (2000). Advanced Distributed Simulation: A Review Of Developments And Their Implication For Data Collection And Analysis. *Simulation Practice and Theory*. Vol. 8(3-4), 201–231. DOI: Https://Doi.Org/10.1016/S0928-4869(00)00023-9

Wu, Z., Wu, H., Li, W., Zhang, X., (2007). Extending Distributed Simulation's Run-Time Infrastructure With Web Services, *In: Proceedings Of The IEEE International Conference On Automation And Logistics, ICAL 2007*. Pp. 1528–1532. DOI: Https://Doi.Org/10.1109/ICAL.2007.4338814

Www.Cloudbroker.Com [Http://Cloudbroker.Com/Platform/] (Accessed 7.18.20).

Xiaoguang Wang, Turner, S.J., Malcolm Yoke Hean Low, Boon Ping Gan, (2004). Optimistic Synchronization In HLA Based Distributed Simulation, *In: Proceedings of the 18th Workshop On Parallel And Distributed Simulation, 2004. PADS 2004*. IEEE, Pp. 123–

130. DOI: Https://Doi.Org/10.1109/PADS.2004.1301293

Xu, F., Liu, F., Jin, H., Vasilakos, A. V., (2014). Managing Performance Overhead Of Virtual Machines In Cloud Computing: A Survey, State Of The Art, And Future Directions. Proceedings of the IEEE. Vol. 102(1), 11–31. DOI: Https://Doi.Org/10.1109/JPROC.2013.2287711

Xu, S., Mcginnis, L.F., (2006). Optimistic-Conservative Synchronization In Distributed Factory Simulation, *In: Proceedings – 2006 Winter Simulation Conference (WSC)*. Pp. 1069–1074. DOI: Https://Doi.Org/10.1109/WSC.2006.323196

Yang, W., Su, Q., Huang, S.H., Wang, Q., Zhu, Y., Zhou, M., (2019). Simulation Modeling And Optimization For Ambulance Allocation Considering Spatiotemporal Stochastic Demand. *Journal of Management Science and Engineering*. Vol. 4(4), 252–265. DOI: Https://Doi.Org/10.1016/J.Jmse.2020.01.004

Yin, R.K., (2014). Case Study Research : Design And Methods. (5th Ed.). Thousand Oaks, CA: Sage. (ISBN 978-1-4522-4256-9) .

Yin, R.K., Campbell, D.T., (2018). Case Study Research And Applications : Design And Methods. Thousand Oaks: Sage Publications.

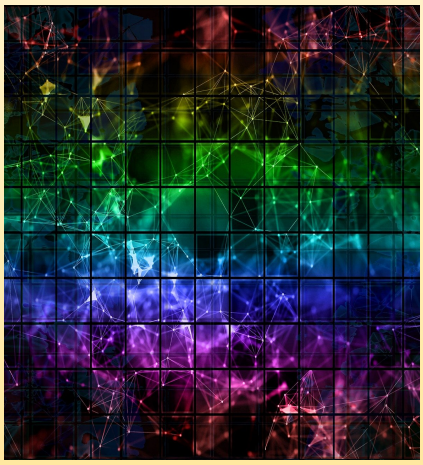
Yücesan, E., Luo, Y.C., Chen, C.H., Lee, I., (2001). Distributed Web-Based Simulation Experiments For Optimization. *Simulation Practice and Theory*. Vol. 9(-2), 73–90. DOI: Https://Doi.Org/10.1016/S0928-4869(01)00037-4

Yvonne Feilzer, M., (2010). Doing Mixed Methods Research Pragmatically: Implications For The Rediscovery Of Pragmatism As A Research Paradigm. Journal of Mixed Methods Research. Vol. 4(1), 6–16. DOI: Https://Doi.Org/10.1177/1558689809349691

Zainal, Z. (2017). Case Study As a Research Method. *Jurnal Kemanusiaan*, 5(1). 1–6.

Zhang, L., Wang, F., Li, F., (2019). Cloud-Based Simulation. Springer, Cham, Pp. 97–115. DOI: Https://Doi.Org/10.1007/978-3-030-17164-3_6

Zhen, L., Wang, K., Hu, H., Chang, D., (2014). A Simulation Optimization Framework For Ambulance Deployment And Relocation Problems. Computers & Industrial Engineering. Vol. 72, 12–23. DOI: Https://Doi.Org/10.1016/J.Cie.2014.03.008



APPENDICES

APPENDIX 1
APPENDIX 2
APPENDIX 3

Appendices

Appendix 1: EMS Case Study Prototype Model

Components

The ambulance federate (Anagnostou, 2014) has emergency vehicles with paramedic crews and emergency call centres. Call operators at these centres respond to distress calls, assess the severity of the incident, use the information to find the available vehicle closest to the site, and send with the appropriate crew. When the paramedics arrive at the scene, they give the necessary on-site treatment and then decide whether the victim needs to transfer to the hospital for further medical attention or release from the scene. If the situation calls for transfer to a hospital, the crew searches to find the fastest routes to the closest available hospital capable of handling the patient's situation. The ambulance scenario illustrates many interactions with various system entities based on conditions and dynamic behaviours. The characteristics identified can be accurately captured with the ABS simulation technique. This is backed by literature publications reported in chapter two.

The Ambulance Federate Logical Flowchart

Simulation runs have a starting time and predefined ending time where the simulation collects results and stops automatically. Similarly, the ambulance federate is triggered by incoming emergency calls. The processes continue until the end of the calling procedure, as seen in Figure 5-1. The following flowchart explains the decisions and activities from call to hospital transfer of a patient.

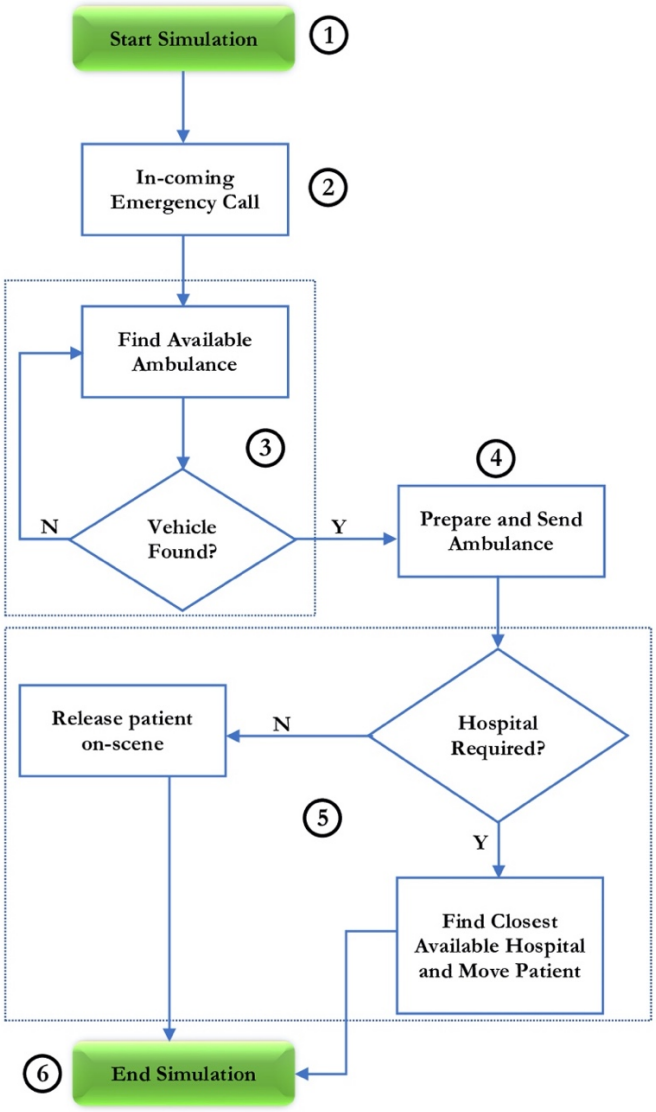


Figure 0-1 Ambulance Federate Flowchart

When the simulation begins, the EMS process starts from the top of the flowchart depicted in Figure 5-1. The numbers attached to the diagram symbols correspond with the process details, and the logical relationship is shown.

1. Simulation Start: The federation goes live and federates acquire the requirements. For example, the federation public IP address and port number are some of the information needed by each federate in the CBDS to join, interact and exchange data through the RTI.
2. Emergency Call-in: An emergency call is generated, and the ambulance model schedules the next call generation. It then generates a patient at a specific location.

3. Search Ambulance: The system then begins searching for the closest available ambulance. If an ambulance vehicle is found, the system moves to the next logical process. If not, it will wait and search again until it finds one.
4. Send Ambulance: When a suitable vehicle is found, its attribute is set to change to *Availability = FALSE*. The model then calculates the distance and travel time between the ambulance location and the incident scene and begins treating the victim.
5. Transfer to Hospital? If the paramedic attending to the patients assess the situation and see the need to be transferred to the hospital for further attention, the searching subprocess starts. Find the Closest Available Hospital: After finding the suitable hospital, the system sets the on-scene *treatment time* and calculates the *distance* and *travel time* between the scene and the chosen hospital. Before the vehicle leaves the scene, the paramedics contact the selected hospital, book the patient, and eventually reserve the required resources. The model moves the patient and the vehicle to the hospital using the *current time + treatment time + travel time* to the hospital. On a successful transfer, it calculates the distance and travel time from that hospital to the ambulance location. At this point, the ambulance goes back to its station, and its attribute change to *Availability = TRUE*. Release Patient On-Scene: If the patient is found to be fine at the scene, and no need for hospitalisation, the patient is released, and treatment time is recorded. The ambulance moves to its station at the *current time + treatment time* and its attribute change to *Availability = TRUE*.
6. The model exits the loop when either the paramedics release the patient on-scene or transfer the patient to the hospital. Finally, the vehicle goes back to the station and becomes available for the next emergency request.

As seen from the logical descriptions above, the ambulance federate makes decisions based on certain conditions which is the reason the model is designed and built with ABS paradigm to accommodate unfixed behaviours.

