A Framework for Evaluating the Impact of Communication on Performance in Large-Scale Distributed Urban Simulations

Kwabena Ntim Amponsah

Intelligent Modelling and Analysis Group

School of Computer Science

University of Nottingham

This thesis is submitted for the degree of Doctor of Philosophy

May 2021

ABSTRACT

A primary motivation for employing distributed simulation is to enable the execution of large-scale simulation workloads that cannot be handled by the resources of a single stand-alone computing node. To make execution possible, the workload is distributed among multiple computing nodes connected to one another via a communication network. The execution of a distributed simulation involves alternating phases of computation and communication to coordinate the co-operating nodes and ensure correctness of the resulting simulation outputs. Reliably estimating the execution performance of a distributed simulation can be difficult due to non-deterministic execution paths involved in alternating computation and communication operations. However, performance estimates are useful as a guide for the simulation time that can be expected when using a given set of computing resources. Performance estimates can support decisions to commit time and resources to running distributed simulations, especially where significant amounts of funds or computing resources are necessary. Various performance estimation approaches are employed in the distributed computing literature, including the influential Bulk Synchronous Parallel (BSP) and LogP models. Different approaches make various assumptions that render them more suitable for some applications than for others. Actual performance depends on characteristics inherent to each distributed simulation application. An important aspect of these individual characteristics is the dynamic relationship between the communication and computation phases of the distributed simulation application. This work develops a framework for estimating the performance of distributed simulation applications, focusing mainly on aspects relevant to the dynamic relationship between communication and computation during distributed simulation execution. The framework proposes a meta-simulation approach based on the Multi-Agent Simulation (MAS) paradigm. Using the approach proposed by the framework, meta-simulations can be developed to investigate the performance of specific distributed simulation applications. The proposed approach enables the ability to compare various what-if scenarios. This ability is useful for

comparing the effects of various parameters and strategies such as the number of computing nodes, the communication strategy, and the workload-distribution strategy. The proposed meta-simulation approach can also aid a search for optimal parameters and strategies for specific distributed simulation applications. The framework is demonstrated by implementing a meta-simulation which is based on case studies from the Urban Simulation domain.

ACKNOWLEDGEMENTS

"He has made everything beautiful in its time." Ecc 3:11.

The outcome of this journey has been influenced directly or indirectly by the immense support that I have been fortunate enough to receive from numerous people, and I would like to express my profound gratitude to them.

First of all, I thank my principal supervisor Dr Peer-Olaf Siebers for his patient guidance, encouraging mentorship, for always being ready to lend an ear, and inspiring confidence. I have been blessed with an outstanding supervision team who have helped me navigate the journey. I am indebted to Dr Brian Logan for his knowledgeable comments and invaluable suggestions, to Dr Sameh Zakhary for his constant support, attention to detail and critical feedback, and to Professor Paul Nathanail for his indispensable advice and lending a broader perspective. I am privileged to have received guidance from my supervisors, and this work would not be possible without the counsel they provided.

I am grateful for the companionship of my PhD comrades from the Intelligent Modelling and Analysis group (IMA), and from the Laboratory for Urban Complexity and Sustainability (LUCAS). Their company has helped to brighten the path on this journey and made it more fun and interesting.

The unrelenting care and support I have received from my family throughout my life has made this moment possible. I am thankful for my father James and my sister Abena. This thesis is dedicated to my mother Christine of blessed memory.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Abbreviations

ABS	Agent-Based Simulation
BES	Building Energy Simulation
BSP	Bulk Synchronous Parallel model
CA	Cellular Automata
CityGML	City Geographic Mark-up Language
CMB	Chandy-Misra-Bryant algorithm
COTS	Commercial Off-The-Shelf
DES	Discrete Event Simulation
DS	Distributed Simulation
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FOM	Federation Object Model
HLA	High Level Architecture
HPC	High Performance Computing
LP	Logical Process
MAS	Multi-Agent Simulation
MPI	Message Passing Interface
No-MASS	Nottingham Multi-Agent Stochastic Simulation
OMT	Object Model Template

PDES	Parallel Discrete Event Simulation
PRAM	Parallel Random Access Machine model
RO	Receive Order
RTI	Run Time Infrastructure
SOM	Simulation Object Model
TAG	Time Advance Grant
TAR	Time Advance Request
TSO	Time Stamp Order
TWOS	Time Warp Operating System

1 INTRODUCTION

1.1 Motivation

Some Urban Simulations can grow in scale to the point where they cannot be executed efficiently using the resources (e.g. memory, CPU, disk space) of a single stand-alone computer (Zehe *et al.*, 2015). One approach to managing such large-scale simulations is to execute them on a distributed computer system composed of two or more computing nodes connected to one another via a communication network. Using the distributed approach, a large-scale simulation workload is partitioned into smaller workloads which are assigned to various computing nodes and executed in coordination. In order for the distributed simulation to produce correct results, it is important for the participating computing nodes to cooperate with one another by communicating during simulation execution. In this manner, a distributed simulation involves alternating computation and communication operations.

Communication is used in the distributed simulation to ensure data integrity by supplying each computing node with the data it requires from other nodes to process simulation events during distributed execution. As the concurrent execution paths of the distributed nodes proceed in a non-deterministic order, it is also essential for communication to deliver timing information that can be used to synchronize the nodes and ensure that distributed simulation events are processed in the correct causal order. Consequently, synchronization regulates the pace at which separate parts of the distributed simulation advance with respect to one another. Communication between computing nodes involves sending and receiving data over their interconnecting network, which introduces an additional performance bottleneck that is not normally present in stand-alone simulations. The extent to which the added communication overheads can impact distributed simulation performance depends on factors such as the frequency of communication between the distributed computing nodes, the size of data that needs to be exchanged and the latency and bandwidth of the interconnecting network.

The work in this thesis focuses on developing a framework for evaluating the execution performance of distributed simulations, taking into account the dynamic relationship between non-deterministic communication and computation operations necessary to produce correct outputs.

The case studies that are employed to demonstrate the framework proposed in this thesis are selected from the Urban Simulation domain. Cities are large complex systems where simulation studies are needed to support important decision making. This is evident in the range of tools available for conducting simulation studies on various urban systems including buildings and transportation. The need to conduct Urban Simulations is given urgency by the fast-growing world urban population. The United Nations Department of Economic and Social Affairs (UN DESA) reports that the proportion of the world's population residing in urban areas rose from 30% in 1950 to 54% in 2014. The UN DESA further projects that the proportion of the world's urban population will exceed 66% by 2050. These figures represent a progression from 746 million urban dwellers in 1950 to 3.9 billion in 2014 and 6.3 billion in 2050. The UN DESA also reports that while about half of urban dwellers in 2014 lived in cities with a population less than 500,000, there were 28 highly populated *mega-cities* in 2014, each with a population in excess of 10 million people. These mega-cities accounted for one-eighth of the global urban population in 2014 (United Nations, 2014). This rapid growth in the urban population creates a significant need for support tools to assist decision-makers in making the right choices that will enable urban areas to accommodate population growth in a sustainable manner as urban demand continues to grow for energy, housing, transportation and other resources and services (Jain *et al.*, 2016).

Cities are composed of subsystems such as transportation and housing which encompass concerns that can logically be separated, but which also have influential interactions with one another. As it is difficult to capture all urban subsystems in a single model, Urban Simulation approaches normally focus on modelling specific urban subsystems rather than attempting to build a unified urban model that encompasses all subsystems. This approach has the advantage of focusing on a single problem so that the required domain expertise can be brought to bear on the specific urban subsystem being studied in order to produce a sufficiently accurate model for the limited set of concerns covered by that subsystem. However, in cases where the

simulation study requires interaction between one or more urban subsystems, it becomes necessary for the model to capture relationships between the concerns covered by all subsystems relevant to the study. A convenient solution to this case is to employ distributed simulation to unify existing models that focus on separate urban subsystems. This has the advantage of re-using established domain-specific simulation models developed by domain experts to prevent the unnecessary duplication of efforts which may require a significant investment of time. The *heterogeneous* simulations need be coordinated together in a single distributed simulation execution such that they interact meaningfully to produce correct and reproducible results. Enabling such *interoperability* between heterogeneous simulations requires careful consideration of aspects such as different simulation paradigms and implementation tools, the consistency of measurement units between simulations, and compatibility of modelling assumptions.

The simulation approach employed can have a significant impact on simulation size. On one hand, micro-level simulations try to replicate the behaviour of low-level entities in the system being studied. On the other hand, macro-level simulations focus on directly reproducing high-level patterns that may arise as a result of interactions between low-level entities. Simulations that model micro-level interactions in particular tend to grow large when considering the urban scale. The number of low-level entities and interactions tends to be much larger than the corresponding number of high-level entities and interactions. This generally results in a greater need for computing resources where micro-level simulations are involved. Therefore, using the macro-level simulation approach is a sensible choice in cases where it is applicable because complexity is reduced, and fewer computing resources are required. However, some cases are better suited to the micro-level simulation approach. This includes cases where it is difficult to directly reproduce high-level behaviours with sufficient accuracy. For example, *emergent phenomena* occur in Agent-Based Simulations in scenarios where micro-level interactions can lead to macro-level patterns that would be difficult to program explicitly.

Simulation approaches that result in large simulation size are essential in some cases but can place a large demand on computing resources. This presents the challenge of *scaling* up computing resources to support a single large *homogeneous* simulation that may not be possible to execute on a stand-alone computing node. A suitable

example of this case is the CitySim (Robinson 2012) tool for Building Energy Simulation (BES). When provided with a large enough model configuration, its execution requirements can exceed the memory capacity of a single computing node with reasonable memory resources. CitySim is designed to run physics-based Building Energy Simulations on a cluster of buildings represented by a 3D model. It calculates micro-level interactions between building surfaces and the environment during execution. Simulations at the neighbourhood scale consider only a few hundred buildings and can be executed on a single computing node within reasonable time. However, simulations at the urban scale comprising tens of thousands of buildings are more difficult to manage on a single node.

Taylor (2019) identifies three main modes for the use of distributed simulation:

- Scalability (Mode A): In this mode, a single homogeneous simulation model is partitioned into sub-models and distributed over multiple computing nodes, offering the prospect of speedup due to parallel execution. For large-scale models that cannot be managed within the resources of a single computing node, this mode also makes it possible for the simulation to be executed in the first place. For example, Collier, Ozik and Macal (2015) develop a distributed simulation, using multiple computing nodes to execute a large-scale disease transmission model which tracks the movements and interactions of 2.9 million individuals in a large city.

- Interoperability (Mode B): For this mode, multiple *heterogeneous* simulation models which would normally be executed separately are linked together to enable communication and co-ordinated as a single simulation execution. This mode also facilitates the re-use of existing simulation models. For example, Anagnostou, Nouman and Taylor (2013) develop a distributed simulation for Emergency Medical Services which links an Ambulance service model with an Accident & Emergency service model.

- Batch Runs (Mode C): This mode refers to the case where a large number of separate simulation runs is needed to obtain a required level of confidence in the results of a simulation experiment (Law and Kelton, 1984). The simulation runs can be performed in parallel to obtain the desired results. As

the simulation runs in this case are independent of one another, this is considered a pleasingly parallel problem and communication involved is minimal.

The framework proposed in this thesis considers two approaches to using the distributed simulation approach: to tackle the *scalability* concern for large homogeneous Urban Simulations, and to tackle the *interoperability* concern for heterogeneous Urban Simulations. This scope is reasonable because both approaches are useful in the urban context. Urban simulations can grow exceptionally large and may also need to be coupled with simulations of other urban subsystems. However, the scope does not cover the third concern of *batch runs* as communication is minimal or non-existent in this case.

These issues of scalability and interoperability are important matters in the wider literature, in which several approaches have been proposed for orchestrating distributed simulations to address these concerns. A discussion of approaches to distributed simulation is presented in Chapter 2. Among these, the IEEE High Level Architecture (HLA) (IEEE, 2010a) is one approach that is considered a mature standard (Strassburger, Schulze and Fujimoto, 2008). The experimental work conducted on distributed Urban Simulation in Chapter 5 is based on the HLA standard, as it includes features that support dealing with interoperability as well as scalability in distributed simulations. Chapter 3 outlines a more detailed rationale for selecting this approach.

The research work carried out in this thesis has been performed within the context of the Leverhulme Sustaining Urban Habitats (SUH) project. The SUH project is an interdisciplinary research effort to understand and evaluate urban sustainability in order to inform policies for sustainable urban growth. To achieve its goals, one of the methods used by the SUH project is to utilize various Urban Simulation applications to investigate different what-if policy scenarios and estimate the long-term effects on urban sustainability. Following this approach, the project has a need to run Urban Simulations on real life case studies, some of which can grow large in scale to the point where the resources required to execute the simulation exceed the capacity of a single computer. The SUH project is composed of six themes. Three reflect various

aspects of urban sustainability – the *Environmental*, *Economic*, and *Social* themes. Two include the expertise needed to conduct modelling and simulation efforts – the *Measurement and Data*, and the *Modelling and Optimisation* themes. The sixth theme, *Policy and Governance*, investigates policy issues related to urban sustainability. The SUH project aims to develop methodology that is general enough for broad application to urban areas other than its two case study cities: Nottingham and Shanghai.

1.2 Aim and Objectives

The work in this thesis focuses on developing a framework for evaluating the execution performance of distributed simulations, taking into account the dynamic relationship between interleaved non-deterministic communication and computation operations necessary to produce correct simulation outputs. Following this approach, the framework will consider various communication related parameters and alternative communication strategies that can have significant impact on distributed simulation performance. The proposed framework will enable selection of optimal communication related parameters and strategies for the efficient execution of Urban Simulations that require distribution over multiple computing nodes. The development of the framework, which is the primary contribution of this thesis, will be informed by distributed simulation experiments that will be carried out using appropriate case studies from the Urban Simulation domain. A concrete meta-simulation will also be implemented to demonstrate the application of the completed framework.

The aim will be achieved by carrying out the following objectives:

- Literature Review:
    - Identify performance bottlenecks from research literature relevant to the execution of large-scale distributed simulations.
    - Isolate significant bottlenecks and investigate how they exert a concerted influence on distributed execution performance.

- Identify relevant parameters and alternative strategies for managing communication during distributed simulation execution.
- Identify existing distributed performance models.

- Homogeneous Experimentation:
  - Case Study Selection:

    Identify homogeneous Urban Simulation that satisfies the criteria:
    - At whole city level, the *scalability* problem is encountered. The simulation becomes too large to execute using a stand-alone computing node.
    - The simulation workload can be distributed.
    - The simulation can potentially interoperate meaningfully with other Urban Simulation models for a heterogeneous case study.

  - Experimentation:
    - Develop a homogeneous distributed simulation based on the selected Urban Simulation, using the HLA to address the scalability concern.
    - Conduct experiments to obtain performance measurements, varying parameters and using alternative communication strategies.

- Heterogeneous Experimentation:
  - Case Study Selection:

    Identify a second Urban Simulation that satisfies the criteria:
    - The simulation can interoperate meaningfully with the simulation selected for Case Study One.
    - The simulation workload can be distributed.
    - The simulation will introduce communication patterns that differ from those investigated in Case Study One.

- Experimentation:
  - Develop a heterogeneous distributed simulation based on the two selected Urban Simulations, using the HLA to address the *interoperability* issue.
  - Conduct experiments to obtain performance measurements, varying parameters and using alternative communication strategies.

- Framework Development:
  - Analysis of Experimental Results:
    - Compare the effects that various communication strategies have on performance.
    - For communication strategies that can incur data loss, investigate the effect on simulation output accuracy.

  - Conceptual Model:
    - Identify the main components required to model the distributed case study experiments.
    - Isolate the important interactions between the identified components.
    - Identify the significant parameters that influence relationships between the identified components.
    - Incorporate the capability to consider alternative communication strategies.

  - Demonstration:
    - Implement a meta-simulation based on the developed framework.
    - Use the implemented simulation to reproduce the performance of the experimental case studies.

1.3 Contribution

The primary contribution of the work in this thesis is the development of a framework for estimating the execution performance of distributed simulation applications. The framework is a contribution to the research field of Parallel and Distributed Simulation. The developed framework proposes a meta-simulation approach which enables performance evaluation for distributed simulation applications in which the dynamic relationship between communication and computation can significantly influence execution performance. The framework uses the Multi-Agent Simulation (MAS) paradigm to set out the main components of a distributed simulation and define how they interact with one another during execution. Using the framework, custom meta-simulations can be created for evaluating specific distributed simulation applications. The approach proposed by the framework also enables the comparison of various what-if scenarios, exploring different communication strategies. This capability can aid a search for optimal strategies for the distributed simulation application under investigation. A demonstration of the framework has been provided by the implementation of a meta-simulation based on real world case studies selected from the Urban Simulation domain.

1.4 Publications

Part of the research work presented in this thesis has been published at the following conference:

Amponsah, K., Zakhary, S., Robinson, D., Nathanail, P., Logan, B., & Siebers, P. O. (2019). 'Distributed building energy simulation with the HLA', *Proceedings of the 2019 Summer Simulation Conference*, pp. 1-12.

1.5 Structure of the Thesis

Chapter 1 introduces the subject matter of this thesis and explains the motivation and context for performing the research work. Set against this background, the research aim is explained, and details are provided for the relevant objectives required to achieve the aim.

Chapter 2 provides a review of the literature on previous research relevant to the work conducted in this thesis. This mostly includes research from the field of Parallel and Distributed Simulation. It also covers work from the field of Urban Simulation, focusing in particular on Building Energy Simulation.

Chapter 3 proposes an approach for performing the work that is set out in the research aims and objectives. It explains the approach selected and provides rationale for justifying the choice of approach. This includes the advantages and disadvantages of the chosen approach in comparison to other potentially feasible approaches. This chapter also introduces appropriate Urban Simulation case studies which are suitable for the chosen approach. Finally, this chapter provides details of various communication strategies which will be employed in the case study experiments.

Chapter 4 discusses the context within which the framework is to be developed, exploring the key factors to consider. Several factors relevant to distributed simulation performance are discussed. The relationships between the factors are established in relation to communication. These preliminary considerations provide a background context within which the framework will be developed, based on the results from subsequent experimental work conducted on the selected case studies.