The A&E department federates the hospital part of the EMS model. Patients' arrival at the A&E can either be walk-in or transfer from the incident site by ambulance. In any case, the receiving reception decides treatment based on the patient's condition and the attention required. The patients go through the various treatment activities depending on the availability of the resources required for each step. Resources can be a nurse, doctor, bed, or equipment. The patients wait in a queue if the resources needed are busy. DES is used for the A&E

department based on the process-driven nature of the patient handlined characteristics. Typical DES has a series of activities with resources attached. Entities join queues and enter activity for processing only, when necessary, resources become available. They then move from the first activity to the next logical one until they exit the system model. Entities or objects in DES change states depending on the system activity. They do not change on their own as in the ABS.

The Hospital (A&E) Federate Logical Flowchart

A&E federate can run in a stand-alone mode or distributed mode in a federation. When in a stand-alone, both the walk-in and ambulance arrival are scheduled. However, in a distributed run, only the walk-in is scheduled using *the normal distribution*, the ambulance arrival is decided by the ambulance service federate. Also, from Figure 5-1, the following figure further shows patients' logical movement as they arrive from the two entry points; walk-in or brought by an ambulance.

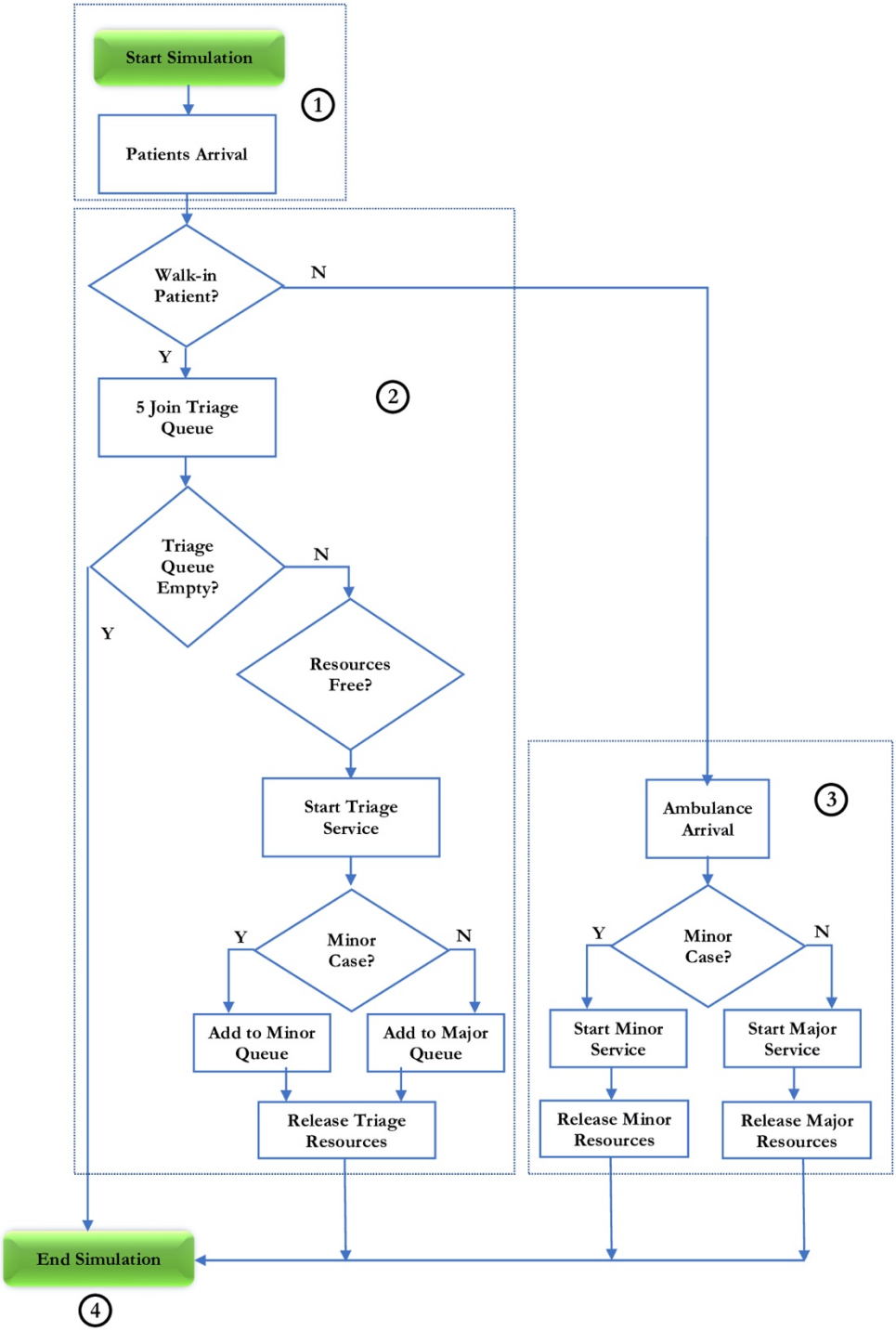


Figure 0-2 Hospital (A&E) Federate Flowchart

When the simulation begins, the EMS process starts from the top of the flowchart depicted in Figure 5-2. The numbers attached to the diagram symbols grouping corresponds to the process details, and the logical relationship is shown.

1. Simulation Start: The federation goes live and federates acquire the requirements. For example, the federation public IP address and port number

are some of the information needed by each federate in the CBDS to join, interact and exchange data through the RTI. The A&E model schedule the next walk-in arrival in the DES engine.

2. Determine Walk-in Arrival: The model checks the patient, whether it is a walk-in or ambulance arrival. If it is a walk-in patient, they are added to the triage queue. The patients wait in the queue until the required resources become available. As the resources become available, the triage service starts by seizing resources and releasing them after the service is complete. The resource availability is updated for the next patient in the queue. After the triage service, the patient exits the system or is sent further to either minors' or majors' unit queue at *the current time + triage treatment time*. After triage, the patients join majors' or minor's depending on how serious their cases are.
3. Ambulance-Arrival Patients: The Patients are sent to majors' or minors' according to their entry condition. These patients do not have to seize resources before starting the major or minor service because the ambulance has already booked and reserved them before arriving at the hospital. After the minor or major service, resources are released. Their availability is updated in the system for both the ambulance service and A&E federate. This information is then used for the next scheduled event according to the input data files' scenarios.
4. The model exits the loop when the predetermined end time is reached.

We can reasonably deduce that A&E federate uses essential components; for example, there is only one single resource type. This is used as a nurse, doctor, and lab technician. Many intermediary processes were cut-out. E.g., laboratory test, x-ray scans, optical test, etc., do not compromise, but to make the model simpler to analyse the architecture feasibility in the cloud DS environment.

Prototype Model Realisation

This part of the project takes the EMS from concept into prototype realisation. From the EMS scenarios explained in earlier sections, there are variable, local, and global, and updating them is embedded in the design stage. The prototype has two kinds of federates: ambulance (ABS) and hospital (DES). The federates should be able to run independently. They can also be linked up to form a large-scale CBDS using the proposed DICE.

Ambulance Federate

The previous chapter explains the EMS model's ambulance component where all the events, interactions with its environment, and other models in the federation were presented

at the conceptual level. Active and passive agents, space topology, were identified and included in the prototype building. An ambulance vehicle is one of the active agents that live in the federation environment. Ambulance stations and hospital locations are some of the stationary passive agents and do not carry out any autonomous activity. Both the ambulance and hospital stations have *location* and *capacity* attributes. The grid topology uses X, and Y Cartesian coordinates to define either hospital or ambulance station. The ambulance capacity shows how many ambulance vehicles are there in a given location. In contrast, hospital A&E department capacity is calculated using Equation 5-1.

$$Capacity = Availability + Occupancy$$

Equation 0-1 Hospital A&E availability

In DS, when the A&E model first joins the federation, the capacity and location data are sent to the ambulance federate. The ambulance uses the information to search for the nearest available A&E to transfer patients with severe medical cases.

A&E Department Federate

Naturally, due to a higher population, urban-based Accident and Emergency departments such as the London area are busy with a beehive of clinical activities around the clock. As explained in previous sections, the EMS model captures essential components. This research focuses on the interoperability and feasibility of the proposed DICE. In this work, there is one ambulance federate, and several A&E federates. All the A&Es are identical in every conceivable aspect – duplicated for experimentation. From the published data, freely accessible from the NHS UK, the scaled-down A&E model is designed and developed with three procedures. The first is *triage* – walk-in patients get advice only and exit the system. Walk-in patients with injuries go through triage first and then join the *minors'* or *majors'* unit queue, depending on how severe their conditions are. Patients arriving with an ambulance, skip any queue and are taken directly to minors' or majors' unit, which was reserved by paramedic even before leaving an incidence scene.

All the participating federates in the CBDS federation generate comma-separated values (.csv) and store it in the default model directory as programmed to do so. In both ambulance (ABS) and hospital (DES) federates, agents and entities ID, condition, and events timestamp are collected respectively.

EMS Model Specifications

Defining model specifications for each federate is part of the steps taken for a successful study in this research. Existing data is gathered, and it fits the theoretical distribution used during the model design stage. For example, the patient's arrival rate at an A&E, resources, number of emergency calls per given time, and so forth. The following are specifications for both the EMS sub-models and the combined summary are tabulated in Table 5-1.

The Ambulance Service Federate Specification

This federate gets input from the data published by the DoH about containing the London Ambulance Service (LAS) monthly performance measure from April 2011 to March 2012 (England NHS Statistics, 2011). In the scenario used from the data, LAS has 998 vehicles, out of which 375 are emergency ambulances. Up to writing this thesis, LAS service coverage is 620mi^2, 70 ambulance stations, five headquarters, and 32 A&E departments (LAS Coverage, 2020).

In this prototype EMS, the emergency calls were distributed equally, instead of having more calls from London crowded areas and less from less populated areas. Also, the average vehicle speed is uniform for both in and outside London's busy roads. For this simulation experiment, the following are specifications for the ambulance federate.

Emergency Call Arrival Rate	=	23.80 per hour
Coverage Area	=	150mi2
Ambulance Stations	=	14 nos
Ambulance Vehicles	=	75 nos
Travel Speed	=	15mph
Number of A&E Depts.	=	6

The Hospital A&E Federate Specification

The dataset used for the A&E departments is from the same reporting period as the ambulance service model. Similarly, the data is aggregated and distributed normally across the A&E departments in the coverage area, making all the A&Es have the same capacity (capacity based on average A&E in London), workload, and resources. Specifications for the A&E departments are as follows.

Walk-in Arrivals	=	12.60 per hour

Triage Beds	=	5
Minors Beds	=	12
Majors Beds	=	24
Clinical Staff	=	15

Table 0-1 EMS Model Data Specification and Distribution Summary

Ambulance Federate			
Inter-Arrival Time **Normal** **Distribution** Mean 2.52 SD 0.09	**Patient Condition** Minors 26% Majors 74%	**Average Speed** **Correction Factor** **Coverage Area** **Ambulance Stations** **Ambulances Per Station** **Hospitals**	15mph 1.32 150sqmi 14 9*5+5*6 6
Time On-scene (min) **Normal** **Distribution** Mean 22.52 SD 10.54	**Need Transfer to A&E** Yes 62% No 38%		
Ambulance Federate			
Walk-in Inter-Arrival Time **Normal Distribution** Mean 4.81 SD 0.59		**Patient Condition** Minors 35% Majors 65%	
Time in Triage **Normal Distribution (with staff)** mean 7.00 SD 2.00		**Need Treatment** Yes 60% No 40%	
Time in Minors **Normal Distribution (with staff)**		**Number of Staff** **Triage Capacity**	15 5

Mean 30.00 SD 10.00		**Minors Capacity** **Majors Capacity**	12 24
Time in Majors **Normal Distribution (with staff)** Mean 40.00 SD 10.00			

Experimentation Results – Table

The tables show the number of federates in each scenario, experimental run iterations in minutes, and calculated average time and standard deviation both in minutes. The following formula calculates the federates composition (first column) in each results table.

$$1^{AES} + N^{A\&E}$$

Equation 0-2 EMS federates composition formula

Where *AES (Ambulance Emergency Service)* is the ambulance service model, and *A&E* is a hospital Accident & Emergency model. Backed by the literature, the federation is incremented by two federates as the experiment progress. For example, in Table 5-3, *N = {2, 4, 6, 8, 10, 12, 14, 16}*. The first data row with 3 federates, run one ambulance model and two accidents and emergency hospital models.

Table 0-2 Experimental Results of Scheme 1 Runtime in Minutes

No. of Federates	Run 1	Run 2	Run 3	Ave. Time (minutes)	Std. Dev. (SD)
3	76	75	75	75.4	0.55
5	80	76	78	78.6	1.67
7	83	83	88	83.6	2.51
9	86	86	85	80.8	11.08
11	87	88	88	90.2	4.49
13	104	92	98	99.8	6.26
15	100	101	106	106.2	7.16
17	107	104	108	102.8	5.07

Table 5-3 shows the number of federates with corresponding average execution time in minutes under each simulation run.

Table 0-3 Scheme 2a: Multiple Clouds – Single Experiment Runs in Minutes (with cloud-based router)

No. of Federates	Run 1	Run 2	Run 3	Ave. Time (mins)	Std. Dev. (SD)
3	219	216	216	217.00	1.73
5	225	193	216	211.33	16.50
7	160	221	239	206.67	41.40
9	312	308	362	327.33	30.09
11	358	362	360	360.00	2.00
13	383	380	377	380.00	3.00
15	415	391	416	407.33	14.15
17	442	442	439	441.00	1.73

Table 0-4 Scheme 4a: Multiple Clouds – Single Experiment Run Time in Minutes (on-premises router)

No. of Federates	Run 1	Run 2	Run 3	Ave. Time (mins)	Std. Dev. (SD)
3	145	139	140	141.33	3.21
5	178	179	176	177.67	1.53
7	193	195	183	190.33	6.43

Table 0-5 Comparison of the average three scenarios of three schemes

No. of Federates	Scheme 1	Scheme 2	Scheme 3
3	75.4	217.00	141.33
5	78.6	211.33	177.67
7	83.6	206.67	190.33

Table 0-6 Average execution time comparison between schemes one and two

No. of Federates	Scheme 1	Scheme 2
3	75.4	217.00
5	78.6	211.33
7	83.6	206.67
9	80.8	327.33
11	90.2	360.00

13	99.8	380.00
15	106.2	407.33
17	102.8	441.00

Table 0-7 Standard Deviation for the Three Schemes

No. of Federates	Scheme 1	Scheme 2	Scheme 3
3	0.55	1.73	3.21
5	1.67	16.50	1.53
7	2.51	41.40	6.43
9	11.08	30.09	
11	4.49	2.00	
13	6.26	3.00	
15	7.16	14.15	
17	5.07	1.73	

Appendix 2: DICE Implementation Code Fragments

Steps: Adding New Instance (computing node)

1. Please use **`CS-EMStestKey.pem`** while creating VM on CloudSigma
2. For any VM created on any other cloud service provider add the content of **`~/.ssh/authorized_keys`** of *HOS1/HOS2* at the bottom of the file placed at the same location of the newly created VM
3. If `~/.ssh/authorized_keys` file does not exist on the newly created VM, create one with the content described in previous step and set file permission to 600 using command below
4. `$ chmod 600 ~/.ssh/authorized_keys`
5. Once the previous step is complete you should be able to login to the newly created system manually with the provided username and the IP address and the `CS-EMStestKey.pem key`
6. If manual login is successful add host info to `~/ansible/hosts` file on **EMS_master** machine. Your hosts file could look something like below,
7. `[amb_group]`
8. `amb ansible_host=212.147.209.140`

```
[hos_group]
hos1 ansible_host=212.147.209.64
hos2 ansible_host=212.147.209.13
hos3 ansible_host=212.147.209.145
hos4 ansible_host=139.59.75.11 ansible_ssh_user=root hos5
ansible_host=18.216.110.157 ansible_ssh_user=ubuntu
```

9. For test purpose you might need to manually copy and paste *MyModels/hospitalmodel/hospitalOutput_ID(<host_id>).csv* file from an old VM

WAN Gateway Configuration

```
# 5. WAN Connection Options
# ========================================

# (5.1) WAN Mode Enable/Disable
#       If true, WAN mode will be enabled and this federate will act as
both a
#       local participant, and also as a bridge for all the local
federates.
#       Messages exchanged on the local JGroups channel will be forwarded
to a
#       central router (see 5.2) to be reflected out to other sites.
Messagest
#       received from the router will be pushed out to the local JGroups
channel
#       so everyone here can process them.
#
#       Note that this mode does not support bundling. If enabled in the
RID, it
#       will be active on the local JGroups network, but ignored for the
WAN.
#       Note: Router must be running before federate startup. If not -
federates
#             will fail to start.
#       Default: false
 portico.wan.enabled = true

# (5.2) Router Address/Port
#       Specifies the address and port of the WAN router to use. Note that
the
#       syntax is "address:port".
#       Default: 127.0.0.1:23114
 portico.wan.router = 80.225.173.77:23114

# (5.2) Enable / Disable Bundling
#       Bundling enables higher throughput by grouping together a number of
#       smaller messages and sending them as one. This makes much more
efficient
#       use of the network and provides considerable improvements to
throughput
#       at a minor cost to latency.
#       If enabled, the subsequent options will control how it is applied.
#       Default: true
# portico.wan.bundle.enabled = true

# (5.3) Max Bundle Size
#       Messages sent over the WAN will be grouped into bundles and sents
as
#       a batch when their total size exceeds this value. Specify a size
with
#       a suffix of 'b', 'k' or 'm' (or 'g' if you dare!)
#       Default: 64k
# portico.wan.bundle.maxsize = 64K
```

```
# (5.4) Max Bundle Timeout
#       The maximum amount of time we will hold messages in the bundler for
while
#       waiting for more messages to arrive and bundle together. From the
time that
#       a message is received, the bundler will hold it for no longer than
this
#       value (specified in milliseconds).
#       Default: 20
# portico.wan.bundle.timeout = 20
```

Ambulance Federate (ambulance.xml)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<objectModel xsi:schemaLocation="http://standards.ieee.org/IEEE1516-
2010 http://standards.ieee.org/downloads/1516/1516.2-2010/IEEE1516-DIF-
2010.xsd" xmlns="http://standards.ieee.org/IEEE1516-2010"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<modelIdentification>
        <name>Ambulance and Hospital model</name>
        <type>FOM</type>
        <version>1.0</version>
        <modificationDate>2013-05-07</modificationDate>
        <securityClassification>Unclassified</securityClassification>
        <releaseRestriction>Other organizations not related to
Ambulance and A&amp;amp;E</releaseRestriction>
        <purpose>To define a combine module for a Ambulance and A&amp;amp;E
federation comunication</purpose>
        <applicationDomain>Ambulance operations</applicationDomain>
        <description>Comunication module for the Ambulance and
A&amp;amp;E</description>
        <useLimitation>Derived models must reference this
document</useLimitation>
        <useHistory>Originally used as an example in the HLA
IEEE1516.2-2000 specification (MIM information stripped
out)</useHistory>
        <useHistory>Used in the previous model of Ambulance and A&amp;amp;E
with HLA 1.3 implementaion</useHistory>

        <keyword>
            <taxonomy>NHS Taxonomy</taxonomy>
            <keywordValue>Ambulance</keywordValue>
        </keyword>
        <keyword>
            <taxonomy>NHS Taxonomy</taxonomy>
            <keywordValue>Hospital A&amp;amp;E</keywordValue>
        </keyword>
        <poc>
            <pocType>Supervisour </pocType>
            <pocName>Dr. Simon Taylor </pocName>
            <pocOrg>Brunel University</pocOrg>
            <pocTelephone>018-952-74000</pocTelephone>
            <pocEmail>simon.taylor@brunel.ac</pocEmail>
        </poc>
        <poc>
            <pocType>Author</pocType>
            <pocName>Mr. Athar Nouman</pocName>
```