Chapter 5 explains the Urban Simulation case studies in more detail. Distributed simulation experiments are conducted on the selected case studies using the methodology proposed in Chapter 3. Results from the experiments are presented, showing the influence of the communication strategies used on execution performance and simulation output accuracy.

Chapter 6 presents the framework based on the context provided by Chapter 4, the results from the distributed simulation experiments in Chapter 5, and lessons learned from developing the distributed simulations for experimental work. Here, details are given about the components of the framework, explaining how they interact with one another. A demonstration of the framework is provided using a concrete implementation of the framework, based on the experimental results from Chapter 5.

Chapter 7 provides a summary of the contributions made by the work in this thesis. It explains how the aim and objectives which were set out in Chapter 1 have been achieved by the work in various chapters. It discusses the current limitations of the framework and proposes potential directions for future work.

2 LITERATURE REVIEW

2.1 Computer Modelling and Simulation

As explained by White and Ingalls (2016), a simulation model is a simplified representation of a real or abstract system, and simulation is the process of executing an experiment on such a representative model in order to mimic the behaviour of the modelled system. As noted by Epstein (2008), simulations are useful for many different purposes; among other things, they can be used to predict and compare the outcomes of various what-if-scenarios, to gain a better understanding or explanation of how a system functions, and can also be used for training purposes. Robinson (2014) further elaborates that some circumstances where simulations are useful include instances where it is not feasible nor desirable to run experiments on an actual system, and situations where the desired results cannot be easily obtained from direct mathematical analysis, as is the case with many real-life complex systems including cities. A simulation model captures the system components and interactions that are essential to the phenomena being studied, and incorporate rules that govern how the system state evolves as time moves forward (Borshchev and Filippov, 2004). Different approaches to simulation modelling vary in how they capture these essential elements, and the characteristics of the system and phenomena being studied determines the suitability and effectiveness of a given modelling approach. A general overview of simulation characteristics and alternative modelling approaches is provided in the following sections.

2.1.1 Model Properties: Discrete, Stochastic and Dynamic

As explained by Law & Kelton (2000), simulation models can be classified using three sets of properties, as either *continuous* or *discrete*, as either *deterministic* or *stochastic*, and as either *static* or *dynamic*. The state in a computer simulation is represented by a collection of variables that are updated as and when necessary, to reflect changes in the simulated system.

- Continuous vs Discrete:

  In continuous simulation models, the phenomena being studied in the real system varies continuously through time, and the relevant state variables are continuously updated in the simulation to track the changing system state. On the other hand, the system state in discrete models changes at instantaneous points in time, and accordingly the relevant state variables need only to be updated at those specific points in time when the changes occur.

- Deterministic vs Stochastic:

  Deterministic simulation models do not include any elements of randomness during simulation execution. From a given initial state, a deterministic simulation always proceeds along the same execution path and produces the same results. Stochastic simulation models, on the other hand, include elements of randomness and can therefore proceed along different execution paths with given probabilities. Stochastic computer simulations handle randomness by drawing values from pseudo-random number streams. This procedure is important for the reproducibility of stochastic simulation experiments, as using a fixed seed for the pseudo-random number generator will reproduce the same stream of numbers. Using the same random seed effectively reduces a stochastic simulation to follow a single deterministic execution path, which makes it possible for experiments to be reproduced.

- Dynamic vs Static:

  For dynamic simulation models, tracking the passage of time is an explicit part of the model which is essential for the progression of the simulation execution. The system state in a dynamic simulation is updated progressively as time moves forward. On the other hand, static simulation models do not

have a need to model the flow of time as part of simulation execution. Nevertheless, static simulation models can potentially be used to produce forecasts into the future, for example with the use of Monte-Carlo simulation methods for the purpose of risk analysis (Schriber, 2009).

2.1.2 Levels of Abstraction: Micro to Macro

Simulation models can be used to capture the behaviour of systems at different levels of abstraction, ranging from the micro-level or low abstraction level to the macro-level or high abstraction level (Borshchev and Filippov, 2004). Simulation models created at the micro-level include detailed representations of the entities comprising the system and their interactions with one other. The collective behaviour of the individual micro-level entities produces aggregate patterns and trends that can be observed on the entire system.

On the other hand, macro-level simulation models focus on modelling the aggregate system behaviour directly without modelling the micro-level entities that contribute to producing the aggregate trends. For a given system, it may be possible to use both the micro-level and macro-level modelling approaches to produce simulation models at different levels of abstraction. For example, traffic simulation models have been created using both the micro-level and macro-level modelling approaches (Helbing *et al.*, 2002). Traffic simulation at the micro-level defines the movements of each car and the interactions it has with neighbouring cars. The flow of traffic in the simulation emerges from these defined micro-level behaviours. On the other hand, traffic simulation at the macro-level accounts directly for flow of traffic. Generally, a macro-level simulation model of a given system can be executed faster than an equivalent micro-level model as it the macro-level simulation does not need to account for micro-level behaviour.

2.1.3 Time Flow Mechanisms

As discussed, tracking the flow of time as simulation execution progresses is an essential property of dynamic simulation models. Fujimoto (1998) describes three different concepts of time that are of interest during a simulation execution – *wall-clock time*, *physical time*, and *simulation time*:

- Wall-Clock Time:
  This refers to the actual time that passes in the real world while the simulation executes. For example, a traffic simulation that is run for 1 minute to simulate 24 hours of traffic flow has used up 1 minute of wall-clock time.

- Physical Time:
  This refers to the progression of time within the simulated system itself. For example, the previously mentioned traffic simulation progresses 24 hours in physical time.

- Simulation/Logical Time:
  This is closely related to the concept of physical time. Simulation time or logical time refers to the units used within the simulation to represent the passage of physical time during simulation execution. For example, if time is represented by a positive integer in the traffic simulation with an increment of 1 corresponding to an advancement of 1 minute in physical time, simulation time after 24 hours of physical time is 1,440.

Fujimoto (1998) further describes how the relationships between these concepts of time are used to control time flow in various simulation applications: *real-time*, *scaled real-time* and *as-fast-as-possible* simulations.

- Real-Time:
  In real-time simulations, simulation time proceeds at the same pace as wall-clock time.

- Scaled Real-Time:
  In scaled real-time simulations, the pace at which simulation time proceeds is directly proportional to the pace of wall-clock time, but not equal. Assuming sufficient processor speed, the pace may be greater to allow quicker

execution than real-time, or slower to allow more detailed examination of processes that occur too quickly in real-time for direct observation. For example, simulation time may advance at 10 seconds for every second of wall-clock time for execution that is faster than real-time. Conversely, simulation time may advance at 1 second for every 10 seconds of wall-clock time for execution slower than real-time.

- As-Fast-As-Possible:
  In as-fast-as-simulations, the goal is simply to complete the simulation execution in the quickest time allowed by the available processing resources. In this case, the simulation time is not paced proportionally to wall-clock time at all.

During the execution of a dynamic computer simulation model, simulation time is divided into discrete timesteps. In practice, both discrete and continuous computer simulations use such discrete timesteps. Continuous simulations use them as an approximation for continuous time flow, and the chosen size of timestep reflects on the fidelity of simulation results. Smaller timesteps produce more accurate results, but also demand more processing time for simulation execution as the total number of timesteps is increased. A computer simulation execution progresses in simulation time by moving from the current timestep to another timestep in the future, updating the system state each time it advances in time. As noted by Ferscha and Tripathi (1998), the two main approaches for advancing simulation time are the *time-driven* mechanism and the *event-driven* mechanism. With the time-driven approach, simulation time progresses at regular time intervals, commonly referred to as *ticks*, and state variables are updated at each tick. On the other hand, event-driven time flow uses the concept of *events* to designate points in time where system state is scheduled to change. Event-driven simulation execution skips over time steps where no events are scheduled and therefore system state does not need to be updated. It only processes those timesteps where events are scheduled to occur, and thus progresses at potentially irregular intervals compared to the time-driven approach. For this reason, an event-driven simulation can also be faster to execute than a corresponding time-driven simulation as not all timesteps require processing. Figure

2.1, adapted from (Fujimoto 2000) illustrates state updates in the time-driven and event-driven approaches.

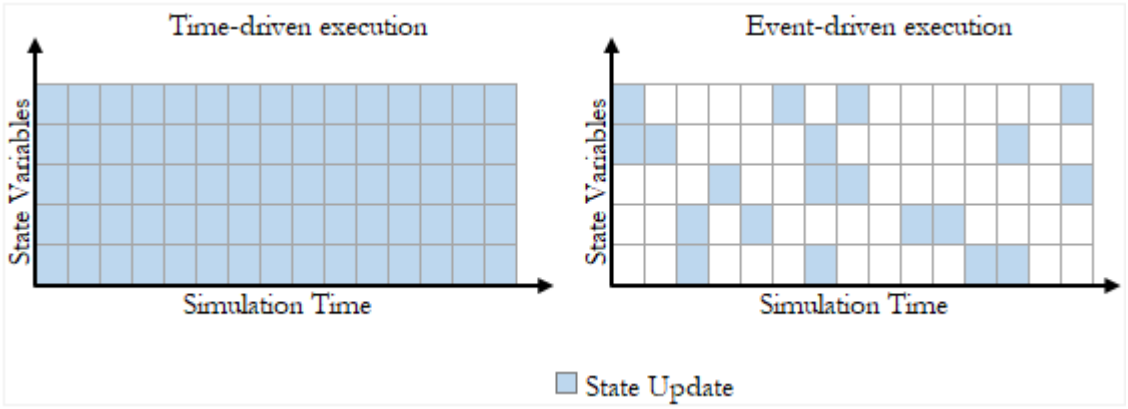

Figure 2.1: Time-driven and event-driven simulation execution
(Modified from source: Fujimoto 2000)

2.1.4 Live, Virtual and Constructive (LVC) Simulations

The LVC simulation classification framework, which originates from military simulation research, is useful for clarifying whether a simulation application involves real-world interaction or not. Depending on the degree of interaction between simulations and the real world, simulations can be classified as *live*, *virtual* or *constructive*.

2.1.4.1 Live Simulation

According to Hodson and Baldwin (2009), the term *live simulation* does not refer to simulation in the sense of computer simulations at all, but can more accurately be described as a rehearsal or dry run operation, in which actual human beings operate real-life equipment.

2.1.4.2 Virtual Simulation

The term *virtual simulation* includes simulations which involve either live humans and simulated equipment, or simulated humans and actual equipment. This class covers human-in-the-loop simulations such as flight simulators used for pilot training, hardware-in-the-loop simulations used for testing prototype equipment, and simulations of Cyber-Physical systems (Rajkumar *et al.*, 2010). These types of simulations are usually constrained to real-time execution to allow normal interaction with the live human or equipment.

2.1.4.3 Constructive Simulation

Unlike live simulations and virtual simulations, constructive simulations do not involve any real-world interaction with equipment or humans but consist entirely of simulated components.

Hodson and Hill (2014) note that the divisions in the LVC classification are not always clear and that it could benefit from the idea of the "reality-virtuality continuum" (Milgram and Kishino, 1994) which provides a more nuanced taxonomy for the degree of involvement between the real world and the computer simulation.

2.1.4.4 Research Scope

For clarity, the scope of the work in this thesis is limited to constructive as-fast-as-possible computer simulations. Therefore, no consideration is given to the possibility of any interactions with live humans or equipment during simulation execution. Admittedly, virtual real-time distributed simulations also have a need to manage communication efficiently, for example with the use of dead reckoning algorithms in training exercises conducted using distributed simulation (Lin, Blair and Woodyard, 1997). However, this type of simulation is not within the scope of the work covered in this thesis.

2.1.5 Simulation Modelling Paradigms

Multiple modelling paradigms exist for specifying simulation models. Each paradigm takes a different view of how to reduce a real system to a representative model and incorporates a separate set of techniques for mapping parts of the system to entities within the simulation model, defining the relationships between the model entities, representing system state, and directing the simulation execution flow. Simulation modelling paradigms that are widely used in various domains include System Dynamics (SD), Discrete Event Simulation (DES), and Agent-Based Simulation (ABS) (Borshchev and Filippov, 2004). Other paradigms commonly employed in the Urban Simulation context include Cellular Automata (CA) and Microsimulation Modelling (MSM). Although the paradigms mentioned in this section are not meant to form an exhaustive list, they provide a good overview of the main modelling approaches relevant to Urban Simulation. Some modelling paradigms are better suited for certain types of systems and may be easier to apply to a given system than other approaches. Also, some paradigms take a macro-level view

of systems while others are more suited for the micro-level view. The following sections provide brief descriptions of the modelling paradigms mentioned in this section.

2.1.5.1 System Dynamics

System Dynamics is a well-established simulation modelling paradigm rooted in the seminal work of Forrester on Industrial Dynamics (Forrester, 1968) and Urban Dynamics (Forrester, 1970). As explained by Kirkwood (1998), SD makes use of causal links and feedback loops to express the dynamic relationships that exist between system components. A causal link from X to Y specifies that a change in X causes a change in Y, and the nature of this change is represented mathematically using differential equations. A feedback loop results when X indirectly causes a change to itself through its influence on other parts of the system. For instance, if a causal link exists from X to Y, another from Y to Z, and yet another back from Z to X, a feedback loop has been established. SD models are normally presented conceptually with causal loop diagrams and stock-and-flow diagrams. The SD paradigm is mainly employed for continuous, deterministic, dynamic modelling at the macro-level of abstraction, and has been applied in a wide range of domains(Morecroft and Robinson, 2005).

2.1.5.2 Discrete Event Simulation

Discrete Event Simulation is a commonly used paradigm that is especially suited for modelling systems that consist of processes which can be represented as a network of connected queues. Models created with DES are normally discrete, stochastic, and dynamic (Morecroft and Robinson, 2005). Unlike most other modelling paradigms, DES particularly employs event-driven execution rather than time-driven execution. As explained by Schriber et al. (2016), a DES model is composed of *entities* that arrive in a system and queue to access limited *resources*, creating and responding to *events* while doing so. Events mark points in the model where system state changes. For example, the instants of arrival or exit of an entity, and the times when an entity starts or finishes using a resource are all marked by events. A DES simulation maintains an *event list* that determines the order in which events need to be processed. In some cases, processing one event will lead to scheduling other events. For instance, when an entity gains access to a resource that supplies a service, a future event is scheduled for the time when the service being provided to the entity is

due to end. The DES paradigm is described as process-oriented because it mainly focuses on representing system processes as queues. Because many business and industrial systems easily lend themselves to this way of thinking, DES is a popular method in those application domains.

2.1.5.3 Agent-Based Simulation

The Agent-Based Simulation paradigm is relatively novel compared to the more established SD and DES modelling paradigms (Siebers *et al.*, 2010). However, it has gained popularity over time, especially for modelling complex systems behaviour (Heath and Hill, 2010). As explained by Macal & North (2015), an ABS model is composed of individual autonomous *agents* that interact with each other and with their *environment*. These dynamic interactions can cause patterns to arise in system behaviour, which is called *emergent* behaviour as such behaviour is not explicitly defined by the model. An agent makes autonomous decisions concerning its own actions, and for this it may consider its own present state, the states of other agents, and the state of its environment. A frequently cited early example of ABS is the Boids artificial life simulation (Reynolds, 1987), which mimics the synchronized flight pattern of a flock of birds by treating each bird as an individual autonomous agent that adjusts its own flight path to fit in with its closest neighbours. As a result, a complex collective flocking pattern emerges which would be difficult to define directly as system behaviour. As the ABS paradigm defines micro-level behaviours that result in macro-level patterns, it is considered a *bottom-up* modelling approach. ABS has been applied in a wide range of domains including Economics (Zhang, Siebers and Aickelin, 2012) and Sociology (Schelling, 1971) (Epstein and Axtell, 1996). The Multi-Agent Simulation (MAS) paradigm (Wooldridge and Jennings, 1995) and the ABS paradigm are similar in their approach, and the terms are sometimes used interchangeably. While ABS is normally associated with research work related to the social sciences, MAS research work usually belongs to the field of Artificial Intelligence. Although the focus and goals of the research areas might differ, MAS and ABS share a similar view of modelling and employ similar language and techniques.

2.1.5.4 Cellular Automata

The Cellular Automata modelling paradigm is older than the ABS paradigm. It may, however, be described as a restricted form of ABS that makes use of a two-

dimensional grid of cells to represent a spatial environment, with agents residing in square grid cells. Rules defined for the CA model determine how cells interact with neighbouring cells to update the system state. Two of the most common ways for defining CA neighbourhoods are the Moore neighbourhood and the von Neumann neighbourhood. In the Moore neighbourhood, each cell considers all eight cells surrounding it as neighbours. On the other hand, the von Neumann neighbourhood defines neighbour cells as those that share an edge, resulting in four neighbours per cell. Wolfram (2002) investigates various rules for CA models in some detail and compares the emergent patterns to various natural phenomena. Belying their apparent simplicity, CA models have been used to model some complex systems such as traffic flow (Nagel and Schreckenberg, 1992).

2.1.5.5 Microsimulation Modelling

The Microsimulation modelling paradigm is rooted in the seminal work of Orcutt (1957). Similar to the ABS paradigm, MSM models view a system as comprising of micro-level decision-making entities. However, as explained by Birkin and Wu (2012), MSM places specific emphasis on fitting the properties and behaviour of the population of decision-making entities to reliable statistical data, resulting in a model that can be used as a basis for forecasting future population states through simulation experiments. MSM is a frequently used paradigm for investigating the effect of public policy on a population, and has been applied widely in Economics (Bourguignon and Spadaro, 2006).

2.1.5.6 Paradigm Comparison

Figure 2.2 provides an overview of the properties of the modelling paradigms that have been discussed briefly in the previous sections. The figure illustrates the normal characteristics of the paradigms, although work has been done to extend them. For example, Meyer (2014) proposes a framework for using event-driven execution in ABS.