```
            <pocOrg>Brunel University</pocOrg>
        </poc>
        <other>NA</other>
              <glyph alt="Restaurant" width="74" height="74"
type="jpg">NA</glyph>
              </modelIdentification>
    <objects>
        <objectClass>
                      <name>HLAobjectRoot</name>
                      <sharing>Neither </sharing>
 <attribute>
            <name>HLAprivilegeToDeleteObject</name>
            <dataType>HLAtoken</dataType>
            <updateType>Static</updateType>
            <updateCondition>NA</updateCondition>
            <ownership>DivestAcquire</ownership>
            <sharing>PublishSubscribe</sharing>
            <transportation>HLAreliable</transportation>
            <order>TimeStamp</order>
         </attribute>
              <objectClass>
                                    <name>Ambulance</name>
                                    <sharing>PublishSubscribe</sharing>
                                    <semantics>NA</semantics>
                                    <attribute>
                                           <name>aa</name>
                                           <dataType>HLAinteger32BE
</dataType>
                                           <updateType>Conditional
</updateType>
                                           <updateCondition>NA
</updateCondition>

        <ownership>NoTransfer</ownership>

        <sharing>PublishSubscribe</sharing>
                                           <dimensions>NA
</dimensions>
        <transportation>HLAreliable</transportation>
                                           <order>TimeStamp</order>
                                           <semantics>NA</semantics>
                                    </attribute>
                                    <attribute>
                                           <name>ab</name>
                                           <dataType>HLAinteger32BE
</dataType>
                                           <updateType>Conditional
</updateType>
                                           <updateCondition>NA
</updateCondition>
        <ownership>NoTransfer</ownership>
        <sharing>PublishSubscribe</sharing>
                                           <dimensions>NA
</dimensions>
        <transportation>HLAreliable</transportation>
                                           <order>TimeStamp</order>
                                           <semantics>NA</semantics>
                                    </attribute>
```

```
                                        <attribute>
                                                <name>ac</name>
                                                <dataType>HLAinteger32BE
</dataType>
                                                <updateType>Conditional
</updateType>
                                                <updateCondition>NA
</updateCondition>
        <ownership>NoTransfer</ownership>

        <sharing>PublishSubscribe</sharing>
                                                <dimensions>NA
</dimensions>
        <transportation>HLAreliable</transportation>
                                                <order>TimeStamp</order>
                                                <semantics>NA</semantics>
                                        </attribute>
                                        <attribute>
                                                <name>ad</name>
                                                <dataType>HLAinteger32BE
</dataType>
                                                <updateType>Conditional
</updateType>
                                                <updateCondition>NA
</updateCondition>
        <ownership>NoTransfer</ownership>

        <sharing>PublishSubscribe</sharing>
                                                <dimensions>NA
</dimensions>
        <transportation>HLAreliable</transportation>
                                                <order>TimeStamp</order>
                                                <semantics>NA</semantics>
                                        </attribute>
                                        <attribute>
                                                <name>ae</name>
                                                <dataType>HLAfloat32BE
</dataType>
                                                <updateType>Conditional
</updateType>
                                                <updateCondition>NA
</updateCondition>
        <ownership>NoTransfer</ownership>
        <sharing>PublishSubscribe</sharing>
                                                <dimensions>NA
</dimensions>
        <transportation>HLAreliable</transportation>
                                                <order>TimeStamp</order>
                                                <semantics>NA</semantics>
                                        </attribute>
            </objectClass>
         </objectClass>
    </objects>
    <interactions>
        <interactionClass>
                        <name>HLAinteractionRoot</name>
                        <sharing>PublishSubscribe</sharing>
                        <dimensions>NA</dimensions>
```

```
                    <transportation>HLAreliable</transportation>
                    <order>Receive</order>
                    <interactionClass>
                           <name>X</name>
                           <sharing>PublishSubscribe</sharing>
                           <dimensions>NA</dimensions>
    <transportation>HLAreliable</transportation>
                           <order>TimeStamp</order>
                           <semantics>NA</semantics>
                                  <parameter>
                                         <name>xa</name>
    <dataType>HLAinteger32BE</dataType>
                                         <semantics>NA</semantics>
                                  </parameter>
                                  <parameter>
                                         <name>xb</name>
    <dataType>HLAinteger32BE</dataType>
                                         <semantics>NA</semantics>
                                  </parameter>
                                  <parameter>
                                         <name>xc</name>
    <dataType>HLAinteger32BE</dataType>
                                         <semantics>NA</semantics>
                                  </parameter>
                           <interactionClass>
                                  <name>Y</name>
                                  <sharing>PublishSubscribe</sharing>
                                  <dimensions>NA</dimensions>
    <transportation>HLAreliable</transportation>
                                  <order>TimeStamp</order>
                                  <semantics>NA</semantics>
                                         <parameter>
                                                <name>ya</name>
    <dataType>HLAinteger32BE</dataType>
    <semantics>NA</semantics>
                                         </parameter>
                                         <parameter>
                                                <name>yb</name>
    <dataType>HLAinteger32BE</dataType>
    <semantics>NA</semantics>
                                         </parameter>
                                         <parameter>
                                                <name>yc</name>
    <dataType>HLAinteger32BE</dataType>
    <semantics>NA</semantics>
                                         </parameter>
                           </interactionClass>
                    </interactionClass>
             </interactionClass>
    </interactions>
<dimensions>     </dimensions>
<time>
     <timeStamp>
                    <dataType>HLAfloat64BE</dataType>
                    <semantics>Time in seconds </semantics>
             </timeStamp>
             <lookahead>
                    <dataType>HLAfloat64BE </dataType>
```

```
                    <semantics>Time in seconds </semantics>
                </lookahead>
    </time>
    <tags>
    </tags>
    <synchronizations>
        <synchronization>
                    <label>StartTest</label>
                    <dataType>NA</dataType>
                    <semantics>NA</semantics>
        </synchronization>
                <synchronization>
                    <label>EndTest</label>
                    <dataType>NA</dataType>
                    <semantics>NA </semantics>
                </synchronization>
    </synchronizations>
    <transportations>
        <transportation>
            <name>HLAreliable</name>
            <description>Provide reliable delivery of data in the sense
that TCP/IP delivers its data reliably </description>
              </transportation>
        <transportation>
            <name>HLAbestEffort</name>
            <description>Make an effort to deliver data in the sense
that UDP provides best-effort delivery </description>
              </transportation>
    </transportations>
    <switches>
              <autoProvide>Disabled </autoProvide>

<conveyRegionDesignatorSets>Disabled</conveyRegionDesignatorSets>
        <attributeScopeAdvisory>Disabled</attributeScopeAdvisory>

<attributeRelevanceAdvisory>Disabled</attributeRelevanceAdvisory>

<objectClassRelevanceAdvisory>Disabled</objectClassRelevanceAdvisory>

<interactionRelevanceAdvisory>Disabled</interactionRelevanceAdvisory>
        <serviceReporting>Disabled</serviceReporting>
       </switches>
    <dataTypes>
        <basicDataRepresentations>
            <basicData>
                <name>HLAinteger16BE</name>
                <size>16</size>
                <interpretation>Integer in the range [-2^15, 2^15 - 1]
</interpretation>
                <endian>Big</endian>
                <encoding>16-bit two's complement signed integer. The
most significant bit contains the sign.</encoding>
                    </basicData>
                    <basicData>
                <name>HLAinteger32BE</name>
                <size>32</size>
                <interpretation>Integer in the range [-2^31, 2^31 - 1]
</interpretation>
```

```
                <endian>Big</endian>
                <encoding>32-bit two's complement signed integer. The
most significant bit contains the sign.</encoding>
                      </basicData>
                      <basicData>
                <name>HLAinteger64BE</name>
                <size>64</size>
                <interpretation>Integer in the range [-2^63, 2^63 - 1]
</interpretation>
                <endian>Big</endian>
                <encoding>64-bit two's complement signed integer. The
most significant bit contains the sign.</encoding>
                      </basicData>
              <basicData>
                <name>HLAfloat32BE</name>
                <size>32</size>
                <interpretation>Single-precision floating-point number
</interpretation>
                <endian>Big</endian>
                <encoding>32-bit IEEE normalized single-precision
format (see IEEE Std. 754-1985).</encoding>
                      </basicData>
                       <basicData>
                <name>HLAfloat64BE</name>
                <size>64</size>
                <interpretation>Double-precision floating-point number
</interpretation>
                <endian>Big</endian>
                <encoding>64-bit IEEE normalized double-precision
format (see IEEE Std. 754-1985).</encoding>
                      </basicData>
                      <basicData>
                <name>HLAoctetPairBE</name>
                <size>16</size>
                <interpretation>16-bit value</interpretation>
                <endian>Big</endian>
                <encoding>Assumed to be portable among hardware
devices.</encoding>
                      </basicData>
                      <basicData>
                <name>HLAinteger16LE</name>
                <size>16</size>
                <interpretation>Integer in the range [-2^15, 2^15 -
1]</interpretation>
                <endian>Big</endian>
                <encoding>16-bit two's complement signed integer. The
most significant bit contains the sign.</encoding>
                      </basicData>
                      <basicData>
                <name>HLAinteger32LE</name>
                <size>32</size>
                <interpretation>Integer in the range [-2^31, 2^31 -
1]</interpretation>
                <endian>Little</endian>
                <encoding>32-bit two's complement signed integer. The
most significant bit contains the sign.</encoding>
                      </basicData>
              <basicData>
```