	Precision		Randomness		Timing		Time Mechanism		Orientation	
Paradigm	Continuous	Discrete	Deterministic	Stochastic	Static	Dynamic	Time-Driven	Event-Driven	Process	Individual
Agent-Based Simulation		■		■		■	■			■
Multi Agent Simulation		■		■		■	■			■
Microsimulation		■		■		■	■			■
Discrete Event Simulation		■		■		■		■	■	
Cellular Automata		■		■		■	■			■
System Dynamics	■		■			■	■		■	

Figure 2.2: Overview of modelling paradigms

2.2 Large-Scale Simulation Methods

Parry and Bithell (2012) provide an overview of alternative approaches for executing large-scale simulations that require computing resources which exceed the memory or processing capacity of a reasonably resourced stand-alone computer. The approaches for enabling large scale simulation include *model simplification*, *modelling paradigm shift*, *vertical resource scaling* and *horizontal resource scaling*.

2.2.1 Model Simplification

Model simplification aims to reduce the scale of the model by lowering the fidelity of model contents to the extent that will permit the simulation to be executed in reasonable time on the available computing resources. One way to achieve simplification is to reduce the number of entities involved in the model to a number that can be managed within the available computing resources. Another simplification method involves the use of a single model entity to represent multiple system entities, aggregating them together into a "super" entity. Although model simplification methods provide a straightforward means for scaling down simulation models to fit within the limits of computing resources, such approaches may result in modelled behaviour that does not reflect that of the system being modelled with sufficient accuracy.

2.2.2 Modelling Paradigm Shift

This approach involves converting a simulation model from one paradigm to another that requires fewer computing resources. For example, a model created using a paradigm with a micro-level view can be converted to an equivalent macro-level model. This approach assumes that such a conversion is feasible and that the outputs from both models will be equivalent. If these assumptions are true, then the essence of this process is the translation of the simulation model in question into a more suitable modelling paradigm. These assumptions do not hold where the behaviour produced by the original model cannot readily be replicated using the target paradigm. For example, emergent behaviour from collective individual actions in a micro-level model may not be feasible to replicate using macro-level approaches.

2.2.3 Vertical Resource Scaling

Vertical resource scaling refers to upgrading the resources of a single computing node to enable the execution of larger workloads. Vertical scaling includes actions such as increasing the amount of installed memory and replacing the processor with a faster one. Where applicable, the advantage of this approach is that no changes need to be made to the simulation model. In essence, a more powerful computer is simply made available to run the simulation. However, the application of vertical scaling has physical limits. Processor speeds, for example, have stagnated in recent times (Sutter, 2005).

2.2.4 Horizontal Resource Scaling

In horizontal resource scaling, total computing resources are scaled up by adding more computing nodes to form a greater pool of resources which can be coordinated to execute the large-scale simulation. This is the approach used in Distributed Simulation, in which a large-scale simulation model is partitioned and distributed among the computing nodes in the pool which communicate with one another via an interconnecting network. Unlike vertical scaling, horizontal resource requires changes to be made to the structure of the stand-alone simulation.

Of the four different approaches discussed in this section for handling large-scale simulations, horizontal resource scaling is one of the most generally applicable for executing large-scale models without losing model fidelity and offers flexibility for

scaling up total computing resources beyond the physical limits of a stand-alone node.

2.3 Distributed Simulation

The research field of Parallel and Distributed Simulation is concerned with the use of computer systems composed of multiple interconnected processors to execute simulation experiments. Although both the terms *Parallel Simulation* and *Distributed Simulation* have this concern in common and are used interchangeably, they are sometimes employed individually to indicate different primary concerns. Fujimoto (2016) explains how the use of these two terms originated from the development of the research field from groups with different perspectives. The *Parallel Simulation* research perspective primarily focused on methods for accelerating the execution of large-scale homogeneous simulations by dividing the workload between communicating processors. *Distributed Simulation* research, on the other hand, was principally interested in establishing interoperability between heterogeneous simulations. This would enable existing simulation models to be reused in combination with other models and would reduce the need for re-implementing existing functionality when simulation studies with new questions can be answered by re-using existing models in a co-ordinated manner. Fujimoto (2000) points out that the early work in Parallel Simulation was mainly from the scientific High-Performance Computing community, emphasizing techniques for efficient parallel simulation execution, while Distributed Simulation work was mainly carried out in military research which focused on applications such as enabling joint training simulation exercises to be performed by geographically distributed participants.

The computing platforms considered by each research perspective for achieving its primary goals also differ. Parallel Simulation tended to rely on High-Performance Computing (HPC) computing clusters composed of computing nodes connected by high-speed, low latency interconnects in order to run as-fast-as-possible simulations efficiently. Distributed Simulation, on the other hand, had to cover needs such as enabling real-time collaboration between members of a geographically dispersed team and therefore had to consider constraints involved in the use of geographically dispersed computing nodes connected by unreliable networks having low data

transmission rates and high latency. The use of parallel and distributed simulation is also a concern in other areas such as the entertainment industry where it is used in massively multiplayer online games (Improbable, 2020) and in films for rendering CGI simulations.

In this work, the term Distributed Simulation (DS) is used as an all-inclusive term. The relevant goals are specified where they are not clear from the surrounding context. The two main goals for using DS in this work are simulation scalability and interoperability as explained in Section 1.1 based on the classification by Taylor (2019).

2.3.1 Strong and Weak Scaling

With regards to simulation scaling, the two main types are *strong scaling* and *weak scaling*. These two types refer to different goals for applying horizontal resource scaling.

Strong scaling refers to the case where additional computing nodes are added with the aim of accelerating the execution of a fixed size workload (Amdahl, 1967).

Weak scaling, on the other hand, refers to the case where more computing nodes are added as the size of the workload increases in order to complete execution of the larger workload in the same time required by the smaller workload (Gustafson, 1988).

In both cases, distributing the workload over multiple computing nodes makes it possible to execute large workloads that cannot be managed within the resources of a single stand-alone computing node.

2.3.1.1 Horizontal Scaling Measurements

Two of the main measurements for evaluating the performance obtained from horizontal scaling are *speedup* and *efficiency* (Karp and Flatt, 1990).

Speedup is defined as:

$$Speedup = \frac{T_1}{T_N}$$

Where T_1 is the time taken to complete the workload sequentially on a stand-alone computing node, and T_N is the time required to process the workload in parallel using N computing nodes.

Efficiency is defined as:

$$Efficiency = \frac{Speedup}{N} = \frac{T_1}{N\,T_N}$$

In the ideal case, efficiency is 1 which indicates that the processing resources on each computing node is fully utilized.

2.3.1.2 Strong Scaling

Using N nodes for a problem of fixed size, the maximum possible speedup that can theoretically be achieved is N, in which case scaling up to N nodes produces ideal *linear speedup*. However, the limits of strong scaling are described by Amdahl's Law (Amdahl, 1967) which reasons that for a fixed computational workload with a sequential portion that cannot be distributed, the execution time required for the sequential portion represents a lower bound on the total execution time for the entire workload. This places an upper bound on the speedup, regardless of how many computing nodes are added to share the workload.

Amdahl's Law is summarized as follows:

$$T_1 = f\,T_1 + (1 - f)\,T_1$$

Where f is the *Amdahl fraction*, the fraction of the workload that is inherently sequential and cannot be distributed.

$$T_N = f\,T_1 + \frac{(1 - f)\,T_1}{N}$$

$$Speedup = \frac{T_1}{T_N} = \frac{1}{f + \frac{(1-f)}{N}}$$

According to Amdahl, this provides an upper bound on the speedup possible, with the maximum achievable speedup being $\lim_{N\to\infty} Speedup = 1/f$.

2.3.1.3 Weak Scaling

The potential of weak scaling is described by Gustafson's Law (Gustafson, 1988). Contrary to Amdahl's Law, Gustafson's Law does not assume a fixed problem size

and proposes that the motivation for increasing the number of computing nodes depends on the practical need to process larger workloads in the same amount of time as the original workload. In place of a fixed problem size, Gustafson's Law assumes a fixed execution time and scales up the number of nodes with the aim of maintaining a constant execution time as the workload grows larger. As the inherently sequential portion of many practical problems does not necessarily grow with problem size, there is the additional potential for exploiting more parallelism from the workload structure as the problem size grows larger. Therefore, Gustafson's Law assumes a fixed size for the inherently sequential workload rather than assigning it a fixed fraction of the total workload.

Gustafson's Law is summarized as follows:

Let f_{scaled} be the scaled sequential fraction of the workload. Unlike the Amdahl fraction, f_{scaled} depends on the size of the problem. As the problem size grows and N is increased to accommodate the larger workload, f_{scaled} decreases. The execution time T_N remains constant as N scales with problem size.

T_1 is the time it would take to complete a given workload sequentially on a stand-alone computing node instead of parallel execution on N nodes:

$$T_1 = f_{scaled} T_N + N \times (1 - f_{scaled})\, T_N$$

Therefore,

$$Scaled\ Speedup = \frac{T_1}{T_N} = N + (1 - N)\, f_{scaled}$$

Gustafson's Law shows that for practical problems where f_{scaled} diminishes as problem size grows, the upper bound on speedup imposed by the inherently sequential workload is not as low as the strict view held by Amdahl's Law.

Snyder (1986) provides an analysis of weak scaling, determining how N varies with problem size if a constant execution time is maintained. For a given problem of size n which exhibits a time complexity of $O(n^x)$,

$$T = cn^x$$

Where T is the sequential execution time on a single computing node.

Assuming that the entire workload can be executed in parallel and the inherently sequential workload is negligible, linear weak scaling applies. As problem size increases, a constant execution time T can be maintained by scaling up resources to N computing nodes. If problem size increases by a factor of m,

$$T = \frac{c(mn)^x}{N}$$

The number of computing nodes N required to maintain a constant execution time is then

$$N = m^x$$

According to Snyder (1986), N grows at a rate similar to the time complexity of the problem. This is an important consideration for determining the resources required for weak scaling in various types of large-scale simulation problems as their model configurations grow large.

In this section, the analyses presented for speedup in both strong and weak scaling does not account for the communication and synchronization overheads associated with distributed simulation. Communication and synchronization costs comprise additional overheads that are not included in the computational workload. These overheads can increase with the number of nodes, N, and can present a significant performance bottleneck in distributed simulations where exchanging messages between computing nodes is necessary to ensure correct execution.

This work focuses on the weak scaling approach and not on strong scaling, as the primary goal is to enable the execution of large simulation workloads by horizontal resource scaling while managing communication between computing nodes efficiently.

2.3.2 Task and Data-Parallelism

As Lin & Snyder (2009) explains, two general strategies for performing parallel computations are the *task-parallel* and the *data-parallel* approaches. With the task-parallel approach, the total workload is divided among processors such that each processor is entrusted with a different kind of computation from the others. With the data-parallel approach, on the other hand, all processors perform the same kind of

computation at the same time, but each performs it on a different subset of data from the others. Both the task-parallel and data-parallel approaches have been used in the parallel simulation literature, as discussed in the following sections.

2.3.2.1 Task Parallelism

Much of the early research literature on Parallel Simulation that adopts the task-parallel approach is from the scientific High-Performance Computing community working on Parallel Discrete Event Simulation (PDES). This large body of work is concerned with distributing Discrete Event Simulations (DES) over multiple computing nodes for parallel execution. The PDES research literature proposes various techniques for efficient synchronization between nodes. The techniques seek to exploit as much parallelism as possible by allowing each node to advance independently of the others wherever possible, while ensuring that simulations execute correctly by processing all events in the right order. Comprehensive surveys conducted on the PDES research field over the years include those of Righter and Walrand (1989), Fujimoto (1990), Ferscha and Tripathi (1998), Fujimoto (2000), Perumalla (2006), Jafer, Liu and Wainer (2013), and Fujimoto (2015). Although the PDES field primarily focuses on the DES paradigm which relies on event-driven execution, the techniques developed from PDES research has also been applied to other paradigms such as Multi-Agent Simulation (Theodoropoulos and Logan, 1999).

2.3.2.2 Data Parallelism

In recent years, much attention has been paid to using data-parallel methods for performing computer simulations using GPU hardware. As noted by Huang et al. (2008), the reasons for this interest include the stagnation of CPU clock in recent times (Sutter, 2005) and the fact that GPUs incorporate a larger number of processing cores than CPUs. Also, GPUs are cheaper and easier to set up and manage compared to a cluster of interconnected CPU nodes with a similar number of processing cores. The interest in data-parallel simulation using GPUs has also been helped by recent development of frameworks for convenient General-Purpose computing on GPUs (GPGPU), including CUDA and OpenCL. Such frameworks enable programmers to more easily harness the high data-parallel processing throughput that GPUs provide for computations that are not related to computer graphics.

GPUs are able to combine large numbers of processing cores on a single chip because individual GPU cores are relatively simple and slow compared to CPU cores and therefore are able to take up less space. Although individual GPU cores are slower than CPU cores, combining them in large numbers to perform data-parallel tasks provides a higher processing throughput than can be achieved by CPUs which have a comparatively small number of cores.

The GPGPU programming model involves an arrangement which includes a CPU and an attached GPU co-processor. The CPU performs sequential tasks and offloads large data-parallel tasks to the attached GPU co-processor to accelerate execution. Many modern high-performance computing platforms include GPU co-processors, including several in the Top500 (Top500, 2020).

Prior to the introduction of GPGPU frameworks such as CUDA and OpenCL, research efforts towards using GPUs for simulation relied on techniques to repurpose the GPU's graphics programming capabilities to perform general computing (Owens *et al.*, 2006). For example, using such techniques Lysenko & D'Souza (2008) created a framework for executing agent-based simulations on GPUs, reporting a speedup of 1000 compared to corresponding sequential execution using mature ABS toolkits and demonstrated the capability of their framework to handle more than 2,000,000 agents on the Sugarcape model of Epstein and Axtell (1996). The GPGPU approach has been shown to produce significant speedup in large-scale simulation case studies. Work on ABS simulation using GPGPU frameworks includes Chen et al. (2015) who use CUDA to implement a simulation of evacuation scenarios showing a speedup of 38 for 8000 agents, and Ho et al. (2015) who execute an ABS over multiple GPUs, achieving speedups greater than 180.

To enable execution on GPU co-processors, simulation workloads need to be re-formulated as data-parallel computations in order to take advantage of the resources GPUs provide. However, re-working simulations to fit in with the data-parallel execution strategy is not straightforward. Tools such as FLAME-GPU (Richmond and Chimeh, 2017) for ABS help to address this issue by providing a framework for specifying simulation models such that the required computations can be mapped to data-parallel execution on GPU hardware. Using FLAME-GPU, Richmond (2015) reports a speedup of 250 for an experimental model with over 130,000 agents which

includes features such as agent movement within an environment and detection of nearby agents.

2.3.2.3 Research Scope

Although off-loading large data-parallel workloads from the CPU to the GPU for faster processing can be a useful approach for some problems, this work focuses on the task-parallel approach which is more generally applicable to Distributed Simulation.

2.4 Parallel Discrete Event Simulation

2.4.1 Logical Process Decomposition

The Parallel Discrete Event Simulation (PDES) literature, rooted in the seminal work of Chandy and Misra (1979), approaches parallel DES by decomposing a DES simulation model into a set of *Logical Processes* (LPs). LPs are assigned to separate interconnected processors which will execute the LPs in parallel while exchanging the messages necessary for ensuring overall simulation correctness. Fujimoto (2000) points out that a DES can be decomposed into LPs in two different ways. One method is the process-oriented approach, where each LP is used to represent one of the various processes that constitute the DES model. Each process is responsible for a distinct subset of the system state. The process-oriented decomposition approach is the one generally taken in the PDES literature. The second approach is an alternative decomposition method in which the simulation run length is divided into multiple blocks of time, with each block being a separate LP. While the second approach proposed is simpler than the first and would require fewer messages to be exchanged between LPs during simulation execution, it is not generally applicable. However, time decomposition would work for particular applications where the system states at the start and end of the specified time block can be predicted accurately. Figure 2.3, adapted from (Fujimoto, 2000), illustrates the two approaches to LP decomposition discussed in this section.

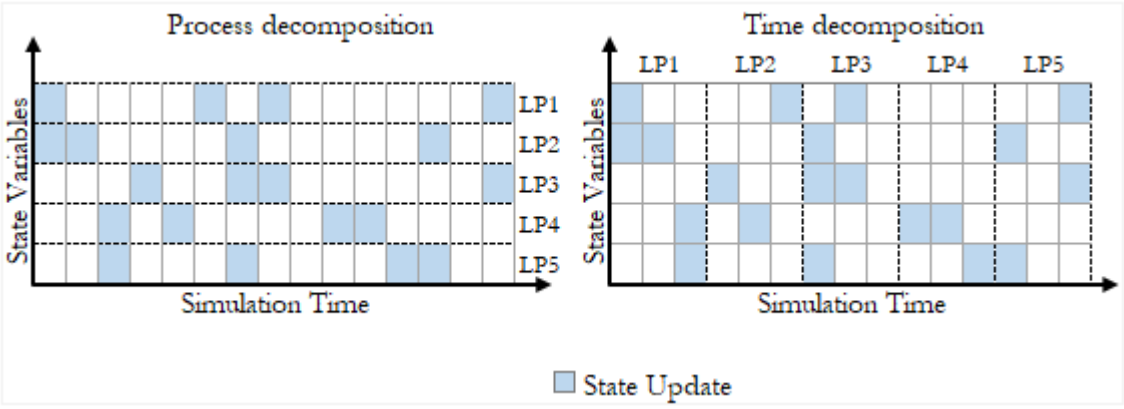

Figure 2.3: Process and Time LP decomposition methods
(Adapted from source: Fujimoto, 2000)

2.4.2 LP Time Synchronization

A large part of PDES research work is dedicated to time synchronization algorithms, which includes several proposed methods to coordinate concurrently executing LPs, ensuring that the distributed execution produces correct results by following the same execution path that a corresponding sequential simulation would follow. Time synchronization algorithms attempt to exploit parallelism in the distributed execution by allowing LPs to execute independently of one another as much as possible.

As set out by Chandy and Misra (1979), LPs communicate during distributed simulation execution by sending each other timestamped messages which contain events to be processed by the receiving LP. If at any time an LP receives a new event that should have been processed before another event that the LP has already processed, the simulation execution is considered invalid because a causal relationship may have been broken. Time synchronization algorithms achieve correctness by ensuring that all events are processed in the same order as they would have been processed in a sequential execution, making certain that causal relationships between events are respected. Fujimoto (1990) explains that in order to achieve correct parallel execution, it is sufficient for each LP to process its own events in non-decreasing timestamp order. This is referred to as the *local causality constraint*. Peschlow and Martini (2007) point out that simultaneous events which have the same timestamp need to be adequately handled by deterministic tie-breaking rules consistent with sequential execution in order to guarantee correctness.

Although observing causal order guarantees correctness, it is not always required in all cases. This is because the timestamp order of two events does not necessarily imply a causal relationship between those two events. An event that occurs later in simulation time may be completely independent of one that comes before in terms of the aspects of system state which they have an effect on. Therefore, restricting event execution order too rigorously by observing event timestamp order too strictly can limit the independent execution of LPs and result in less exploitable parallelism. In spite of this, the local causality constraint remains necessary as a generally applicable method for ensuring distributed simulation correctness without incorporating application-specific information into time synchronization algorithms.

The PDES literature covers two main classes of time synchronization algorithms: *conservative* algorithms, and *optimistic* algorithms. On one hand, conservative algorithms always respect the local causality constraint and therefore prevent events within the same LP from executing out of timestamp order. On the other hand, optimistic algorithms allow the local causality constraint to be violated but provide a means for correcting out-of-order event execution after it has been detected.

2.4.3 Conservative Synchronization

Chandy, Misra (1979) and Bryant (1977), independently presented the first conservative synchronization algorithm for PDES, known as the Chandy/Misra/Bryant (CMB) algorithm. In their proposed scheme, DES LPs execute in parallel, each advancing in time by processing the next event on its event list that has the smallest timestamp. When an LP processes an event, it may result in a need to schedule future events for itself or for other LPs. If the scheduled event needs to be processed by another LP, it sends the other LP a message containing the scheduled event. In the CMB scheme, each LP is restricted from processing its next event until it can guarantee that none of the other LPs will send it any subsequent events that may have a smaller timestamp. This guarantee is obtained in the following manner: since an LP, labelled LP_1, cannot schedule events in its past, other LPs can rely on the timestamp of the latest message they received from LP_1 as a lower bound on the timestamp of any events LP_1 will send in the future. Although this provides the needed safety guarantee, the CMB algorithm recognizes that it can result in a deadlock situation where all the LPs block, each waiting for one

of the others to send a message so that a safety guarantee can be inferred from its timestamp. The CMB algorithm prevents such deadlocks by requiring each LP to send a *null message* to the other LPs each time it advances in simulation time and processes an event. When an LP sends a null message to the others, it also specifies a future time horizon within which it can guarantee that it will not send any further events. This time horizon, known as *lookahead*, varies depending on the simulation application. Larger lookahead values mean that more freedom is allowed for parallel LP execution and LPs can execute independently of each other for longer periods without the need to exchange any messages. Chandy & Misra (1979) show that this method is guaranteed to prevent deadlocks from occurring between LPs. In general, conservative synchronization algorithms follow this pattern of generating safety guarantees to refrain from violating the local causality constraint. The size of lookahead is a principal factor in the performance of conservative synchronization algorithms. This implies that for conservative algorithms to be used efficiently, they need to incorporate specific information about the simulation application in order to adequately exploit the potential parallelism available.

Research work on conservative synchronization algorithms is mainly concerned with proposing efficient means to handle deadlock and maximize lookahead. With regards to deadlock handling, conservative algorithms either prevent deadlocks, as with the CMB algorithm, or try to detect and break them. Deadlock-avoiding conservative algorithms generally employ the method of sending null messages and specifying a lookahead value. As noted by Jafer et al. (2013), such conservative algorithms usually need to deal with a substantial amount of null message overhead during simulation execution. Several strategies have been proposed to reduce the volume of null message communications, for example by sending null messages only on-demand (Misra, 1986). Deadlock-breaking conservative algorithms first allow a deadlock to occur and then attempt to break it. This class does not need to send null messages regularly. For example, in the approach of Chandy & Misra (1981), a controller process breaks deadlock by identifying the events with the smallest global timestamp, and instructing the LPs processing those events to resume execution because it is safe for them to proceed.

2.4.4 Optimistic Synchronization

D. R. Jefferson and Sowizral (1982) and D. R. Jefferson (1985) introduced the first optimistic synchronization algorithm, called the Time Warp Operating System (TWOS). Unlike the conservative approach, the optimistic approach does not require any safety guarantees from LPs in order to advance simulation time. Consequently, optimistic algorithms allow events to potentially be processed out of causal order. However, to correct out-of-order execution and observe the local causality constraint, optimistic algorithms provide a mechanism for *rollback*. Events that are received out of timestamp order are labelled *straggler* events. An LP detects that an event it receives is a straggler if the LP has already processed another event with a lower timestamp. When an LP detects a straggler event, it deduces that the local causality constraint has been violated. The LP then proceeds to initiate the rollback mechanism to revert to the most recent historical state where it was safe to process the straggler event without violating local causality. As part of the rollback process, the effect of all previous messages sent from the LP to other LPs after the established safe point must be undone. Time Warp handles this by using *anti-messages*, which are sent to negate specific messages that have previously been dispatched. When an LP receives an anti-message, it discards the corresponding message if it has not already processed it. If the corresponding message has already been processed, then the receiving LP must initiate its own rollback procedure to return to a safe point in time before the original message was processed. This leads to a cascading sequence of rollbacks that undoes all computation until a globally safe historical state has been reached where local causality has not been violated by any of the LPs.

The rollback process requires that the historical states of the simulation execution should be stored in memory. The Time Warp mechanism uses the concept of Global Virtual Time (GVT) to track the lower bound of simulation time across all LPs. At any point, GVT considers the current simulation time reached by each LP as well as the timestamps of undelivered messages still in transit. GVT is useful for the process known as *fossil collection*, by which historical states earlier than GVT are discarded to free up memory. This is safe because it is not possible for any LP to schedule a new event that occurs before GVT and therefore rollbacks cannot revert to a time before GVT.

The research work on improving the performance of optimistic algorithms is largely focused on decreasing the number of cascading rollbacks and reducing the amount of memory required for storing historical states. Some approaches for improving memory efficiency include techniques for using memory sparsely or salvaging memory (Jefferson, 1990), and techniques for using reverse computation instead of saving historical states (Carothers, Perumalla and Fujimoto, 1999). Reverse computation is not applicable in cases where computations can only be performed in one direction. Optimistic algorithms suffer from significant performance degradation in cases where one rollback causes an avalanche of subsequent cascading rollbacks, destabilizing simulation execution. Methods to counter this include throttling the degree of optimism by using a global execution window (Tay and Teo, 2001), and lazy event cancellation (Lin and Edward, 1991) which delays anti-messages until rolled back events have been re-processed and confirmed to produce different results from their earlier out-of-order execution.

2.4.5 Scalability of Conservative and Optimistic Approaches

Several software systems have been created to support developing PDES, including the μsik micro-kernel (Perumalla, 2005), Rensselaer's Optimistic Simulation System (ROSS) (Carothers, Bauer and Pearce, 2002), the Georgia Tech Time Warp (GTW) system (Das *et al.*, 1994) and the Time Warp Operating System (TWOS) (Jefferson *et al.*, 1987). These provide APIs that ease the implementation and execution of PDES using conservative and optimistic synchronization methods.

The Parallel Hold (PHOLD) model (Fujimoto, 1990b) is widely used for testing the performance of time synchronization algorithms. PHOLD is a parallel DES model in which the time between scheduled events on each LP is drawn from a given probability distribution. When processing one event results in scheduling a new event, the LP responsible for processing the new event is drawn from a uniform distribution. This makes the PHOLD model useful as a general artificial benchmark which is not based on any specific application.

PDES conservative and optimistic methods have been applied to several large-scale simulation applications, enabling the execution of such large-scale simulations in High-Performance Computing (HPC) environments. Neither the conservative nor the optimistic synchronization approach performs better than the other in all cases. The

relative performance of the time synchronization approaches depends on the specific simulation application. Each time synchronization approach possesses characteristics that make it better suited than the other for specific types of applications. On the one hand, conservative algorithms depend on high lookahead values for good performance. However, determining optimal lookahead values requires knowledge of application-specific information. On the other hand, optimistic algorithms are more general and do not require application-specific information such as look-ahead. However, optimistic algorithms can suffer from cascading rollbacks leading to poor performance. Carothers and Perumalla (2010) show that conservative algorithms perform poorly on applications with small lookahead values, and optimistic algorithms are more efficient at exploiting parallelism in this type of application. However, optimistic algorithms require more memory resources due to the need to store historical states in case rollback becomes necessary. Also, optimistic algorithms cannot be used in applications where operations cannot be rolled back once they have been completed.

Fujimoto (2015) surveys the results of several simulation experiments using PDES techniques on high-performance computers from 2003 to 2013 including:

- Fujimoto et al. (2003): 1,536 processors, ~200 million events per second
- Perumalla (2007): 16,384 processors, ~500 million events per second
- Bauer et al. (2009): 65,536 processors, ~12 billion events per second
- Barnes et al. (2013): 1,966,080 processors, ~500 billion events per second

To compare the results between these sets of experiments, their performance is measured as the number of events per second per processor:

- Fujimoto et al. (2003): ~130,000 events/s/processor (conservative)
- Perumalla (2007): ~30,000 events/s/processor (conservative, PHOLD)
- Bauer et al. (2009): ~180,000 events/s/processor (optimistic, PHOLD)
- Barnes et al. (2013): ~250,000 events/s/processor (optimistic, PHOLD)

These experiments demonstrate the feasibility of employing distributed computing systems to enable the execution of large-scale DES models that would not be possible to execute sequentially using the memory resources of a single stand-alone computing node.

2.5 Other Distributed Simulation Approaches

As discussed in previous sections, much research work has been conducted on approaches to enabling the efficient execution of large-scale PDES simulations. An evident limitation is that the methods proposed are developed specifically for the DES paradigm which uses the event-driven time flow mechanism. Other simulation paradigms that use the time-driven mechanism will need to have these methods adapted appropriately. Another limitation is that work on PDES does not account for aspects regarding enabling interoperability between heterogeneous simulations. This is natural as the primary focus of PDES research is on efficiently scaling of homogenous DES simulations. Therefore, the need to coordinate existing simulations together to answer new questions is not a priority.

2.5.1 PDES-MAS

As discussed in previous sections, the LP decomposition in PDES set out by Chandy and Misra (1979) assumes that each LP is responsible for a distinct portion of the system state that does not overlap with the portion of state managed by the other LPs. The portion of system state managed by each LP can only be modified by events which that LP itself processes. The essential idea, according to Chandy and Misra (1979), is that LPs *"cannot interfere with each other"*. Theodoropoulos and Logan (1999) extend ideas developed from PDES research to enable the distributed execution of MAS simulations. They point out that the view of a non-shared state is reasonable for a process-oriented paradigm such as DES where the communication topology between LPs is generally static and not expected to change during simulation execution. However, a non-shared state is not a sound assumption to make when considering an individual-oriented paradigm such as MAS where the topology of interactions between autonomous agents can be unstable and unpredictable. Unlike the processes in DES, the agents in MAS can interfere with each other either through direct interaction or indirectly through their shared environment. Theodoropoulos and Logan (1999) define *shared state* as the portion of system state to which multiple LPs have write or read access. This is analogous to a critical section in general concurrent programming and therefore read and write dependencies need to be observed carefully to guard against data hazards. They propose a PDES-MAS framework (Oguara *et al.*, 2007) which uses an optimistic synchronization approach and consists of three different types of LPs:

1. *Agent Logical Processes* (ALPs),
2. *Environment Logical Processes* (ELPs), and
3. *Communication Logical Processes* (CLPs):

In their proposed framework, CLPs manage the shared state and the communication between all ALPs and ELPs. ALPs and ELPs have different *spheres of influence*, the subset of shared state each ALP or ELP can read or write. The simulation workload is balanced by clustering ALPs and ELPs together based on their spheres of influence.

2.5.2 Distributed ABS Toolkits

Simulation toolkits have been created which provide libraries and a development environment to facilitate the process of implementing and executing distributed ABS. Popular distributed ABS toolkits include Repast HPC (Collier and North, 2012), D-MASON (Cordasco *et al.*, 2013), and FLAME (Coakley *et al.*, 2016). The toolkits include different approaches for accomplishing necessary tasks such as partitioning the model environment and agents into LPs, managing communication between agents in different LPs, and synchronizing time flow among all LPs to ensure simulation correctness. The following sections provide an overview of the approaches to these tasks used by various ABS toolkits.

2.5.2.1 Distributed ABS Partitioning Strategies

Model partitioning strategies attempt to optimize two objectives together

1. Balancing the computational workload evenly between LPs.
2. Minimizing the communication between agents in different LPs.

Cordasco, Spagnuolo and Scarano (2017) identify four broad strategies for ABS partitioning: load-based, space-based, relationships-based and space-relationships-based.

- Load-Based Partitioning:

  The load-based approach primarily focuses on optimizing the first objective, balancing the processing workload among LPs. This results in assigning equal numbers of agents to each LP without regard to where they may reside in the environment or their relationships with other agents. Each LP is responsible for performing the simulation computations on the specific subset

of agents to which it has been assigned. The load-based approach is a simple strategy that can be useful in cases where there are few relationships between agents and communication is minimal. Where there is significant communication in the ABS, it can lead to poor performance due to large volume of communication between LPs.

- Space-Based Partitioning:

  The space-based approach attempts to optimize both the balanced workload and minimal communication objectives by dividing the spatial environment into different partitions. Each partition is assigned to a separate LP which is responsible for simulating all the agents that reside in that partition. The space covered by partitions may be equally sized or unequally sized to balance the computational workload by having large-spaced partitions in parts of the environment where agents are sparse and small-spaced partitions in areas where agents are dense. This approach attempts to minimize communication between LPs based on the assumption that agents are more likely to interaction with other agents that are close in proximity and less likely to interaction with those that are far away. If this assumption holds, the partition borders can be drawn to put agents that are closely clustered together in the same partition and agents that are far apart from one another in different partitions. The effect of this is to reduce the communication between agents in different LPs by increasing local interactions. Partitions generated using this strategy may not remain optimal where agents can move around the environment space. One method to account for movement is to keep the partition borders static and re-assign agents from one LP to another when they move out of the space managed by one LP into the space managed by the other. Another method is to dynamically re-draw the partition borders as and when necessary.

- Relationships-Based Partitioning:

  The relationships-based approach attempts to optimize both objectives using the graph of relationships between agents where each agent is a vertex and an edge between two vertices indicates that an interaction relationship exists between the two agents. Vertices are assigned to partitions such that the

number of local edges is high and the number of edges between vertices in different partitions is low. This approach attempts to minimize communication by reducing the number of inter-LP agent relationships while balancing the workload by assigning each LP an equal number of agents. This approach is useful in models where interactions between agents are defined by explicitly assigned relationships and do not need to be inferred from spatial proximity. Relationship-based partitioning can be carried out using graph partitioning algorithms such as METIS (Antelmi *et al.*, 2015). Partitions created using this strategy may not remain optimal if the relationships between agents can change dynamically as the simulation progresses. One method for address this is dynamic re-partitioning to re-assign agents to LPs as and when necessary.

- Space and Relationships Based Partitioning:
  This approach is a combination of the space-based approach and the relationship-based approach. It is useful in cases where interactions between agents can occur either as a result of their proximity or due to explicitly defined relationships between the agents. One approach to partitioning in this situation is to add edges to the relationships graph based on the proximity of agents. Relationships-based techniques can then be applied to create the partitions.

In order to minimize communication, both the space-based approach and the relationships-based approach rely on the assumption that partitions can be created in such a way that dense inter-agent interactions can be localized to LPs, leaving sparse interactions for inter-LP communication. The two approaches are not effective in cases where this assumption of locality of interactions does not hold, for example in the extreme case of a fully connected ABS in which every agent can interact with all the other agents at any time.

2.5.2.2 Distributed ABS Communication

After partitioning the model, each LP has an *area of interest* which includes:

- Local agents assigned to the LP.
- The local spatial environment assigned to the LP.
- Non-local agents in other LPs with whom local agents can interact.
- The non-local spatial environment in other LPs with which local agents can interact.

Each LP is responsible for executing the simulation computations concerning its local agents and environment. One method for implementing interactions with non-local parts of the model is to let each LP keep local reference copies of the non-local agents and environment which fall within its area of interest. These reference copies are updated when their information changes by exchanging messages with the LPs responsible for simulating their originals. Figure 2.4 illustrates this approach with an ABS that has been spatially partitioned between four LPs. Each LP in the diagram maintains a copy of a portion of the non-local environment that borders on its own as well as copies of non-local agents located within those regions.