```
                <name>HLAinteger64LE</name>
                <size>64</size>
                <interpretation>Integer in the range [-2^63, 2^63 -
1]</interpretation>
                <endian>Little</endian>
                <encoding>64-bit two's complement signed integer. The
most significant bit contains the sign.</encoding>
                      </basicData>
                      <basicData>
                <name>HLAfloat32LE</name>
                <size>32</size>
                <interpretation>Single-precision floating-point
number</interpretation>
                <endian>Little</endian>
                <encoding>32-bit IEEE normalized single-precision
format (see IEEE Std. 754-1985)</encoding>
                      </basicData>
                      <basicData>
                <name>HLAfloat64LE</name>
                <size>64</size>
                <interpretation>Double-precision floating-point
number</interpretation>
                <endian>Little</endian>
                <encoding>64-bit IEEE normalized double-precision
format (see IEEE Std. 754-1985).</encoding>
                      </basicData>
                      <basicData>
                <name>HLAoctetPairLE</name>
                <size>16</size>
                <interpretation>16-bit value</interpretation>
                <endian>Little</endian>
                <encoding>Assumed to be portable among hardware
devices.</encoding>
                      </basicData>
            <basicData>
                <name>HLAoctet</name>
                <size>8</size>
                <interpretation>8-bit value</interpretation>
                <endian>Big</endian>
                <encoding>Assumed to be portable among hardware
devices.</encoding>
                      </basicData>
        </basicDataRepresentations>
        <simpleDataTypes>
            <simpleData>
                              <name>HLAASCIIchar</name>
                              <representation>HLAoctet</representation>
                              <units>NA</units>
                              <resolution>NA</resolution>
                              <accuracy>NA</accuracy>
                              <semantics>Standard ASCII character (see
ANSI Std. X3.4-1986).</semantics>
                      </simpleData>
                      <simpleData>
                              <name>HLAunicodeChar</name>
       <representation>HLAoctetPairBE</representation>
                              <units>NA</units>
                              <resolution>NA</resolution>
```

```
                            <accuracy>NA</accuracy>
                            <semantics>Unicode UTF-16
character</semantics>
                        </simpleData>
                </simpleDataTypes>
        <enumeratedDataTypes>
            <enumeratedData>
                            <name>HLAboolean</name>
                <representation>HLAinteger32BE</representation>
                <semantics>Standard boolean type.</semantics>
                                    <enumerator>
                                           <name>HLAfalse</name>
                                           <values>0</values>
                                    </enumerator>
                                    <enumerator>
                                           <name>HLAtrue</name>
                                           <values>1 </values>
                                    </enumerator>
            </enumeratedData>
        </enumeratedDataTypes>
        <arrayDataTypes>
            <arrayData>
                            <name>HLAASCIIstring</name>
                <dataType>HLAASCIIchar</dataType>
                <cardinality>Dynamic</cardinality>
                <encoding>HLAvariableArray</encoding>
                <semantics>ASCII string representation.</semantics>
                      </arrayData>
                      <arrayData>
                            <name>HLAunicodeString</name>
                <dataType>HLAunicodeChar</dataType>
                <cardinality>Dynamic</cardinality>
                <encoding>HLAvariableArray</encoding>
                <semantics>Unicode string representation.</semantics>
                      </arrayData>
        </arrayDataTypes>
        <fixedRecordDataTypes>
            <fixedRecordData>
                            <name>ExampleStruct</name>
                            <encoding>FixedRecord</encoding>
                            <semantics>NA</semantics>
                    <field>
                                    <name>FieldOne</name>
                                    <dataType>HLAinteger32BE</dataType>
                                    <semantics>NA</semantics>
                            </field>
                            <field>
                                    <name>FieldTwo</name>
                                    <dataType>HLAboolean</dataType>
                                    <semantics>NA</semantics>
                            </field>
                            <field>
                                    <name>FieldThree</name>
                                    <dataType>HLAfloat64BE</dataType>
                                    <semantics>NA</semantics>
                            </field>
            </fixedRecordData>
        </fixedRecordDataTypes>
```

```
    </dataTypes>
</objectModel>
```

Hospital Federate (parameters.xml)

```
<?xml version="1.0" encoding="UTF-8" ?>
<parameters>
<parameter name="randomSeed" displayName="Default Random Seed"
type="int"
                                                defaultValue="0"
                                                isReadOnly="true"
       converter="repast.simphony.parameter.StringConverterFactory$IntC
onverter"/>
                                <parameter name="clinicalStaffCapacity"
displayName="Number of Clinical Staff" type="int"
                                                defaultValue="15"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$IntC
onverter"/>
                                <parameter name="warmup"
displayName="Warmup Period" type="double"
                                                defaultValue="0.0"
                                                isReadOnly="true"

       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                                <parameter name="majorsSD"
displayName="Majors Service Time SD" type="double"
                                                defaultValue="10.0"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                                <parameter name="triageSD"
displayName="Triage Service Time SD" type="double"
                                                defaultValue="2"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                                <parameter name="endTime" displayName="End
Time" type="double"
                                                defaultValue="43200"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter"

              />
                                <parameter name="triageMean"
displayName="Triage Service Time Mean" type="double"
                                                defaultValue="7"
                                                isReadOnly="false"

       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                                <parameter name="minorsMean"
displayName="Minors Service Time Mean" type="double"
                                                defaultValue="30.0"
                                                isReadOnly="false"
```

```
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                                <parameter name="walkInMean"
displayName="Walk In Arrival Mean" type="double"
                                                defaultValue="4.81"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                          <parameter name="walkInSD" displayName="Walk In
Arrival SD" type="double"
                                                defaultValue="0.59"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                                <parameter name="replications"
displayName="Replications" type="int"
                                                defaultValue="1"
                                                isReadOnly="true"
       converter="repast.simphony.parameter.StringConverterFactory$IntC
onverter" />
                                <parameter name="triageCapacity"
displayName="Number of Triage Service Stations" type="int"
                                                defaultValue="5"
                                                isReadOnly="false"

       converter="repast.simphony.parameter.StringConverterFactory$IntC
onverter" />
                                <parameter name="minorsCapacity"
displayName="Number of Minors Service Stations" type="int"
                                                defaultValue="12"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$IntC
onverter" />
                                <parameter name="majorsMean"
displayName="Majors Service Time Mean" type="double"
                                                defaultValue="40"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                                <parameter name="majorsCapacity"
displayName="Number of Majors Service Stations" type="int"
                                                defaultValue="24"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$IntC
onverter" />
                                <parameter name="minorsSD"
displayName="Minors Service Time SD" type="double"
                                                defaultValue="10.0"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
                                <parameter name="ambLamda"
displayName="Ambulance Arrival Lamda" type="double"
                                                defaultValue="0.05"
                                                isReadOnly="false"
       converter="repast.simphony.parameter.StringConverterFactory$Doub
leConverter" />
              </parameters>
```

Ambulance Federate (FederateAmbulance.java)

```
package ambulanceservicemodel;


import repast.simphony.util.collections.IndexedIterable;
import hla.rti1516e.AttributeHandle;
import hla.rti1516e.AttributeHandleValueMap;
import hla.rti1516e.FederateHandleSet;
import hla.rti1516e.InteractionClassHandle;
import hla.rti1516e.LogicalTime;
import hla.rti1516e.NullFederateAmbassador;
import hla.rti1516e.ObjectClassHandle;
import hla.rti1516e.ObjectInstanceHandle;
import hla.rti1516e.OrderType;
import hla.rti1516e.ParameterHandle;
import hla.rti1516e.ParameterHandleValueMap;
import hla.rti1516e.SynchronizationPointFailureReason;
import hla.rti1516e.TransportationTypeHandle;
import hla.rti1516e.encoding.DecoderException;
import hla.rti1516e.encoding.HLAinteger32BE;
import hla.rti1516e.exceptions.FederateInternalError;
import hla.rti1516e.time.HLAfloat64Time;

/**
 * This class handles all incoming callbacks from the RTI regarding a
particular
 * {@link ExampleJava1Federate}. It will log information about any
callbacks it
 * receives, thus demonstrating how to deal with the provided callback
information.
 */
public class FederateAmbassador extends NullFederateAmbassador
{
	//----------------------------------------------------------
	//                     STATIC VARIABLES
	//----------------------------------------------------------

	//----------------------------------------------------------
	//                    INSTANCE VARIABLES
	//----------------------------------------------------------
	// these variables are accessible in the package
	protected double federateTime        = 0.0;
	protected double federateLookahead   = 1.0;

	protected boolean isRegulating       = false;
	protected boolean isConstrained      = false;
	protected boolean isAdvancing        = false;

	protected boolean isAnnounced        = false;
	protected boolean isReadyToRun       = false;
	//private String[]  onames;
	private Federate federate;
	//----------------------------------------------------------
	//                      CONSTRUCTORS
	//----------------------------------------------------------
```

```
	public FederateAmbassador(Federate federate)
	{
		this.federate = federate;
	}
	//----------------------------------------------------------
	//                     INSTANCE METHODS
	//----------------------------------------------------------
	private void log( String message )
	{
		System.out.println( "FederateAmbassador: " + message );
	}
	//////////////////////////////////////////////////////////////////////////
	/////////////////////////// RTI Callback Methods ////////////////////////
	//////////////////////////////////////////////////////////////////////////
	@Override
	public void synchronizationPointRegistrationFailed( String
label,
SynchronizationPointFailureReason reason )
	{
		log( "Failed to register sync point: " + label + ",
reason="+reason );
	}
	@Override
	public void synchronizationPointRegistrationSucceeded( String
label )
	{
		log( "Successfully registered sync point: " + label );
	}
	@Override
	public void announceSynchronizationPoint( String label, byte[]
tag )
	{
		log( "Synchronization point announced: " + label );
		if( label.equals(Federate.READY_TO_RUN) )
			this.isAnnounced = true;
	}
	@Override
	public void federationSynchronized( String label,
FederateHandleSet failed )
	{
		log( "Federation Synchronized: " + label );
		if( label.equals(Federate.READY_TO_RUN) )
			this.isReadyToRun = true;
	}
	/**
	 * The RTI has informed us that time regulation is now enabled.
	 */
	@Override
	public void timeRegulationEnabled( LogicalTime time )
	{
		this.federateTime = ((HLAfloat64Time)time).getValue();
		this.isRegulating = true;
	}
	@Override
	public void timeConstrainedEnabled( LogicalTime time )
```