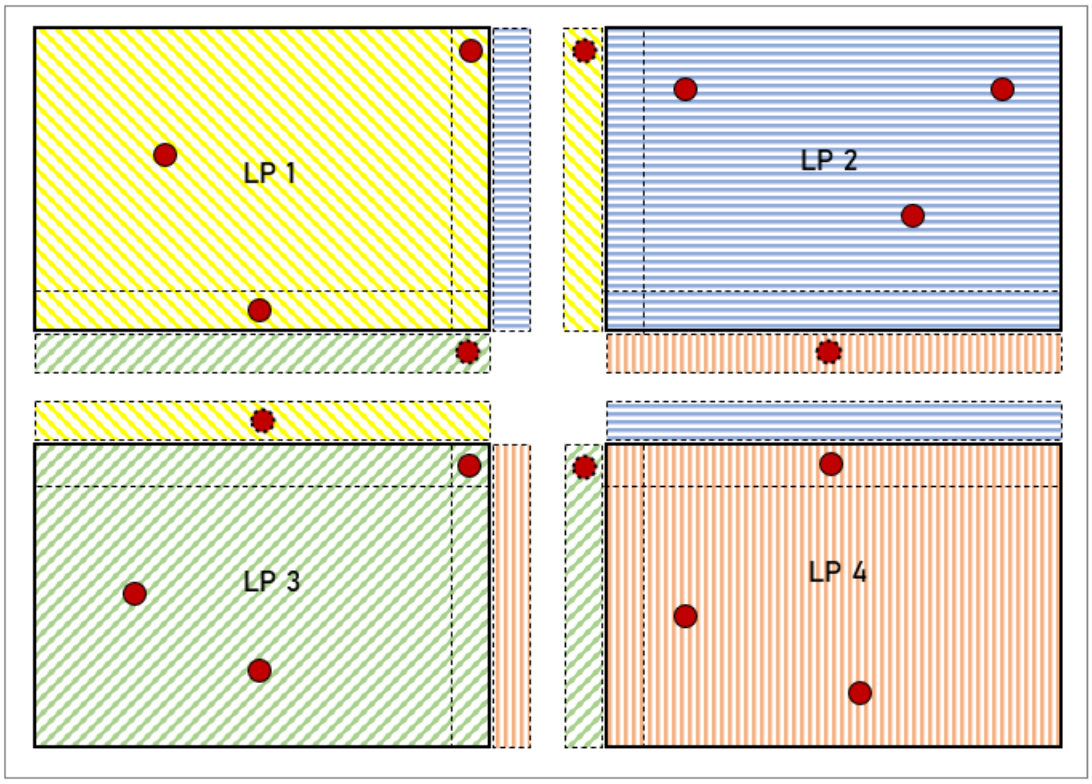

Figure 2.4: Areas of interest in a spatially partitioned ABS

Distributed ABS simulation toolkits generally exchange information between LPs by message passing. Updating the non-local data in LPs can be accomplished using this method. Communication strategies for updating non-local data are mainly based on the publish-subscribe pattern. Using the publish-subscribe approach, each LP subscribes to receive updates relevant to the non-local copies it keeps. At each timestep, all LPs publish updates concerning their local agents to the other LPs that require those updates. This may result in unnecessary communication if the published data has not changed since the last timestep. Hence this strategy can be made more efficient by publishing only data that has changed since the previous timestep. Some updates might still be strictly unnecessary because the subscribing LPs may not actually have any use for the updated data at the timestep when it is delivered. However, regular updates help maintain a coherent view of global data across all LPs and prevent errors in distributed simulation outputs.

2.5.2.3 Distributed ABS Time Synchronization

In distributed ABS simulation toolkits, time synchronization is generally conservative, and LPs are only allowed to move forward to the next timestep when it is safe to do so. In this case, safety is guaranteed in two steps: first, LP performs all their local computations and then exchange the resulting updated data. When each LP has received all updates relevant to its areas of interest, it is safe for it to advance to the next timestep. Generally, distributed ABS simulations move all LPs forward together in lockstep using a synchronization barrier at the end of each timestep.

Distributed simulation toolkits are useful and applicable to various types of simulation problems. The toolkits ease the process of implementing distributed simulations, providing the common functionality required for communication and time synchronization, and in some cases generating code from inputs such as state charts or XML. However, distributed simulation toolkits do not generally provide direct support for integration with other existing simulations that may or may not have been created using other tools. This limits the type of distributed simulation that can be created using this approach alone to a homogeneous simulation that uses one or more of the simulation paradigms supported by the toolkit being used.

2.5.3 Custom-Built Distributed Simulations

Apart from using toolkits, distributed simulation can also be custom-built by implementing the simulations in programming languages such as C++ or Java and making use of communication libraries such as the Message Passing Interface (MPI) (Gropp *et al.*, 1999), Java Message Service (Hapner *et al.*, 2002) and JGroups (JGroups, 2020). The custom-built approach is used for building specialized simulation software for particular problems in specific domains, as well as for creating ad-hoc distributed simulations. Many examples of custom-built distributed simulations are found in the literature of scientific high-performance computing. For example, Buchholz, Bungartz and Vrabec (2011) use C++ and MPI to implement a distributed molecular dynamics simulation, Ouro *et al.* (2019) implement a distributed computational fluid dynamics simulation using MPI, Mostaccio, Suppi and Luque (2005) create a custom distributed fish school simulation using MPI, Komann, Kauhaus and Fey (2005) use MPI communication in a custom distributed Cellular Automata simulation, and Plesser *et al.* (2007) simulate a biological neural network using MPI on a cluster of computing nodes.

Message passing libraries such as MPI enable communication in distributed simulations by transmitting data packets containing messages over the interconnecting network between nodes. Message passing is the standard method for communication in distributed computing systems which do not share a common memory address space. Message passing libraries include functions which provide the programmer with various methods for sending and receiving messages between nodes, including point-to-point communication and collective communication such as broadcasting and multicasting. MPI is the most popular message passing library for high-performance computing applications and includes a rich and mature set of communication functions for a wide range of operations. Some of the main MPI functions include *MPI_Send*, *MPI_Recv*, *MPI_Bcast* and *MPI_Barrier* for sending, receiving, broadcasting and creating a synchronization barrier. Communication in MPI can be synchronous or asynchronous. In synchronous (or blocking) mode, the sender waits until the transmitted message has been received at the destination before it can proceed with further execution. In asynchronous (or non-blocking) mode on the other hand, the sender can continue along its execution path without waiting for the message to reach its destination.

The custom-built distributed simulation approach is aided by communication libraries such as MPI and this approach works well for specific problems that require ad-hoc solutions or for creating specialized software for a particular domain, especially in scientific high-performance computing. Although this approach provides fine-grained control for the software developer, it also comes with the added responsibility of manually implementing measures to avoid deadlock and the responsibility of implementing measures to ensure that time advances are correctly synchronized to avoid corrupting simulation results. This approach is also limited in the sense that the end-product cannot easily be integrated with other simulations in a heterogeneous distributed simulation, although this can be addressed by building support for a common simulation interoperability standard into the custom-built software.

2.6 Distributed Simulation Interoperability

This section provides an overview of two leading standards for enabling interoperability between heterogeneous simulations, the High Level Architecture (HLA) and the Functional Mock-Up interface (FMI).

2.6.1 The High Level Architecture (HLA)

The High Level Architecture (HLA) is a standard for distributed simulation which originated as a military specification designed to facilitate the coordinated execution of simulation models (Kuhl, Weatherly and Dahmann, 1999). As explained by Topçu and Oğuztüzün (2017), some earlier military standards upon which the HLA was founded include Simulation Networking (SIMNET) (Miller and Thorpe, 1995), Distributed Interactive Simulation (DIS) (Hofer and Loper, 1995) and the Aggregate Level Simulation Protocol (ALSP) (Wilson and Weatherly, 1994) which were focused mainly on virtual simulations for training exercises and therefore involved real-time interaction between humans in virtual environments. The first version of the HLA specification was published in 1998 by the US Defense Modeling and Simulation Office (DMSO) as HLA 1.3. The HLA was eventually adapted into an IEEE standard which is currently developed by the Simulation Interoperability Standards Organization (SISO). The first IEEE version was published in 2000 as

HLA 1516-2000, followed by a second version in 2010 known as HLA 1516-2010 or HLA-Evolved (IEEE, 2010a). The HLA standard provides a standard framework for integrating heterogeneous simulations and orchestrating their individual execution to create a joint distributed simulation. The framework provides a rich set of standard services including services for coordinating the exchange of data between the simulations and for regulating their advancement in simulation time with respect to one another. The wide range of services provided by the HLA are also useful for coordinating the distributed execution of a partitioned homogeneous simulation. Consequently, HLA can be useful both for interoperability between heterogeneous simulations and for scaling up homogenous simulations. To the latter end, for example, Lees *et al.* (2003) make use of the HLA for executing a distributed MAS simulation, and Minson and Theodoropoulos (2004) use the HLA for executing distributed simulations created with the Repast Simphony ABS toolkit (North *et al.*, 2013).

In HLA terminology, individual simulations are referred to as *federates* and the whole distributed simulation is referred to as a *federation*. The HLA standard also specifies a software component referred to as a *Run-Time Infrastructure* (RTI) which is responsible for coordinating the federation execution. The RTI controls all communication between federates and directs time synchronization in the federation execution. HLA federates exchange messages by means of HLA *objects* which persist throughout federation execution and HLA *interactions* which are transient events. HLA object classes and interaction classes function as structures for grouping attributes and provide a template for creating concrete instances of objects and interactions. However, HLA object classes and interaction classes and do not include any methods unlike normal OOP classes. An HLA Federation Object Model (FOM) is a federation-level document used to establish an agreement between federates regarding data involved in the federation execution, including object classes and interaction classes. Federates may also have a SOM (Simulation Object Model) to decentralize the definitions specific to individual federates. Figure 2.5 provides a high-level conceptual overview of the HLA, showing how the main components interact with one another.

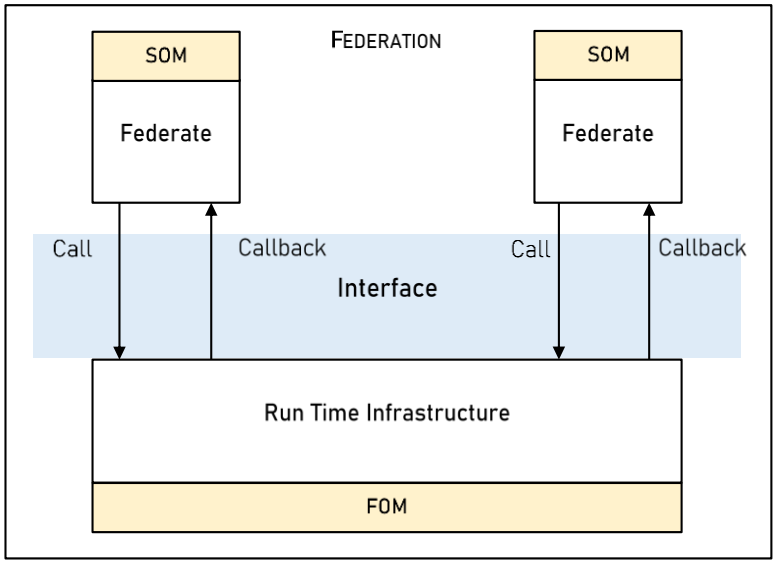

Figure 2.5: Overview of the HLA

The HLA standard is composed of three parts: *Framework and Rules*, *Interface Specification* and *Object Model Template (OMT)*.

2.6.1.1 The HLA Framework and Rules

The HLA Framework and Rules (IEEE, 2010a) provides an overview of the HLA framework, defining its terminology, describing its components and establishing how the components relate to one another. It sets out ten *HLA Rules* that describe the behaviour of federations and expected behaviour from well-behaved federates, including rules about communication and time management.

The first set of five rules describe HLA requirements for federations:

i. A federation's FOM must conform to the OMT specification.
ii. The RTI is not responsible for storing simulation objects, but individual federates are.
iii. All data exchanges between federates use the RTI as a conduit.
iv. All RTI–federate interactions must be done according to the Interface Specification.
v. Federates cannot share joint ownership of a single object instance attribute.

The second set of five rules describe HLA requirements for federates:

vi. A federate's SOM must conform to the OMT specification.

vii. A federate can send or receive attribute updates according to the details in its SOM.

viii. A federate can transfer or accept attribute ownership according to the details in its SOM.

ix. A federate can determine the conditions under which it is necessary for it to provide attribute updates according to the details in its SOM.

x. A federate manages its own local time but must be able to coordinate with other federates for communication and time synchronization.

2.6.1.2 The HLA Federate Interface Specification

The HLA Interface Specification (IEEE, 2010b) provides detailed definitions for several services that can be used by the RTI and federates to interact with each other and accomplish specific tasks. The services defined by the Interface Specification are grouped into seven classes. Each class of services includes a set of related calls that federates can use to interact with the RTI. Each service class also contains callbacks that the RTI uses to interact with federates. Each federate make calls to the RTI via an *RTI Ambassador* component. Federates also implement the necessary functions in a *Federate Ambassador* component to receive callbacks from the RTI. The seven service classes defined by the HLA Interface Specification are *Federation Management*, *Declaration Management*, *Object Management*, *Ownership Management*, *Time Management*, *Data Distribution Management* and *Support Services*.

1. **HLA Federation Management Services**

   This group includes services for managing federations and federates including functions such as:

   - Creating and destroying federations
   - Adding and removing federates from a federation
   - Managing federation barrier synchronization points
   - Saving and restoring federation state.

Figure 2.6 provides an overview of main services in the HLA Federation Management category.

Service Category	Spec #	Service Name	Direction
Federation Management	4.2	Connect	Federate -> RTI
Federation Management	4.3	Disconnect	Federate -> RTI
Federation Management	4.5	Create Federation Execution	Federate -> RTI
Federation Management	4.6	Destroy Federation Execution	Federate -> RTI
Federation Management	4.9	Join Federation Execution	Federate -> RTI
Federation Management	4.10	Resign Federation Execution	Federate -> RTI
Federation Management	4.11	Register Federation Synchronization Point	Federate -> RTI
Federation Management	4.15	Federation Synchronized	RTI -> Federate
Federation Management	4.16	Request Federation Save	Federate -> RTI
Federation Management	4.17	Initiate Federate Save	RTI -> Federate
Federation Management	4.19	Federate Save Complete	Federate -> RTI
Federation Management	4.20	Federation Saved	RTI -> Federate
Federation Management	4.24	Request Federation Restore	Federate -> RTI
Federation Management	4.27	Initiate Federate Restore	RTI -> Federate
Federation Management	4.28	Federate Restore Complete	Federate -> RTI
Federation Management	4.29	Federation Restored	RTI -> Federate

Figure 2.6: Overview of HLA Federation Management services

2. **HLA Declaration Management Services**

In the HLA, data exchange between federates is conducted according to the publish-subscribe pattern. Federates declare the types of data they wish to send and receive, and the RTI manages the routing of relevant updates to appropriate federates during federation execution. This arrangement ensures that federates are anonymous to one another. A subscribing federate need not know the identities of the other federates that publish the data it requires, and a publishing federate does not need to keep a list of other federates that must receive the updates that it publishes during federation execution. Only the RTI needs to have the global view of publishers and subscribers, receiving and forwarding updates as necessary.

The services in the Declaration Management group include functions to allow federates to inform the RTI regarding the types of object attributes and interactions they intend to publish or wish to subscribe to during federation execution. Figure 2.7 provides an overview of main services in the HLA Declaration Management category.

Service Category	Spec #	Service Name	Direction
Declaration Management	5.2	Publish Object Class Attributes	Federate -> RTI
Declaration Management	5.4	Publish Interaction Class	Federate -> RTI
Declaration Management	5.6	Subscribe Object Class Attributes	Federate -> RTI
Declaration Management	5.8	Subscribe Interaction Class	Federate -> RTI

Figure 2.7: Overview of HLA Declaration Management services

3. **HLA Object Management Services**

The services in the Object Management group are used for performing functions relevant to concrete instances of HLA objects and interactions including:

- Federates registering new object instances with the RTI.
- Federates discovering object instances that others have registered.
- Deleting object instances.
- Sending interactions and attribute value updates.
- Receiving interactions and (reflecting) attribute value updates.
- Specifying reliable or unreliable data transmission.

As prescribed in the HLA Rules, federates store the data related to simulation objects. Object instances must be registered with the RTI before their attributes updated to the federation. Before a federate can register an object instance with the RTI, it must first declare that it wishes to publish the relevant object class attributes. When a federate registers the object instances which it owns with the RTI, the RTI creates a *handle* which is a unique identifier for each object instance. The federate can then use the unique handles to refer to those specific object instances in future calls to the RTI. The RTI also forwards registered object instance handles to other federates that have subscribed to receive updates for the relevant object class attributes using the *Discover Object Instance* service. Figure 2.8 provides an overview of main services in the HLA Object Management category.

Service Category	Spec #	Service Name	Direction
Object Management	6.2	Reserve Object Instance Name	Federate -> RTI
Object Management	6.3	Object Instance Name Reserved	RTI -> Federate
Object Management	6.8	Register Object Instance	Federate -> RTI
Object Management	6.9	Discover Object Instance	RTI -> Federate
Object Management	6.10	Update Attribute Values	Federate -> RTI
Object Management	6.11	Reflect Attribute Values	RTI -> Federate
Object Management	6.12	Send Interaction	Federate -> RTI
Object Management	6.13	Receive Interaction	RTI -> Federate

Figure 2.8: Overview of HLA Object Management services

4. **HLA Ownership Management Services**

As prescribed by the HLA Rules, each object instance attribute can be owned by at most one federate. Only the owner federate will be granted permission to send the relevant attribute value updates for the object instance. When a federate registers a new object instance with the RTI, that federate is the default owner of the object instance attributes which it registers. However, ownership of object attributes can be transferred from one federate to another by means of the functionality provided through HLA Ownership Management services. Using these services, a federate may divest its ownership of an object instance attribute either unconditionally or on the condition that the RTI first searches for a suitable federate willing to take ownership. Only federates that have declared their intention to publish the relevant object class attribute can assume ownership of the divested object instance attribute. A federate may also make a request to acquire ownership of an object instance attribute that is already owned by another federate or that has previously been divested by another federate and has no owner.

Figure 2.9 provides an overview of main services in the HLA Ownership Management category.

Service Category	Spec #	Service Name	Direction
Ownership Management	7.2	Unconditional Attribute Ownership Divestiture	Federate -> RTI
Ownership Management	7.3	Negotiated Attribute Ownership Divestiture	Federate -> RTI
Ownership Management	7.4	Request Attribute Ownership Assumption	RTI -> Federate
Ownership Management	7.8	Attribute Ownership Acquisition	Federate -> RTI
Ownership Management	7.9	Attribute Ownership Acquisition If Available	Federate -> RTI
Ownership Management	7.11	Request Attribute Ownership Release	RTI -> Federate
Ownership Management	7.13	Attribute Ownership Divestiture If Wanted	Federate -> RTI

Figure 2.9: Overview of HLA Ownership Management services

5. **HLA Time Management Services**

An HLA federation execution proceeds according to a logical time axis. At any point during federation execution, each federate can be at a different logical time, coordinated by the RTI. Federates joined to a federation may be time-regulating, time-constrained, both or neither. Time-regulating federates can influence the flow of logical time in the federation execution by sending timestamped messages to the RTI. Federates that are not time-regulating can send only messages that do not include any timestamp information. Time-constrained federates can observe the flow of logical time in the federation execution by receiving timestamped messages from the RTI. Federates that are not time-constrained can receive messages that do not have any timestamp information, as well as messages that were originally timestamped but have had the timestamp information stripped away by the RTI.

In the HLA, federates exchange messages with one another by using Object Management services. Services such as *Update Attribute Values* and *Send Interaction* are used for sending messages, while services such as *Reflect Attribute Values* and *Receive Interaction* are used for receiving messages. Using these services, messages may be sent and received either in Time Stamp Order (TSO) or Receive Order (RO). In order to send a TSO message, the sending federate must be time-regulating and must supply a logical timestamp together with the message. In order to receive a TSO message, the receiving federate must be time-constrained. For subscribing federates that are not time-constrained, the RTI converts TSO messages into RO messages as appropriate.

Federates move forward in logical time by requesting permission from the RTI using Time Management services such as *Time Advance Request* (TAR)

and *Next Message Request* (NMR). The RTI responds to such requests by first delivering any necessary TSO messages. When all TSO messages have been delivered and it is safe to honour the request to move forward in logical time based on the states of all time-regulating federates, the RTI then notifies the requesting federate using the *Time Advance Grant* (TAG) service. Although this approach to time synchronization is conservative in nature, it is also possible to implement optimistic synchronization using HLA Time Management services. For example, Ferenci, Perumalla and Fujimoto (2000) perform optimistic synchronization in HLA using their Federated Simulations Development Kit (Georgia Tech, 2001) to develop an RTI that implements a subset of HLA Time Management services including *Retract*, *Request Retraction*, *Flush Queue Request* and *Time Advance Grant*.

Figure 2.10 provides an overview of services in the HLA Time Management category.

Service Category	Spec #	Service Name	Direction
Time Management	8.2	Enable Time Regulation	Federate -> RTI
Time Management	8.5	Enable Time Constrained	Federate -> RTI
Time Management	8.8	Time Advance Request	Federate -> RTI
Time Management	8.10	Next Message Request	Federate -> RTI
Time Management	8.12	Flush Queue Request	Federate -> RTI
Time Management	8.13	Time Advance Grant	RTI -> Federate
Time Management	8.17	Query Logical Time	Federate -> RTI
Time Management	8.19	Modify Lookahead	Federate -> RTI
Time Management	8.20	Query Lookahead	Federate -> RTI
Time Management	8.21	Retract	Federate -> RTI
Time Management	8.22	Request Retraction	RTI -> Federate

Figure 2.10: Overview of HLA Time Management services

6. HLA Data Distribution Management Services

Federates subscribe to receive updates concerning relevant object class attributes and interaction classes using HLA Declaration Management services. However, there are no mechanisms in the Declaration Management services that allow federates to subscribe to updates for specific object instance attributes or specific interaction instances. On the receiving end, this may result in federates receiving an excessive number of updates concerning object instances in which they have no interest. On the transmission end, this can result in federates sending instance updates which none of the other

federates needs. HLA Data Distribution Management (DDM) services provide additional functionality that helps to further refine the sending and receiving of messages during federation execution by filtering out irrelevant updates concerning specific object instance attributes and interaction instances.

HLA DDM filtering is accomplished by the use of publication and subscription *regions*. A region is composed of a set of defined *ranges* of values for specified class attributes. Each range is a continuous interval that specifies a lower bound and upper bound for a class attribute. A range must fall within the minimum and maximum bounds for the given class attribute, which is specified as a *dimension*. The *default region* covers the full range of all dimensions. Federate can publish updates to regions they specify or subscribe to receive updates from custom regions they define. When the publication region of one federate overlaps with the subscription region of another, the RTI routes the relevant updates from the sender to the recipient. Using these means, DDM can be used to filter out updates concerning instance attribute values which fall outside the region of interest of subscribing federates.

Figure 2.12 shows four example DDM publication and subscription regions based on two dimensions X and Y. From the diagram, Pub_Region_2 does not overlap with either subscription region. Consequently, the RTI will not forward any Pub_Region_2 updates to federates in either of the subscription regions. Similarly, as Sub_Region_1 does not overlap with either of the publication regions, a federate in Sub_Region_1 will not receive any updates from the two publication regions. However, Pub_Region_1 and Sub_Region_2 do overlap with one another. This means that all updates published to Pub_Region_1 will be forwarded by the RTI to Sub_Region_2. This includes updates in Pub_Region_1 with attribute values that fall outside the ranges in Sub_Region_2.

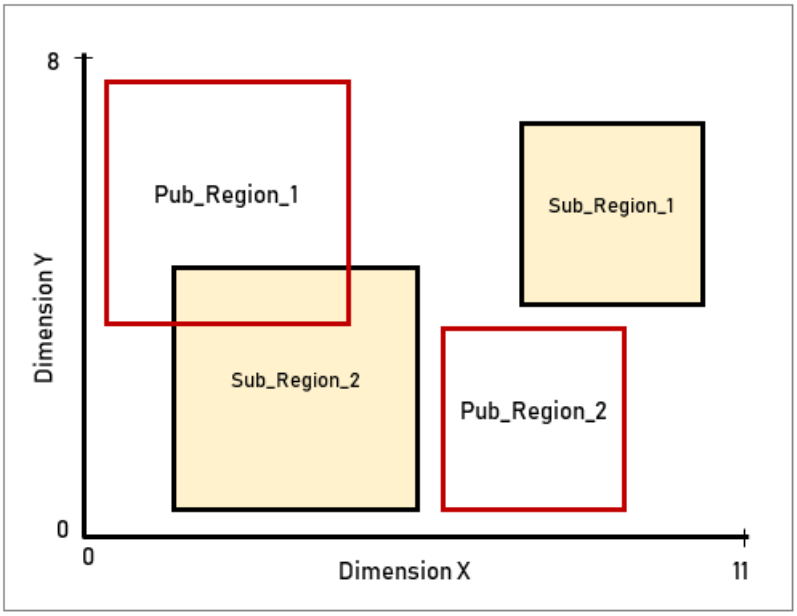

Figure 2.11: HLA DDM publication and subscription regions

Figure 2.12 provides an overview of main services in the HLA DDM category.

Service Category	Spec #	Service Name	Direction
Data Distribution Management	9.2	Create Region	Federate -> RTI
Data Distribution Management	9.3	Commit Region Modifications	Federate -> RTI
Data Distribution Management	9.5	Register Object Instance With Regions	Federate -> RTI
Data Distribution Management	9.6	Associate Regions For Updates	Federate -> RTI
Data Distribution Management	9.8	Subscribe Object Class Attributes With Regions	Federate -> RTI
Data Distribution Management	9.10	Subscribe Interaction Class With Regions	Federate -> RTI
Data Distribution Management	9.12	Send Interaction With Regions	Federate -> RTI

Figure 2.12: Overview of HLA Data Distribution Management services

7. **HLA Support Services**

The support service group includes calls that enable federates to query information from the RTI that is relevant to the state of federation execution. This includes services to retrieve handles for various components of the federation such as object classes and interaction classes. This group also includes services to retrieve various settings such as upper and lower bounds for DDM ranges.

Figure 2.13 provides examples of HLA Support Services.

Service Category	Spec #	Service Name	Direction
Support Services	10.4	Get Federate Handle	Federate -> RTI
Support Services	10.5	Get Federate Name	Federate -> RTI
Support Services	10.6	Get Object Class Handle	Federate -> RTI
Support Services	10.7	Get Object Class Name	Federate -> RTI
Support Services	10.9	Get Object Instance Handle	Federate -> RTI
Support Services	10.10	Get Object Instance Name	Federate -> RTI
Support Services	10.11	Get Attribute Handle	Federate -> RTI
Support Services	10.12	Get Attribute Name	Federate -> RTI
Support Services	10.15	Get Interaction Class Handle	Federate -> RTI
Support Services	10.16	Get Interaction Class Name	Federate -> RTI
Support Services	10.17	Get Parameter Handle	Federate -> RTI
Support Services	10.18	Get Parameter Name	Federate -> RTI
Support Services	10.23	Get Available Dimensions For Class Attribute	Federate -> RTI
Support Services	10.24	Get Available Dimensions For Interaction Class	Federate -> RTI
Support Services	10.25	Get Dimension Handle	Federate -> RTI
Support Services	10.26	Get Dimension Name	Federate -> RTI
Support Services	10.27	Get Dimension Upper Bound	Federate -> RTI
Support Services	10.29	Get Range Bounds	Federate -> RTI
Support Services	10.30	Set Range Bounds	Federate -> RTI
Support Services	10.41	Evoke Callback	Federate -> RTI

Figure 2.13: Overview of HLA Support services

2.6.1.3 The HLA Object Model Template

The HLA Object Model Template (OMT) (IEEE, 2010c) specifies a standard structure for federation data description documents such as FOMs and SOMs. The OMT defines the structure of FOM and SOM components such as object classes and their related attributes, interaction classes and their related parameters, data distribution dimensions, ranges and regions. It is on the basis of the structures defined by the OMT standard that FOM and SOM documents are used to specify details of federation components.

2.6.1.4 HLA Federation Execution

Figure 2.14 shows an example of an HLA federation execution with two federates that are both time-regulated and time-constrained. The illustration uses services from the categories discussed in previous sections. Federation Management services are used for high level federation execution functions including *Connect*, *Create Federation Execution*, *Join Federation Execution*, *Resign Federation Execution* and *Destroy Federation Execution*. Declaration Management Services are used to declare publication and subscription interests including *Publish Object Class Attributes*,

Subscribe Object Class Attributes, *Publish Interaction Class* and *Subscribe Interaction Class*.

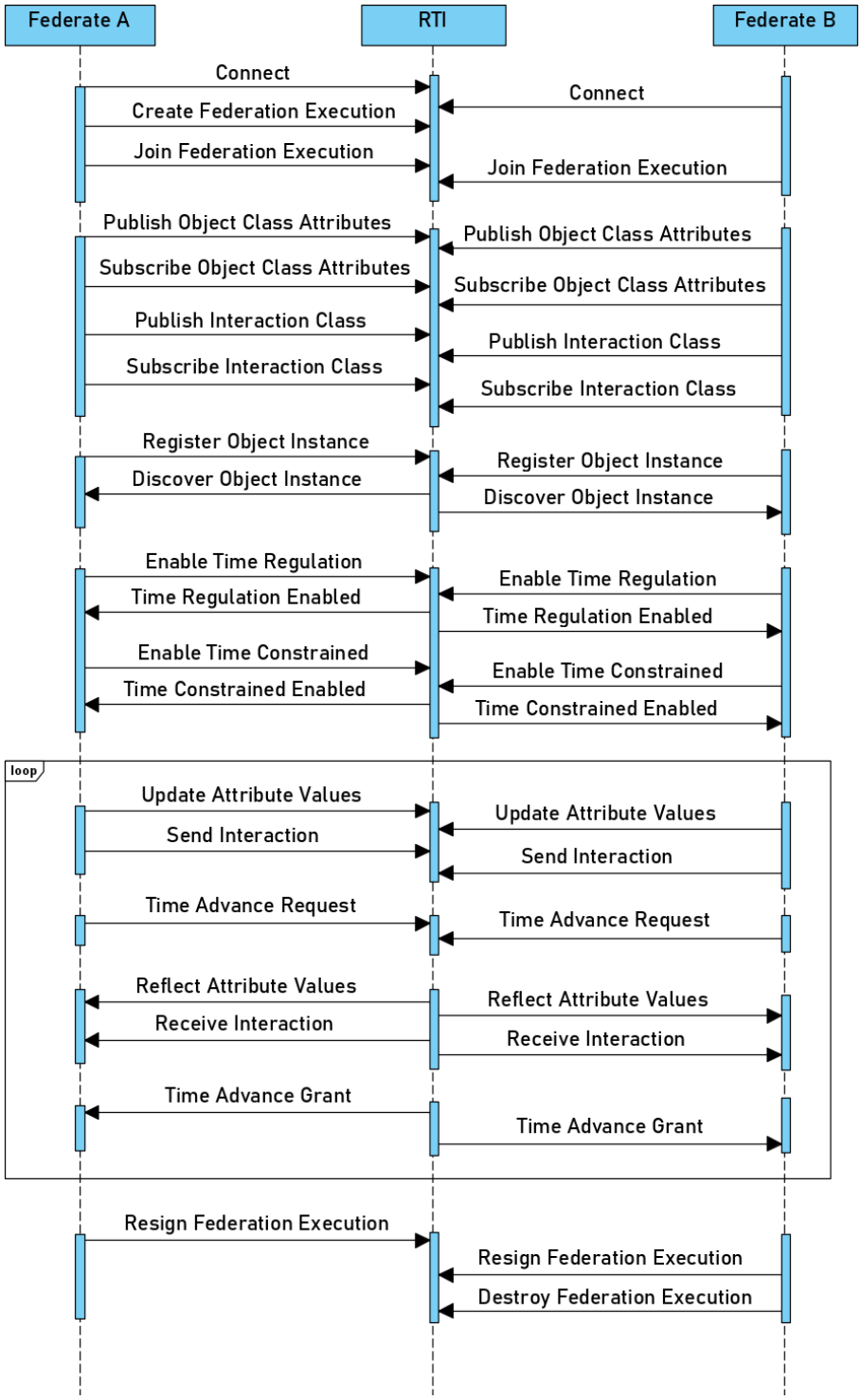

Figure 2.14: Example HLA federation execution

Object Management services are used to create instances and update values, including *Register Object Instance*, *Discover Object Instance*, *Update Attribute Values*, *Send Interaction*, *Reflect Attribute Values* and *Receive Interaction*.

Time Management services are used to set the time-regulation and time-constrained status of federates, as well as for coordinating time advance, including *Enable Time Regulation*, *Time Regulation Enabled*, *Enable Time Constrained*, *Time Constrained Enabled*, *Time Advance Request* and *Time Advance Grant*.

2.6.1.5 RTI Implementations

Various RTI implementations have been developed by commercial and open-source projects using the specifications in the HLA standard. RTI implementations may be fully compliant or partially compliant with the version of the HLA standard they support and may include the full set of services or offer a subset of the services defined in the Federate Interface Specification. Figure 2.15 provides a list of some popular commercial and open-source RTI implementations. The performance of a federation execution can vary depending on the RTI implementation used, as shown in experimental work by Fujimoto and Hoare (1998), Malinga and Le Roux (2009) and Gutlein *et al.* (2020).

RTI Name	Type
Pitch pRTI	Commercial
MAK RTI	Commercial
Portico RTI	Open Source
CERTI	Open Source
OpenRTI	Open Source

Figure 2.15: RTI Implementations

2.6.2 The Functional Mock-Up Interface

The Functional Mock-Up Interface (FMI) is a simulation interoperability standard developed by the Modelica Association. The FMI is popular in industry applications for supporting the integration of heterogeneous simulations. The FMI standard is developed with two main objectives in mind (Blochwitz *et al.*, 2011):

1. **FMI for Model Exchange**

   To produce representations of simulation models in a standard format such that the models can be used by different simulation tools apart from the one in which they were originally created.

2. **FMI for Co-Simulation**

   To enable interoperability between heterogeneous simulations by providing an interface specification that enables coordinating data exchange and time synchronization among the heterogeneous simulations.

In the FMI, individual simulations are referred to as Functional Mock-up Units (FMUs). The FMI for Co-Simulation standard includes a component called the *master algorithm* which performs the function of coordinating the execution of the FMUs. FMUs cannot communicate directly with one another, but only through the master. The role of the master algorithm is similar to that of the RTI in the HLA. As with the HLA, the FMI standard provides specifications for an interface between the master and the FMUs consisting of function calls. However, unlike the HLA RTI, the FMI does not prescribe how interactions between the master and FMUs should proceed for the purposes of time synchronization and data exchange. The implementation of a master algorithm that performs these functions appropriately is a task that is left up to the simulation developer.

2.6.3 Hybrid HLA and FMI

Although the HLA and FMI are developed separately, it can be useful to combine the strengths of these two leading simulation interoperability standards. While the FMI is more widely supported in industry applications, it lacks a generic master algorithm and is geared towards coordinating FMUs together on a single computing node. On the other hand, the HLA has more support in military than in industry applications

but has the advantage of including a generic RTI with a rich set of services for coordinating federates across multiple computing nodes. Some work has been done to fuse the two standards and take advantage of their individual strengths. For example, Awais *et al.* (2013) employ the HLA RTI as a generic master to coordinate the execution of FMUs. (Neema *et al.*, 2014) follow a similar approach and demonstrate a method for automatically wrapping FMUs as HLA federates.

2.7 Communication in Distributed Simulation

As discussed in section 1.1, the work in this thesis is primarily concerned with the efficient management of communication in large-scale distributed Urban Simulations. Communication in distributed simulations is necessary for data exchange and time synchronization. However, the additional overheads due to the need for communication can present a significant bottleneck to the performance of distributed simulations. Therefore, it is important to make efficient use of the communication bandwidth available on the interconnecting network between computing nodes.

2.7.1 Message Types

As discussed in section 2.4, time synchronization is necessary for producing correct results from distributed simulations. Both conservative and optimistic synchronization approaches employ communication to achieve their goals. Time synchronization messages carry information that is required for the synchronization algorithms to work properly. Apart from time synchronization information, distributed simulation messages are also used for the purpose of exchanging information related to simulation entities in order to maintain a consistent global view of data. The two concerns are usually related, as data received at the wrong time can lead to causal errors due to out-of-order execution. In respect of the two concerns of time synchronization and simulation data exchange, distributed simulation messages may be of three types:

1. **Time and Data**

   These are messages that contain both time synchronization information and data concerning simulation entities. These are usually in the form of timestamped messages. For example, PDES timestamped events contain information that is relevant for time synchronization as well as data concerning the event to be processed. Similarly, HLA time-regulating federates send timestamped messages that carry attribute updates relevant to simulation objects as well as logical time information that influences decisions by the RTI to send TAGs in response to TARs from other time-constrained federates joined to the federation execution.

2. **Time Only**

   These are messages that contain only information pertinent to time synchronization. For example, conservative algorithms can employ null messages which carry timestamp information used to provide additional lower bound time guarantees that help to prevent deadlocks from occurring during distributed execution. Optimistic algorithms use anti-messages which carry information about previous messages whose effect must be cancelled during cascading rollbacks when causal errors are detected. Another example of this type of message is the use of TARs and TAGs in the HLA for requesting and granting permission for federates to move forward in time.