```
        {
                this.federateTime = ((HLAfloat64Time)time).getValue();
                this.isConstrained = true;
        }
        @Override
        public void timeAdvanceGrant( LogicalTime time )
        {
                this.federateTime = ((HLAfloat64Time)time).getValue();
                this.isAdvancing = false;
        }
        @Override
        public void discoverObjectInstance( ObjectInstanceHandle
theObject,
                                            ObjectClassHandle
theObjectClass,
                                            String objectName )
            throws FederateInternalError
        {
                log( "Discoverd Object: handle=" + theObject + ",
classHandle=" +
                     theObjectClass + ", name=" + objectName );
        }
        @Override
        public void reflectAttributeValues( ObjectInstanceHandle
theObject,
                                            AttributeHandleValueMap
theAttributes,
                                            byte[] tag,
                                            OrderType sentOrder,
                                            TransportationTypeHandle
transport,
                                            SupplementalReflectInfo
reflectInfo )
            throws FederateInternalError
        {
                        // just pass it on to the other method for
printing purposes
                        // passing null as the time will let the other
method know it
                        // it from us, not from the RTI
                        reflectAttributeValues( theObject,
                                                theAttributes,
                                                tag,
                                                sentOrder,
                                                transport,
                                                null,
                                                sentOrder,
                                                reflectInfo );
        }
        @Override
        public void reflectAttributeValues( ObjectInstanceHandle
theObject,
                                            AttributeHandleValueMap
theAttributes,
                                            byte[] tag,
                                            OrderType sentOrdering,
                                            TransportationTypeHandle
theTransport,
```

```
                                                LogicalTime time,
                                                OrderType receivedOrdering,
                                                SupplementalReflectInfo
reflectInfo )
            throws FederateInternalError
        {
                try
                {
                        int msg=-1;
                        int minorHosAvailability=-1;
                        int majorHosAvailability=-1;
                        int HospitalID=-1;
                        for( AttributeHandle attributeHandle :
theAttributes.keySet() )
                        {
                                if(
attributeHandle.equals(federate.aaHandle) )
                                {
        msg=decodeInt(theAttributes.get(attributeHandle));
                                }
                                if(
attributeHandle.equals(federate.abHandle) )
                                {
        minorHosAvailability=decodeInt(theAttributes.get(attributeHandle
));
                                }
                                if(
attributeHandle.equals(federate.acHandle) )
                                {
                                        majorHosAvailability=
decodeInt(theAttributes.get(attributeHandle));
                                }
                                if(
attributeHandle.equals(federate.adHandle) )
                                {
                                        HospitalID=
decodeInt(theAttributes.get(attributeHandle));
                                }
                        }
                        //log("Received Attributes from Hospitals are"+"
MSG:"+msg + ", Hospital ID :"+ HospitalID +", Majors:"+
majorHosAvailability +", Minor:"+minorHosAvailability );

                                if (msg== 0 )
                                {
                        IndexedIterable<Object> hospitals=
federate.context.getObjects(Hospital.class);
                                        Hospital choosenHos;
                                        for(int i=0; i <
hospitals.size();i++)
                                        {
                                                choosenHos = (Hospital)
hospitals.get(i);
                                                if
(choosenHos.getHospitalID()== HospitalID)
                                                {
        //log("ChosenHospital :" + choosenHos.getHospitalID());
```

```
        choosenHos.setHosAvailability(minorHosAvailability,majorHosAvail
ability);
                                                        break;
                                                }

                                        }

                                }
                        }
                        catch( Exception e )
                        {
                                log( "Exception processing received
reflection" );
                                e.printStackTrace();
                        }
        }

        @Override
        public void receiveInteraction( InteractionClassHandle
interactionClass,
                                        ParameterHandleValueMap
theParameters,
                                        byte[] tag,
                                        OrderType sentOrdering,
                                        TransportationTypeHandle
theTransport,
                                        SupplementalReceiveInfo
receiveInfo )
            throws FederateInternalError
            {
                        // just pass it on to the other method for
printing purposes
                        // passing null as the time will let the other
method know it
                        // it from us, not from the RTI
                        this.receiveInteraction( interactionClass,
                                                 theParameters,
                                                 tag,
                                                 sentOrdering,
                                                 theTransport,
                                                 null,
                                                 sentOrdering,
                                                 receiveInfo );
                }
        @Override
        public void receiveInteraction( InteractionClassHandle
interactionClass,
                                        ParameterHandleValueMap
theParameters,
                                        byte[] tag,
                                        OrderType sentOrdering,
                                        TransportationTypeHandle
theTransport,
                                        LogicalTime time,
                                        OrderType receivedOrdering,
                                        SupplementalReceiveInfo
receiveInfo )
            throws FederateInternalError
```

```
	{
		StringBuilder builder = new StringBuilder( "Interaction
Received:" );
		// print the handle
		builder.append( " handle=" + interactionClass );
		if( interactionClass.equals(federate.servedHandle) )
		{
			builder.append( " (DrinkServed)" );
		}
		// print the tag
		builder.append( ", tag=" + new String(tag) );
		// print the time (if we have it) we'll get null if we
are just receiving
		// a forwarded call from the other reflect callback above
		if( time != null )
		{
			builder.append( ", time=" +
((HLAfloat64Time)time).getValue() );
		}
		// print the parameer information
		builder.append( ", parameterCount=" +
theParameters.size() );
		builder.append( "\n" );
		for( ParameterHandle parameter : theParameters.keySet() )
		{
			// print the parameter handle
			builder.append( "\tparamHandle=" );
			builder.append( parameter );
			// print the parameter value
			builder.append( ", paramValue=" );
			builder.append(
theParameters.get(parameter).length );
			builder.append( " bytes" );
			builder.append( "\n" );
		}
		log( builder.toString() );
	}
	@Override
	public void removeObjectInstance( ObjectInstanceHandle
theObject,
					  byte[] tag,
					  OrderType sentOrdering,
					  SupplementalRemoveInfo
removeInfo )
	    throws FederateInternalError
	{
		log( "Object Removed: handle=" + theObject );
	}
	//----------------------------------------------------------
	//                     STATIC METHODS
	//----------------------------------------------------------
	private int decodeInt( byte[] bytes )
	{
		HLAinteger32BE value =
federate.encoderFactory.createHLAinteger32BE();
		// decode
		try
		{
```

```
                        value.decode( bytes );
                }
                catch( DecoderException de )
                {
                        log( "Decoder Exception: "+de.getMessage());
                }
                return value.getValue();
        }
        private double decodeFloat(byte[] bytes )
        {
                HLAinteger32BE value =
federate.encoderFactory.createHLAinteger32BE();
                // decode
                try
                {
                        value.decode( bytes );
                }
                catch( DecoderException de )
                {
                        log("Decoder Exception: "+de.getMessage());
                }
                return value.getValue();
        }
}
```

Hospital Federate (HospitalFederate.java)

```
package hospitalmodel;
import hla.rti1516e.AttributeHandle;
import hla.rti1516e.AttributeHandleSet;
import hla.rti1516e.AttributeHandleValueMap;
import hla.rti1516e.CallbackModel;
import hla.rti1516e.InteractionClassHandle;
import hla.rti1516e.LogicalTime;
import hla.rti1516e.ObjectClassHandle;
import hla.rti1516e.ObjectInstanceHandle;
import hla.rti1516e.ParameterHandleValueMap;
import hla.rti1516e.RTIambassador;
import hla.rti1516e.ResignAction;
import hla.rti1516e.RtiFactoryFactory;
import hla.rti1516e.encoding.EncoderFactory;
import hla.rti1516e.encoding.HLAfloat64BE;
import hla.rti1516e.encoding.HLAinteger16BE;
import hla.rti1516e.encoding.HLAinteger32BE;
import hla.rti1516e.exceptions.FederatesCurrentlyJoined;
import hla.rti1516e.exceptions.FederationExecutionAlreadyExists;
import hla.rti1516e.exceptions.FederationExecutionDoesNotExist;
import hla.rti1516e.exceptions.RTIexception;
import hla.rti1516e.time.HLAfloat64Interval;
import hla.rti1516e.time.HLAfloat64Time;
import hla.rti1516e.time.HLAfloat64TimeFactory;

import java.io.BufferedReader;
import java.io.File;
import java.io.InputStreamReader;
import java.util.Random;
import java.net.MalformedURLException;
```

```
import java.net.URI;
import java.net.URL;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import repast.simphony.context.Context;
import repast.simphony.engine.environment.DefaultRunEnvironmentBuilder;
import repast.simphony.engine.environment.RunEnvironment;
import repast.simphony.engine.environment.RunListener;
import repast.simphony.engine.environment.Runner;
import repast.simphony.engine.schedule.Schedule;
import repast.simphony.engine.schedule.ScheduleParameters;
import repast.simphony.engine.schedule.ScheduledMethod;
import repast.simphony.engine.watcher.Watch;
import repast.simphony.engine.watcher.WatcherTriggerSchedule;
import repast.simphony.essentials.RepastEssentials;
import repast.simphony.relogo.Utility;
import repast.simphony.space.continuous.ContinuousSpace;
import repast.simphony.space.grid.Grid;

public class HospitalFederate {
	//-------------------------------------------------------------
		//                     STATIC VARIABLES
		//-------------------------------------------------------------
		/** The number of times we will update our attributes and
send an interaction */
		public static int majors,minors;
		/** The sync point all federates will sync up on before
starting */
		public static final String READY_TO_RUN = "ReadyToRun";
		private double timestep = 1.0; //time increment/jump used
by RTI
		protected ObjectInstanceHandle objectHandle;
		//-------------------------------------------------------------
		//                    PUSH&SUBSCRIB HANDLERS
		//-------------------------------------------------------------
		protected ObjectClassHandle classHandle;
		protected AttributeHandle aaHandle;
		protected AttributeHandle abHandle;
		protected AttributeHandle acHandle;
		protected AttributeHandle adHandle;
		protected AttributeHandle aeHandle;
		protected InteractionClassHandle servedHandle;
		//-------------------------------------------------------------
		//                    INSTANCE VARIABLES
		//-------------------------------------------------------------
		private RTIambassador rtiamb;
		private HospitalFederateAmbassador fedamb;
		private HLAfloat64TimeFactory timeFactory; // set when we
join
		protected EncoderFactory encoderFactory;     // set when
we join
		public Context<Object> context;
		//-------------------------------------------------------------
		//                       CONSTRUCTORS
		//-------------------------------------------------------------
```