3. **Data Only**

   These are messages that carry information about simulation entities but do not include any time-related information. For example, HLA federates that are not time-regulating can only send RO messages which have no timestamp information and do not affect time synchronization decisions made by the RTI. Similarly, federates that are not time-constrained can only receive RO messages. A situation where this may be appropriate is in the case of an observer federate that aggregates data from other federates during distributed execution.

2.7.2 Communication Message Volume

The volume of communication messages required during distributed execution may vary from one distributed simulation to another depending on factors such as the synchronization algorithm used, the number of entities involved in the simulation, the partitioning strategy and whether the computations required in the simulation are tightly coupled and require frequent communication or loosely coupled and communication is consequently infrequent.

As regards messages solely devoted to time synchronization, reducing the volume of communication depends on the synchronization algorithm. As discussed in section 2.4, strategies for addressing this issue include conservative algorithms reducing the volume of null messages by only transmitting them on request, and optimistic

algorithms employing lazy event cancellation to prevent sending unnecessary anti-messages.

With respect to messages that carry information relevant to both time synchronization and simulation data, one approach to reducing communication overheads in distributed simulations is by the use of *dead reckoning algorithms*. Dead reckoning algorithms have been used extensively in joint training simulations that are based upon the HLA, DIS and other related standards (Lin, Blair and Woodyard, 1997), (Miller and Thorpe, 1995). Using the dead reckoning approach, each federate needs to keep internal approximate models of external objects that it has an interest in but does not own. Federates can then make local approximations to estimate an external object's attributes based on its expected behaviour. This local approximation can serve as a substitute in place of a message containing an update of the external object's attributes. The updates only become necessary when the owning federate detects that an object's simulated behaviour deviates significantly from its approximation on other federates. An example where this method is useful is a spatially partitioned scenario in which each federate simulates a number of vehicles moving across a terrain and each federate needs to keep track of the positions of vehicles in other federates. The position of an external vehicle moving at a constant velocity can be estimated locally on tracking federates as long as it does not perform actions such as changing direction, accelerating, decelerating, or stopping. In cases like these, position updates for the vehicle can be provided in a message from the owning federate. Otherwise, it is not necessary to communicate position updates for the vehicle. This method is effective where communication can be substituted with additional local computation. However, the dead reckoning approach is not applicable in circumstances where accurate local approximations cannot be made to replace the simulation data delivered by communication messages.

2.7.3 Disk Caching Alternative

To supplement the discussion on reducing communication overheads, an alternative approach worth mentioning is the use of disk caching as a substitute for workload distribution across multiple computing nodes. The approach of disk caching has been applied to large-scale data processing (Kyrola, Blelloch and Guestrin, 2012) (Zhu, Han and Chen, 2015) (Ko and Han, 2018). However, there is a lack of application of

this method in the literature to large-scale simulation, which includes additional considerations such as time synchronization. The disk caching approach involves caching portions of large-scale datasets to disk in cases where the entire dataset cannot be processed together due to memory constraints (Dementiev, Kettner and Sanders, 2008) (Imgrund and Arth, 2017).

It could be argued that if the aim is to avoid communication over a network, this approach could be applied to a large-scale simulation whose configurations exceeds the memory requirements of a single computing node. This could be executed in a scheme where the simulation is partitioned into LPs that meet the memory constraints and are loaded into memory turn by turn in a round robin fashion, processed and cached back to disk. In such a scheme, the LPs could communicate with one another using shared memory. It could further be argued that the use of newer storage technologies such as SSDs could improve the I/O performance issues associated with reading and writing to traditional HDDs (Kyrola, Blelloch, and Guestrin 2012). Although such an approach would only require a single computing node, eliminate network communication overheads, and increase processor utilization, the scale to which this approach can be extended is limited as it depends on vertical scaling of disk storage rather than horizontal scaling. As simulation size grows very large, vertical scaling approaches practical limitations which can be overcome using the horizontal scaling approach (Appuswamy *et al.*, 2013). In this case, horizontal scaling also has the added advantage of adding more processing power to share the total computational workload, which is not possible with vertical scaling.

It could also be argued that a combination of the disk caching approach with horizontal scaling could be applied in an approach where each computing node processes multiple LPs to increase processor utilization and aid *latency hiding*, where useful computational tasks are performed to keep the processor busy while waiting to receive communication messages over the interconnecting network. Although this argument has merits and would be a potentially useful direction for exploration, the primary focus of this thesis is on managing communication efficiently for a large-scale simulation running on a distributed system. Therefore, full attention is given to communication between the computing nodes without detailed scrutiny of the local execution schemes that may be possible on individual nodes.

2.7.4 Communication Networks

As noted in previous sections, the volume of messages exchanged during distributed execution can affect performance by increasing communication time. Communication time also depends on the interconnecting network technology including aspects such as the physical medium and the network protocol used. Variance in performance between different network technologies is shown in experiments by Bell *et al.* (2003) and Fujimoto and Hoare (1998). Some high-level properties of the network that affect communication time are bandwidth, latency and error rate. Bandwidth is a measure of throughput capacity calculated as the number of bits transmitted per unit time. Latency is a measure of speed calculated as the delay between the time a bit leaves the sender and the time it arrives at the recipient. Depending on whether a reliable or unreliable communication protocol is used, data that is corrupted or lost in transit may need to be re-transmitted. A high error rate results in a large number of re-transmissions and poor network performance. Although performance differences may result from using different networking technologies, the work in this thesis focuses on making efficient use of the available network resource within its given constraints and does not attempt to optimize for any specific networking technology.

2.8 Distributed Performance Models

A variety of approaches have been proposed in the relevant literature that are useful for estimating the performance of distributed computations. This section provides an overview of the main existing approaches. Each approach considers several factors relevant to distributed performance. Some factors are shared between them, while others are unique to specific approaches. All approaches make assumptions and simplifications that may render them more suitable for some distributed applications than for others.

2.8.1 The Parallel Random-Access Machine

The Parallel Random-Access Machine (PRAM), introduced by Fortune and Wyllie (1978) is a model of parallel computation that consists of multiple cooperating processors communicating via shared memory. The PRAM has several built-in assumptions which help it to realize a simplified approach to estimating the performance of parallel computations. One of the assumptions it makes is that no

additional cost is incurred due to communication between processors during parallel execution. The PRAM also assumes that synchronization cost is negligible, and no additional effort is required to synchronize the execution of the processors. The application of the PRAM model is suitable for shared memory systems where processors execute in lockstep with to one another, and this fits in well with Single Instruction Multiple Data (SIMD) processing. However, due to these inherent assumptions, it is not practical for distributed simulations, where communication costs and time synchronization costs can be significant concerns. Despite these shortcomings, the PRAM has been influential in the development of subsequent models of parallel computation that address some of its limitations.

2.8.2 The Bulk Synchronous Parallel Model

The Bulk Synchronous Parallel (BSP) model introduced by Valiant (1990) is a model of parallel execution composed of three essential elements:

- Processors that perform the required computations.
- A router that delivers messages between the processors.
- A method for performing global barrier synchronization at regular intervals.

In the BSP model, parallel execution proceeds according *supersteps*, during which three stages are conducted in sequence:

- Computation
- Communication
- Global barrier synchronization

Figure 2.16 provides a conceptual illustration of the three stages in a single BSP superstep.

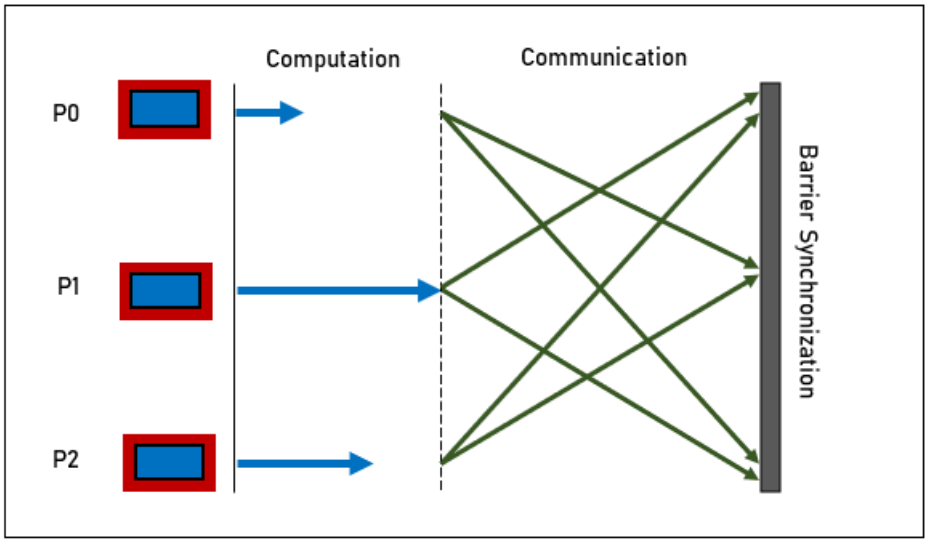

Figure 2.16: Conceptual illustration of a BSP superstep

Unlike in the PRAM model, the BSP model includes both communication costs and synchronization costs in its distributed performance estimates. Also, processors in the BSP model are not assumed to work in lockstep as they are in the PRAM. Because of these considerations, the BSP is applicable to Multiple Instruction Multiple Data (MIMD) processing and can offer more realistic performance estimates for distributed simulations. The BSP model is useful for estimating an upper bound on total execution time. It achieves this result by treating each of the three stages of a superstep as one bulk operation that covers all of the individual operations conducted by the processors. Therefore, the total cost of each super-step is an upper bound on the sum of computation cost, communication cost and barrier synchronization cost. Calculating costs in the BSP model relies on the following parameters:

- S: Number of supersteps.
- P: Number of processors.
- g: Minimum time required to send or receive a message between processors.
- l: Synchronization cost in a superstep.
- h_S: Maximum number of messages sent or received by any processor in a superstep s.
- w_S: Maximum time for work completed by any processor in a superstep s.

For each superstep s, an upper bound on communication cost is determined by:

$$\text{communication cost}_S = h_S \times g$$

For each superstep s, an upper bound on the total cost is determined as:

$$\text{total cost}_S = w_S + h_S \times g + l$$

For all supersteps, an upper bound on the total cost is determined as:

$$\text{total cost} = W + H \times g + S \times l$$

Where:

$$W = \sum_{s=1}^{S} w_s$$

and

$$H = \sum_{s=1}^{S} h_s$$

Some parameters, such as S and w_S depend on the simulation application. Other parameters such as h_S depend on the communication strategy during distributed execution. The value of the parameter l depends on the cost of barrier synchronization. The parameter g, also called the *bandwidth factor*, depends on the properties of the interconnecting network, and can be measured empirically. As g is measured empirically, the BSP model is non-specific with respect to the topology and protocols used in the interconnecting network.

2.8.3 The LogP Model

The LogP model introduced by Culler *et al.* (1993) is a model of parallel computing introduced after the PRAM and BSP models. Similar to the BSP model, the LogP model considers communication and synchronization costs in its performance estimation. The LogP model relies on the following parameters:

- L: Upper bound on the *latency* of message transmission.
- o: The *overhead* associated with communication operations.
- g: Lower bound on the time *gap* between send operations or between receive operations.
- P: Number of *processors*.

Figure 2.17 provides an illustration showing LogP model parameters in an interaction between two processors. Unlike the BSP model, the LogP model is not limited to the use of barriers as the sole method for synchronization.

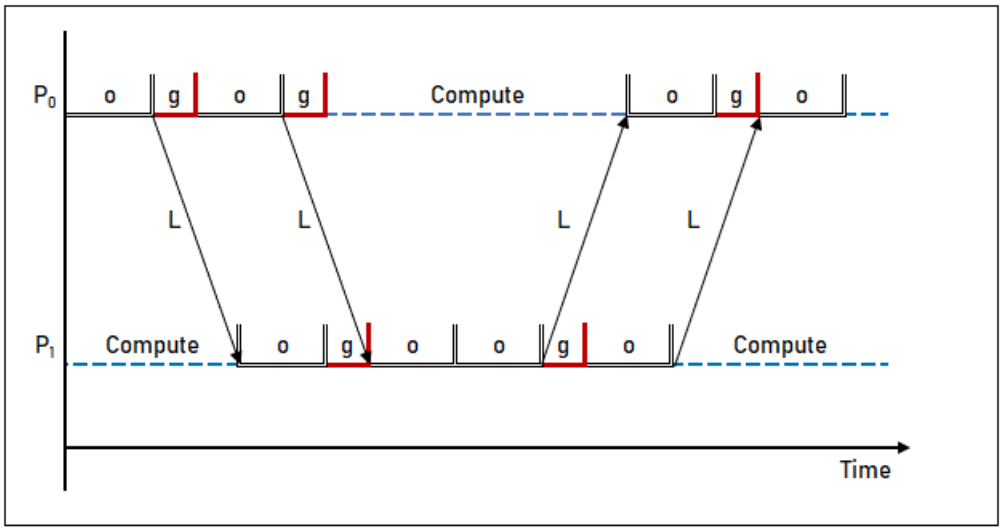

Figure 2.17: LogP model interactions

The LogGP model proposed by Alexandrov *et al.* (1995) extends the LogP model to account for performance differences due to long messages. This extension is applied with the recognition that communicating with long message can lead to better performance than multiple equivalent short messages. The LogGP model relies on the four parameters of the LogP model together with a new parameter, G, the time *gap per byte*.

Another notable extension is the LogGPS model introduced by Ino, Fujimoto and Hagihara (2001) which extends the LogGP model to account for additional overhead incurred due to synchronous or blocking communication.

2.8.4 Other Approaches

From the systematic review conducted by Flores-Contreras et al. (2020) approaches for estimating the performance of parallel applications include analytical methods, statistical methods and trace-based methods.

Analytical methods are equation-based and can produce accurate performance evaluations. However, a new set of equations needs to be crafted for each application, relying on application-specific domain knowledge to produce good

accuracy. With equation-based approaches, it can be difficult to account for non-deterministic behaviour in parallel applications.

Statistical approaches attempt to use regression-based methods or machine learning methods to fit a statistical model to the performance characteristics of the parallel application. Although such approaches can produce good estimates, they are data-intensive and require substantial amounts of performance data to fit an accurate statistical model. Multiple executions of the parallel application need to be conducted to provide enough data for this purpose. As with the analytical approach, it can be difficult to account for non-deterministic behaviour in parallel applications when using statistical approaches.

Trace-based methods attempt to replicate the behaviour of the parallel application, relying on trace data recorded from previous executions. As with statistical methods, this approach is data intensive and requires multiple executions of the parallel application to replicate application behaviour accurately. Trace-based methods may also require setting up a system that scales similarly to the one being replicated.

3 METHODOLOGY

3.1 Overview

Chapter 1 set out the aim for this research work as the development of a framework for evaluating the performance of distributed simulations. There, a list of objectives was also provided which will be carried out in order to accomplish the stated aim. This chapter provides an overview of the systematic approach adopted to address the stated aim and objectives. The framework development is carried out by first implementing selected distributed simulations in order to identify the major components involved. Experiments will be conducted using the implemented simulations to identify key factors that exert significant influence on communication time during distributed execution. The framework will consider alternative communication strategies which are aimed at reducing the communication overheads in distributed simulations. Following this approach, appropriate case studies will be used from the Urban Simulation field which meet the requirements that were set out in the aim and objectives.

One of the listed objectives is to develop distributed simulations for conducting experiments. The approach adopted for developing these distributed simulations is based on the High Level Architecture (HLA) standard which was introduced in section 2.6. However, as noted in Chapter 2, the HLA is not the only possible approach. In Chapter 2, other potential methods for implementing distributed simulations were discussed. Alternative approaches include the use of distributed simulation toolkits (e.g. Repast HPC, D-MASON and FLAME), custom development using communication libraries (e.g. MPI), other distributed simulation standards (e.g. FMI). The HLA was selected from among the alternative approaches due to advantages such as maturity of the standard, support for scalability when executing large homogeneous simulations, support for interoperability between heterogeneous simulations, extensibility, flexibility, inclusion of standard procedure for time synchronization, availability of supporting documents such as manuals and standard specifications, and availability of software. Although there are merits to

basing the distributed simulation implementation on the HLA standard, one limitation of using this approach is that additional development work needs to be done for existing simulations that are not already HLA-compliant to bring them in line with the standard, modifying them to observe the HLA Rules and function as well-behaved federates during federation execution. In many cases, this requires having direct access to the application code or developing a middleware capable of translating between the RTI and the simulation application. However, this disadvantage is not peculiar to the HLA alone and is shared by all the other approaches discussed.

3.2 Distributed Simulation Approach

The following sections expand on the points raised in the previous section, highlighting the main advantages of employing the HLA and comparing it to other potential approaches for implementing distributed simulations. In the following sections, the factors chosen to compare the alternative distributed simulation approaches reflect practical concerns for enabling the experiments proposed in the aim and objectives.

3.2.1 Distributed Simulation Standard

It is useful to build the work on a mature distributed simulation standard as this opens it up to support other existing simulation applications that are compliant with the standard. The HLA and FMI share this advantage as both are distributed simulation standards with detailed specifications. However, the other approaches discussed do not have this advantage. Figure 3.1 compares the HLA with other approaches in relation the standardization of their approach to distributed simulation. The MPI library is a mature standard for message passing in distributed systems. However, it is targeted towards distributed applications in general, and does not directly address concerns specific to distributed simulations such as time management and interoperability.

Criterion	HLA	DS Toolkits	Custom MPI	FMI
Mature DS Standard	✓	x	x	✓

Figure 3.1: Standards – HLA and other approaches

3.2.2 Homogeneous Simulation Scalability Support

All the approaches discussed offer support for scaling up computing resources to coordinate the execution of a large-scale homogeneous simulation that has been partitioned. The degree of support offered various from one approach to another. Scalability support is the main goal of some approaches. For example, simulation toolkits such as Repast HPC and FLAME are primarily designed to support homogeneous simulation scalability. Similarly, most custom distributed simulations in the literature that employ MPI on HPC have this goal in mind. Other approaches such as HLA and FMI also offer support for this goal. Although it not their main concern, it is not difficult to adopt them for this purpose. Figure 3.2 compares the HLA with other approaches in relation to their support for homogeneous scalability.