```
		//-------------------------------------------------------
		//                    INSTANCE METHODS
		//-------------------------------------------------------
	/////////////////////////////////////////////////////////////////
	/////////////////////////// Main Simulation Method
	/////////////////////////////////////////////////////////////////
		 * This is the main simulation loop. It can be thought of
as the main method of
		 * the federate. For a description of the basic flow of
this federate, see the
		 * class level comments
		 */
		////////////////////////////// constructor
/////////////////////////////////
		public HospitalFederate(Context<Object>
context,ContinuousSpace<Object> space,Grid<Object> grid,String
federateName) throws Exception
		{
			this.context=context;
			////////////////////////////////////////////////
			// 1 & 2. create the RTIambassador and Connect //
			////////////////////////////////////////////////
			log( "Creating RTIambassador" );
			rtiamb =
RtiFactoryFactory.getRtiFactory().getRtiAmbassador();
			encoderFactory =
RtiFactoryFactory.getRtiFactory().getEncoderFactory();
			// connect
			log( "Connecting..." );
			fedamb = new HospitalFederateAmbassador(this);
			rtiamb.connect( fedamb, CallbackModel.HLA_EVOKED
);
			//////////////////////////////
			// 3. create the federation //
			//////////////////////////////
			log( "Creating Federation..." );
			// We attempt to create a new federation with the
Ambulance.xml
			try
			{
				URL[] modules = new URL[]{
					(new
File("ambulance.xml")).toURI().toURL()
				};
				rtiamb.createFederationExecution(
"AmbulanceFederate", modules );
				log( "Created Federation" );
			}
			catch( FederationExecutionAlreadyExists exists )
			{
				log( "Didn't create federation, it already
existed" );
			}
			catch( MalformedURLException urle )
			{
				log( "Exception loading one of the FOM
modules from disk: " + urle.getMessage() );
				urle.printStackTrace();
```

```
                        return;
                }
                ////////////////////////////
                //      4. join the federation //
                ////////////////////////////
                URL[] joinModules = new URL[]{       (new
File("ambulance.xml")).toURI().toURL()      };
                rtiamb.joinFederationExecution( federateName,
// name for the federate
                                        "AmbulanceFederateType",    //
federate type
                                        "AmbulanceFederate",     // name of
federation
                                        joinModules );           // modules
we want to add
                log( "Joined Federation as " + federateName );
                // cache the time factory for easy access
                this.timeFactory =
(HLAfloat64TimeFactory)rtiamb.getTimeFactory();
                ////////////////////////////////
                // 5. announce the sync point //
                ////////////////////////////////
                // announce a sync point to get everyone on the
same page. if the point
                // has already been registered, we'll get a
callback saying it failed,
                // but we don't care about that, as long as
someone registered it
                rtiamb.registerFederationSynchronizationPoint(
READY_TO_RUN, null );
                // wait until the point is announced
                while( fedamb.isAnnounced == false )
                {
                        rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
                }
                // WAIT FOR USER TO KICK US OFF
                // So that there is time to add other federates,
we will wait until the
                // user hits enter before proceeding. That was,
you have time to start
                // other federates.
                waitForUser();
        ///////////////////////////////////////////////////
                // 6. achieve the point and wait for
synchronization //
        ///////////////////////////////////////////////////
                // tell the RTI we are ready to move past the
sync point and then wait
                // until the federation has synchronized on
                rtiamb.synchronizationPointAchieved( READY_TO_RUN
);
                log( "Achieved sync point: " +READY_TO_RUN+ ",
waiting for federation..." );
                while( fedamb.isReadyToRun == false )
                {
                        rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
// changed need to investigate by Athar
                }
```

```
                        ////////////////////////////
                        // 7. enable time policies //
                        ////////////////////////////
                        // in this section we enable/disable all time
policies
                        // note that this step is optional!
                        enableTimePolicy();
                        log( "Time Policy Enabled" );
                        ////////////////////////////
                        // 8. publish and subscribe //
                        ////////////////////////////
                        // in this section we tell the RTI of all the
data we are going to
                        // produce, and all the data we want to know
about
                        publishAndSubscribe();
                        log( "Published and Subscribed" );
                        /////////////////////////////////////
                        // 9. register an object to update //
                        /////////////////////////////////////
                        this.objectHandle = registerObject();
                        log( "Registered Object, handle=" +
this.objectHandle );
                        /////////////////////////////////////
                        // 9. do the main simulation loop //
                        /////////////////////////////////////
                        // here is where we do the meat of our work. in
each iteration, we will
                        // update the attribute values of the object we
registered, and will
                        // send an interaction.
                        // 9.1 update the attribute values of the
instance //
                        majors = getMajorHosAvailability();
                        minors = getMinorHosAvailability();
                        updateAttributeValues();
                                // 9.2 send an interaction
                        //      sendInteraction();
                                // 9.3 request a time advance and wait
until we get it
                        //advanceTime();
                        log( "Time Advanced to " + fedamb.federateTime );
                }
                ////////////////////////////// Destructor
        ////////////////////////////////////////////////////////////////////
                public void finalize()  throws RTIexception//Destructor
function
                {
                        /////////////////////////////////////
                        // 11. delete the object we created //
                        /////////////////////////////////////
                        deleteObject( objectHandle );
                        log( "Deleted Object, handle=" + objectHandle );
                        /////////////////////////////////////
                        // 12. resign from the federation //
                        /////////////////////////////////////
```

```
                        rtiamb.resignFederationExecution(
ResignAction.DELETE_OBJECTS );
                        log( "Resigned from Federation" );
                        ////////////////////////////////////////
                        // 13. try and destroy the federation //
                        ////////////////////////////////////////
                        // NOTE: we won't die if we can't do this because
other federates
                        //       remain. in that case we'll leave it for
them to clean up
                        try
                        {
                                rtiamb.destroyFederationExecution(
"ExampleFederation" );
                                log( "Destroyed Federation" );
                        }
                        catch( FederationExecutionDoesNotExist dne )
                        {
                                log( "No need to destroy federation, it
doesn't exist" );
                        }
                        catch( FederatesCurrentlyJoined fcj )
                        {
                                log( "Didn't destroy federation, federates
still joined" );
                        }
                }
                /**
                 * This is just a helper method to make sure all logging
it output in the same form
                 */
                private void log( String message )
                {
        System.setProperty("java.util.Arrays.useLegacyMergeSort","true")
;
                        System.out.println( "ExampleFederate   : " +
message );
                }
                /**
                 * This method will block until the user presses enter
                 */
                private void waitForUser()
                {
        log( " >>>>>>>>>> Press Enter to Continue <<<<<<<<<<" );
                        BufferedReader reader = new BufferedReader( new
InputStreamReader(System.in) );
                        try
                        {
                                reader.readLine();
                        }
                        catch( Exception e )
                        {
        log( "Error while waiting for user input: " + e.getMessage() );
                                e.printStackTrace();
                        }
                }
//////////////////////////////// Helper Methods
////////////////////////////////
```

```
	////////////////////////////////////////////////////////
		/**
		 * This method will attempt to enable the various time
related properties for
		 * the federate
		 */
		private void enableTimePolicy() throws RTIexception
		{
			// NOTE: Unfortunately, the
LogicalTime/LogicalTimeInterval create code is
			//	Portico specific. You will have to alter
this if you move to a
			//	different RTI implementation. As such,
we've isolated it into a
			//	method so that any change only needs to
happen in a couple of spots
			HLAfloat64Interval lookahead =
timeFactory.makeInterval( fedamb.federateLookahead );
			////////////////////////////
			// enable time regulation //
			////////////////////////////
			this.rtiamb.enableTimeRegulation( lookahead );
			// tick until we get the callback
			while( fedamb.isRegulating == false )
			{
				rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
			}
			//////////////////////////////
			// enable time constrained //
			//////////////////////////////
			this.rtiamb.enableTimeConstrained();
			// tick until we get the callback
			while( fedamb.isConstrained == false )
			{
				rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
			}
		}
		/**
		 * This method will inform the RTI about the types of
data that the federate will
		 * be creating, and the types of data we are interested
in hearing about as other
		 * federates produce it.
		 */
		private void publishAndSubscribe() throws RTIexception
		{
			/////////////////////////////////////////////
			// publish all attributes of ObjectRoot.Ambulance
			/////////////////////////////////////////////
			// before we can register instance of the object
class ObjectRoot.Ambulance and
			// update the values of the various attributes,
we need to tell the RTI
			// that we intend to publish this information

			// get all the handle information for the
attributes of ObjectRoot.Ambulance
```

```
                        this.classHandle = rtiamb.getObjectClassHandle(
"HLAobjectRoot.Ambulance" );
                        this.aaHandle = rtiamb.getAttributeHandle(
classHandle, "aa" );
                        this.abHandle = rtiamb.getAttributeHandle(
classHandle, "ab" );
                        this.acHandle = rtiamb.getAttributeHandle(
classHandle, "ac" );
                        this.adHandle = rtiamb.getAttributeHandle(
classHandle, "ad" );
                        this.aeHandle = rtiamb.getAttributeHandle(
classHandle, "ae" );
                        // package the information into a handle set
                        AttributeHandleSet attributes =
rtiamb.getAttributeHandleSetFactory().create();
                        attributes.add( aaHandle );
                        attributes.add( abHandle );
                        attributes.add( acHandle );
                        attributes.add( adHandle );
                        attributes.add( aeHandle );
                        // do the actual publication
                        rtiamb.publishObjectClassAttributes( classHandle,
attributes );
        ////////////////////////////////////////////////////////
                        // subscribe to all attributes of
ObjectRoot.Ambulance //
        ////////////////////////////////////////////////////////
                        // we also want to hear about the same sort of
information as it is
                        // created and altered in other federates, so we
need to subscribe to it
                        rtiamb.subscribeObjectClassAttributes(
classHandle, attributes );
        ////////////////////////////////////////////////////
                        // publish the interaction class
InteractionRoot.X //
        ////////////////////////////////////////////////////
                        // we want to send interactions of type
InteractionRoot.X, so we need
                        // to tell the RTI that we're publishing it
first. We don't need to
                        // inform it of the parameters, only the class,
making it much simpler
                        servedHandle = rtiamb.getInteractionClassHandle(
"InteractionRoot.X" );
                        // do the publication
                        rtiamb.publishInteractionClass(servedHandle);
        ////////////////////////////////////////////////////////
                        // subscribe to the FoodServed.DrinkServed
interaction //
        ////////////////////////////////////////////////////////
                        // we also want to receive other interaction of
the same type that are
                        // sent out by other federates, so we have to
subscribe to it first
                        rtiamb.subscribeInteractionClass(servedHandle);
                } /**
```

```
                * This method will register an instance of the class
ObjectRoot.A and will
                * return the federation-wide unique handle for that
instance. Later in the
                * simulation, we will update the attribute values for
this instance
                */
               private ObjectInstanceHandle registerObject() throws
RTIexception
               {
                       return rtiamb.registerObjectInstance( classHandle
);
               }
               /**
                * This method will update all the values of the given
object instance. It will
                * set each of the values to be a string which is equal
to the name of the
                * attribute plus the current time. eg "aa:10.0" if the
time is 10.0.
                * <p/>
                * Note that we don't actually have to update all the
attributes at once, we
                * could update them individually, in groups or not at
all!
                */
               public void updateAttributeValues() throws RTIexception
               {
                       ////////////////////////////////////////////////
                       // create the necessary container and values //
                       ////////////////////////////////////////////////
                       // create a new map with an initial capacity -
this will grow as required
                       AttributeHandleValueMap attributes =
rtiamb.getAttributeHandleValueMapFactory().create(4);
                       //sends hospital id and minor and major capacity
of the hospital to the ambulance model.
                       // create the collection to store the values in,
as you can see
                       // this is quite a lot of work. You don't have to
use the encoding
                       // helpers if you don't want. The RTI just wants
an arbitrary byte[]
                       // generate the value for the number of cups
(same as the timestep)
                       HLAinteger32BE aaValue =
encoderFactory.createHLAinteger32BE( 0 );
                       HLAinteger32BE abValue =
encoderFactory.createHLAinteger32BE(getMinorHosAvailability());
                       HLAinteger32BE acValue =
encoderFactory.createHLAinteger32BE( getMajorHosAvailability() );
                       HLAinteger32BE adValue =
encoderFactory.createHLAinteger32BE(HospitalBuilder.HOS_ID);
                       attributes.put( aaHandle, aaValue.toByteArray()
);
                       attributes.put( abHandle, abValue.toByteArray()
);
```

```
                        attributes.put( acHandle, acValue.toByteArray()
);
                        attributes.put( adHandle, adValue.toByteArray()
);
                //      log("Sending Attributes from Hospitals" +
"Hospital ID :"+ HospitalBuilder.HOS_ID +", Majors:"+
getMajorHosAvailability() +", Minor:"+getMinorHosAvailability() );
                        ////////////////////////
                        // do the actual update //
                        ////////////////////////
                        //rtiamb.updateAttributeValues( objectHandle,
attributes, generateTag() );
                        // note that if you want to associate a
particular timestamp with the
                        // update. here we send another update, this time
with a timestamp:
                        HLAfloat64Time time = timeFactory.makeTime(
fedamb.federateTime+fedamb.federateLookahead );
                        rtiamb.updateAttributeValues( objectHandle,
attributes, generateTag(), time);
                }
                /**
                 * This method will send out an interaction of the type
InteractionRoot.X. Any
                 * federates which are subscribed to it will receive a
notification the next time
                 * they tick(). Here we are passing only two of the three
parameters we could be
                 * passing, but we don't actually have to pass any at
all!
                 */
                public void sendInteraction(int status) throws
RTIexception
                {
                        ////////////////////////
                        //      send the interaction //
                        ////////////////////////
                        ParameterHandleValueMap parameters =
rtiamb.getParameterHandleValueMapFactory().create(0);
                        rtiamb.sendInteraction( servedHandle, parameters,
generateTag() );
                        // if you want to associate a particular
timestamp with the
                        // interaction, you will have to supply it to the
RTI. Here
                        // we send another interaction, this time with a
timestamp:
                        HLAfloat64Time time = timeFactory.makeTime(
fedamb.federateTime+fedamb.federateLookahead );
                        rtiamb.sendInteraction( servedHandle, parameters,
generateTag(), time );
                }
                /**
                 * This method will request a time advance to the current
time, plus the given
                 * timestep. It will then wait until a notification of
the time advance grant
                 * has been received.
```

```
		 */
		@ScheduledMethod(start = 1, interval = 1, priority =
ScheduleParameters.LAST_PRIORITY)
		public void advanceTime()  throws RTIexception
		{
			// request the advance
			if ((majors!=getMajorHosAvailability()) ||
(minors!=getMinorHosAvailability()))
			{
				majors=getMajorHosAvailability();
				minors=getMinorHosAvailability();
				updateAttributeValues();
			}
			fedamb.isAdvancing = true;
			HLAfloat64Time time = timeFactory.makeTime(
fedamb.federateTime + timestep );
			rtiamb.timeAdvanceRequest( time );
			// wait for the time advance to be granted.
ticking will tell the
			// LRC to start delivering callbacks to the
federate
			while( fedamb.isAdvancing )
			{
				rtiamb.evokeMultipleCallbacks( 0.1, 0.2 );
			}
		}
		/**
		 * This method will attempt to delete the object instance
of the given
		 * handle. We can only delete objects we created, or for
which we own the
		 * privilegeToDelete attribute.
		 */
		private void deleteObject( ObjectInstanceHandle handle )
throws RTIexception
		{
			rtiamb.deleteObjectInstance( handle,
generateTag() );
		}
		private byte[] generateTag()
		{
			return ("(timestamp)
"+System.currentTimeMillis()).getBytes();
		}
		//----------------------------------------------------------
		//                     STATIC METHODS
		//----------------------------------------------------------
		public int getMinorHosAvailability(){// gives us the
minor capacity to be send to ambulance model by athar
			int minhos=0;
			int minors =
HospitalBuilder.minorsR.getnumServers()-
HospitalBuilder.minorsR.getnumBusy();
			int staff =
HospitalBuilder.clinicalStaffR.getnumServers()-
HospitalBuilder.clinicalStaffR.getnumBusy();
			minhos = Math.min(minors, staff);
			return minhos;
```

```
            }
            public int getMajorHosAvailability(){ // gives us the
major capacity to be send to ambulance model by athar
                    int majorhos=0;
                    int majors =
HospitalBuilder.majorsR.getnumServers()-
HospitalBuilder.majorsR.getnumBusy();
                    int staff =
HospitalBuilder.clinicalStaffR.getnumServers()-
HospitalBuilder.clinicalStaffR.getnumBusy();
                    majorhos = Math.min(majors, staff);
                    return majorhos;
            }
}
```

Appendix 3: CBDS Launch Script - Ansible PlayBook

```
- name: ansible automation for amb hosts
  hosts: amb_group
  gather_facts: no
  tasks:
    - name: Ensure output folder exists
      file:
        path: ~/output
        state: directory
    - name: Ensure out.txt file exist for logging activities
      copy:
       dest: ~/output/out.txt
       content:  "{{  lookup('pipe','date  +%Y-%m-%d-%H-%M-%S')  }}:
Starting program !"
    - name: starting with the first VM for the amb
      shell: "./amb.sh --federate-name Ambulance --peers Hospital1 >>
~/output/out.txt"
      args:
        chdir: ~/amb
      async: 1000
      poll: 0
    - name: AMB - check on task string
      wait_for:
       path: ~/output/out.txt
       search_regex: 10Sec Waiting for Hospitals to Join
- name: ansible automation for hos hosts
  hosts: hos_group
  gather_facts: no
  tasks:
    - name: Ensure output folder exists
      file:
        path: ~/output
        state: directory
```

```
    - name: joining hos to the AMB
      shell: "./hos.sh --federate-name Hospital{{ inventory_hostname
| regex_replace('[^0-9]','') }} --peers Ambulance > ~/output/out.txt"
      args:
        chdir: "~/{{ inventory_hostname }}"
      async: 1000
      poll: 0
      register: amb_sleeper
- name: record end time
  hosts: amb_group
  gather_facts: no
  tasks:
    - name: AMB - wait till the program exits
      wait_for:
        path: ~/output/out.txt
        search_regex: "FederateAmbassador: Object Removed: handle"
    - name: Ensure out.txt file exist for logging activities
      lineinfile:
       path: ~/output/out.txt
       insertafter: EOF
       line:   "{{   lookup('pipe','date   +%Y-%m-%d-%H-%M-%S')   }}:
Experiment Finished!"
    - name: Copy the simulation output to ~/output folder - AMB
      copy:
        src:
~/amb/MyModels/ambulanceservicemodel/ambulanceOutput.csv
        dest: ~/output
        remote_src: yes
        force: yes
- name: ansible automation for hos hosts
  hosts: hos_group
  gather_facts: no
  tasks:
    - name: Copy the simulation output to ~/output folder - HOS
      copy:
        src:                 "~/{{                 inventory_hostname
}}/MyModels/hospitalmodel/hospitalOutput_ID({{  inventory_hostname  |
regex_replace('[^0-9]','') }}).csv"
        dest: ~/output
        remote_src: yes
        force: yes
```

<<End of Document>>