Criterion	HLA	DS Toolkits	Custom MPI	FMI
Homogenous Scalability Support	✓	✓	✓	✓

Figure 3.2: Scalability – HLA and other approaches

3.2.3 Heterogeneous Simulation Interoperability Support

The HLA and FMI distributed simulation standards are purposely designed to allow interoperability between heterogeneous simulations and consequently define specifications that facilitate the coordination of such simulations. The other approaches discussed do not have this facility and may require significant development work to enable this functionality. The support or lack of support for interoperability affects the extensibility of the approach. This is evident in cases where existing simulation tools need to be connected together conduct simulation experiments. Figure 3.3 compares the HLA with other approaches in relation to their support for heterogeneous interoperability.

Criterion	HLA	DS Toolkits	Custom MPI	FMI
Heterogeneous Interoperability Support	✓	x	x	✓

Figure 3.3: Interoperability – HLA and other approaches

3.2.4 Standard Time Synchronization

As discussed in section 2.4, time synchronization is an essential element in distributed simulation which sets it apart from other distributed computing applications. The HLA Time Management services, introduced in section 2.6 prescribes basic functionality that can readily be employed to perform conservative synchronization, as well as additional services that can be useful for implementing optimistic synchronization. Distributed simulation toolkits such as Repast HPC and D-MASON also include mechanisms for synchronization. Although the FMI provides functions that can be used for time synchronization purposes, it does not offer any specification for how synchronization is to be done. Time synchronization is a function performed by the master algorithm. However, as discussed in section 2.6.2, there is no generic FMI master algorithm, and the implementation details are left to the developer (Neema *et al.*, 2014). Custom approaches using the MPI library also do not have prescribed methods for time synchronization, although this can be developed using methods from the library to drive simulation time forward while avoiding deadlocks and preserving causal order. Figure 3.4 compares the HLA with other approaches in relation to their inclusion of a standard procedure for time synchronization.

Criterion	HLA	DS Toolkits	Custom MPI	FMI
Standard Time Synchronization	✓	✓	x	x

Figure 3.4: Time Synchronization – HLA and other approaches

3.2.5 Supporting Documentation

Supporting documentation in the form of specifications, manuals, forums and email lists are helpful aids for development. Without such documentation, development work can prove difficult. It is also helpful to have a development community that is active and available to offer peer-to-peer support. Detailed specifications exist for the HLA and FMI which are useful for development, as well as related websites and GitHub projects. The availability of supporting documentation for distributed simulation toolkits varies from one toolkit to another. While Repast HPC offers detailed documentation and maintains an email support group, not all toolkits have these advantages. Widely used

communication libraries such as MPI have detailed documentation that can serve as a useful resource for development. Figure 3.5 compares the HLA with other approaches in relation to the availability of their supporting documentation.

Criterion	HLA	DS Toolkits	Custom MPI	FMI
Supporting Documentation	✓	-	✓	✓

Figure 3.5: Documentation – HLA and other approaches

- Availability of Key Software:

  The availability of relevant software is also an important consideration in selecting the approach. The HLA provides specifications regarding how the RTI functions to coordinate a distributed simulation execution. As discussed in section 2.6, several alternative RTI implementations exist, some of which have been developed as open-source projects and others for commercial interests. Unlike with the HLA, the master algorithm for FMI which coordinates the FMUs is developed on an ad hoc basis. Several open-source distributed simulation toolkits exist including Repast HPC, FLAME and D-MASON. Similarly, multiple versions of the MPI library have been developed by open-source projects, based on the specifications set out by the MPI standard. Figure 3.6 compares the HLA with other approaches in relation to the availability of supporting software.

Criterion	HLA	DS Toolkits	Custom MPI	FMI
Software Availability	✓	✓	✓	x

Figure 3.6: Software – HLA and other approaches

The features discussed in the preceding sections are important considerations for selection of the approach. Although other methods are also applicable to this work, the HLA was selected as the most workable approach given the considerations discussed.

3.2.6 Choice of Middleware

The previous sections have provided justification for employing the HLA as an approach to implementing distributed simulations. As discussed, the RTI is a key component of the HLA of which several implementations have been developed based on HLA specifications. Some RTI implementations are developed by open-source projects and others are commercial. An RTI implementation may support more than one version of the HLA standard, or it may support a particular version of the standard, such as HLA 1.3, HLA 1516-2000 or HLA 1516-2010. While some implementations offer the full range of HLA services for the standards they support, other RTI implementations offer a limited set of functions. Some examples of RTI implementations are Portico RTI, CERTI, OpenRTI, Open HLA, MAK-RTI, and Pitch pRTI. From the available options, the Portico RTI has been selected for use in experiments due to its open-source status, its active development community on GitHub, and its support for an extensive range of HLA services. Although the open-source Portico RTI implementation does not cover the entire spectrum of HLA services, the set of services it does implement are adequate for the development of the distributed simulation experiments.

3.3 Communication Management Strategies

The primary of the planned framework is to provide a means for estimating the execution performance for distributed simulations. The framework will consider factors related to communication during simulation execution. As there are various strategies that can be applied to improve communication performance in distributed simulations, the framework will consider alternative methods of communication management. The performance differences from the application of these methods should be reflected in the results produced from an implementation of the framework. The effects of various strategies on communication time during distributed execution will be evaluated by experimentation. The strategies considered are discussed in the following sections. These methods are neither exhaustive nor mutually exclusive and can be used in combination.

3.3.1 Approximation Strategy

This strategy is a generalized form of the dead reckoning method discussed in section 2.7. As with the dead reckoning method, this strategy attempts to perform local

computations to approximate the properties of external objects that are not present locally, but which are essential to the local simulation due to their interactions with local objects. This eliminates the need to exchange update messages regarding the properties of such external objects. As the data dependencies for performing the required calculations are incomplete, the resulting properties are only an approximation. Depending on the simulation application, this approach could result in a significant loss of accuracy. The resulting degree of accuracy depends on how well external objects can be approximated locally. In distributed simulations where the behaviour of external objects is unpredictable, accuracy is likely to be more heavily impacted than situations where external behaviour can be accurately predicted. Heterogeneous distributed simulations in particular are likely to include more variations in object types, each with a different behaviour. The behaviour of some objects may be easier to predict than others. In such cases, this strategy can be limited to those external objects whose behaviour can be predicted accurately. Update messages will still need to be exchanged for external objects whose behaviour cannot easily be predicted.

3.3.2 Message Elimination Strategy

This strategy is similar to the approximation method because it prevents exchanging unnecessary messages. However, it does not try make predictions of the behaviour of external objects. Instead, messages are eliminated if it can be estimated that the impact of their content on local computations will be insignificant. Identifying and preventing such non-essential communication results in reducing the overall volume of messages exchanged during distributed execution. Practically, identifying non-essential messages depends on the characteristics of the particular simulation application under investigation. This strategy can result in a loss of accuracy if the method for judging the impact of message contents is unreliable. Some high impact messages may be blocked while allowing low impact messages to transfer. However, there will be little impact on accuracy if a reliable method can be found to distinguish between high impact messages and low impact ones.

If a reliable method exists for separating high and low impact messages, the potential performance gains from using this method depend on the characteristics of messages exchanged during distributed simulation execution. In cases where many messages

are low impact and few are high impact, communication overheads can be reduced significantly. On the other hand, there will be little performance gain in cases where most messages are essential for computations. The diagram in Figure 3.7 illustrates a scenario showing a simple application of this method. Messages are exchanged between federates at every time step. However, at certain timesteps all messages are determined to be low impact. These are indicated by the crossed-out arrows. No messages are exchanged at those time steps, and this essentially makes the federates more loosely coupled. This simple scenario is practical in some cases. For example, in physical Building Energy Simulations, daylight computations will not be conducted at night when there is no solar radiation. Therefore, all attribute update messages solely intended for daylight computations can be eliminated for timesteps that occur at night.

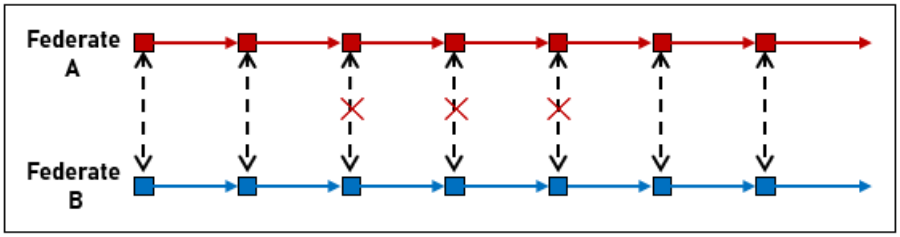

Figure 3.7: Message elimination strategy

3.3.3 Batching and Compression Strategy

As discussed in section 2.7, the performance of the interconnecting network is constrained by factors such as latency and bandwidth. The batching and compression strategy attempts to make efficient use of available network resources by combining multiple individual messages into a single message batch. The message batch is compressed before transmission to reduce the total size of data that needs to be transmitted over the network. This strategy can be effective in cases where high compression ratios can be achieved. It has the advantage of enabling the communication data to be transmitted in fewer packets, avoiding the extra overhead incurred in appending metadata (such as headers) to multiple smaller packets. This strategy has the disadvantage that additional time is required at the sender to pool messages together for batching. If all messages required to be included in a batch are not readily available at transmission time, this can add some delay to the process. This strategy also has the disadvantage that additional time is required at the sender

for compressing the message batch into a compact form. Compression time and final compressed size will vary depending on the contents of messages and the compression algorithm employed. It would be ideal to obtain a high compression ration with low compression time. In practice, there is a trade-off between compression ratio and compression time. At the receiving end, additional time is also required for de-compressing and un-batching messages. The performance of this strategy also depends on the maximum size of data that can be transmitted in a single packet on the interconnecting network.

A best-case scenario for the batching and compression strategy is the scenario where all messages can be combined into a single packet for transmission. This would make efficient use of the available network resource. A worst-case scenario is the case where the number of packets after batching and compression is equal to the number of original messages. In this scenario additional processing overhead is added without improving the efficiency of network resource usage. This scenario can occur in a case where each individual message is large, but message contents cannot be compressed further, yielding a low compression ratio.

Unlike the approximation strategy and the message elimination strategy, the batching and compression strategy preserves all messages, assuming the compression algorithm is lossless. Therefore, this approach preserves the fidelity of the distributed simulation and does not result in a loss of accuracy.

The diagram in Figure 3.8 provides an illustration for the batching and compression strategy, showing the process of combining individual messages together into batches, compressing batches to reduce total size and transmitting via packets.

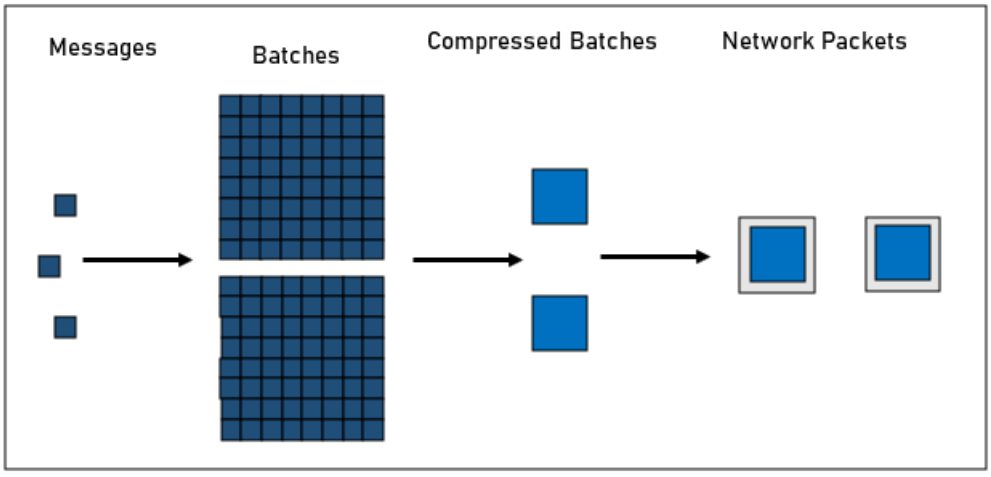

Figure 3.8: Batching and compression strategy

3.3.4 Hybrid Strategies

The three strategies discussed in previous sections are not mutually exclusive. Neither are they intended to represent an exhaustive list of communication strategies in distributed simulations. Two or more communication strategies can potentially be combined to produce a hybrid communication strategy. Such hybrid strategies will achieve the best results when used in contexts where each strategy can make a valuable contribution. Any hybrid strategy which includes one of the *lossy* strategies such as approximation or message elimination can result in a loss of accuracy in the simulation outputs. Where more than one lossy strategy is involved, the total loss of accuracy can potentially be higher than the loss due to any single strategy.

3.4 Case Study Selection

To shape the development of the framework, case studies from the domain of Urban Simulation will be used to conduct experiments. As discussed in Chapter 1, the Urban Simulation domain provides a variety of simulations that meet the requirements set out in the objectives: large-scale simulations that can be interoperate meaningfully in a simulation study. As set out in the objectives, two case studies are selected: a homogeneous distributed simulation and a heterogeneous distributed simulation. The following sections provide a brief overview of the selected case studies. A more detailed treatment is provided in Chapter 5.

Case Study One considers an Urban Simulation which is capable of growing exceptionally large at city scale. At this scale, the computing resources required to execute the simulation exceeds the capacity of a single stand-alone computing node. Therefore, it is necessary to distribute the simulation workload among multiple computing nodes in order to enable the possibility of execution. The simulation tool selected for Case Study One is the physical Building Energy Simulation tool CitySim. Multi-building simulation can be managed by CitySim using a single computing node when only considering a few hundred buildings. However, when considering thousands of buildings at city scale, memory requirements exceed the capacity of a single computer. Therefore, distributed simulation is necessary in order to use CitySim at city scale.

Case Study Two extends Case study one by adding a separate Urban Simulation that can interoperate meaningfully with CitySim in a heterogeneous distributed simulation. This case study enriches the framework by examining the effect of different communication patterns on performance in the distributed simulation. In the heterogeneous arrangement, communication patterns between similar simulators differs from those between dissimilar simulators. For instance, a distributed simulation with type A simulators and type B simulators can have four potential communication arrangements: A-to-A communication, B-to-B communication, A-to-B communication, and B-to-A communication. Each communication type can happen at different frequencies and have different message sizes. Because of these additional considerations, Case Study Two can be viewed as a more general form of Case Study One. Alternatively, Case Study One can be viewed as a special form of Case Study Two. The additional simulation selected for Case Study Two is the Nottingham Multi-Agent Simulation (No-MASS) tool. No-MASS is a Building Occupancy Simulation, simulating the influence of building occupants on energy usage. Interoperation between CitySim and No-MASS is potentially meaningful and can provide better energy predictions.

3.5 MAS Paradigm

As discussed in Chapter 1, the performance framework will adopt a meta-simulation approach. This will be based on the Multi-Agent Simulation (MAS) Paradigm discussed in Chapter 2. The following sections offer justification for the choice of the MAS paradigm for use in the framework:

- Paradigm Suitability:
  The different parts of the federation, including the federates and RTI, readily lend themselves to modelling as agents co-operating to complete a given task.

- Concurrent Execution:
  MAS agents execute concurrently, similar to how federates operate in a distributed simulation.

- Message Exchange:
  Agents communicate with one another by exchanging messages during simulation execution similar to the federates and RTI passing messages to one another during federation execution.

- Stochastic Experiments:
  The non-deterministic execution paths due to concurrency in distributed simulations can be captured by executing multiple stochastic runs of the MAS and determining confidence intervals.

- Heterogeneous Agents:
  MAS includes the flexibility to include different types of agents with varying behaviours. This can be used to capture the execution behaviour of heterogeneous distributed simulators.

- What-If Scenarios:
  MAS can be used to explore various what-if scenarios. This can be used in designing a distributed simulation by adjusting relevant parameters to estimate the impact that they will have on performance.

- Parameter Optimization:
  MAS enables performance optimization by exploring the parameter search space to find optimal parameters for executing the distributed simulation.

- Complex Dynamics:
  MAS can help capture and understand complex dynamics between the different parts of the distributed simulation without having to set up the computing resources required to execute the actual distributed simulation.

For these reasons, the MAS paradigm lends itself naturally to the proposed meta-simulation approach. As the MAS paradigm uses a bottom-up modelling approach, the behaviour of agents involved in the simulation is specified directly for each agent. However, system behaviour is not explicitly specified but arises from

interactions between agents. This feature can be useful for testing the effect of different agent co-operation strategies on the performance of distributed simulations. Agent behaviour can be modified to follow the desired co-operation strategy, which may involve agents taking simple actions or making complex decisions to adapt to changing conditions. This can produce emergent system level patterns that cannot easily be replicated using other modelling paradigms.

The framework to be developed will make use of the MAS paradigm to specify the key components of a distributed simulation, the behaviour of each component, the interactions between them, and the essential aspects of their shared environment that significantly influence communication performance. As the framework will be specifically designed for distributed simulations, it will account for considerations that are important for distributed simulations but are not required for distributed systems in general, such as the synchronization of timesteps among distributed simulators to avoid causal errors. A meta-simulation based on the framework can be implemented using any simulation toolkit that supports the MAS paradigm such as Repast Simphony or AnyLogic. Meta-simulations derived from the framework may also conceivably be implemented using simulation toolkits designed for general distributed systems, including packet-level simulators such as ns-3 (Henderson and Riley, 2006) or grid resource scheduling simulators such as GridSim (Buyya and Murshed, 2002).

3.6 Summary of Methodology

This chapter has provided details of the approach that will be applied to carry out the work listed by the objectives in Chapter 1. As discussed, Urban Simulation case studies are selected from the Building Energy Simulation and Building Occupancy Simulation application areas. Based on the selected case studies, distributed simulations will be developed using the HLA approach. Experiments will be conducted using the HLA distributed simulations to measure execution performance. Case Study One, the homogeneous distributed simulation, will includes multiple instances of a physical Building Energy Simulation. Case Study Two, the heterogeneous distributed simulation, will includes Building Energy Simulation instances as well as Building Occupancy Simulation instances. The communication strategies discussed in this chapter will be employed in the experiments. The

performance estimation framework, which is the main focus of this thesis, will be informed by lessons learned from the development process and by the outcomes of conducted experiments. Following the development of the framework, a concrete meta-simulation will be implemented based on the case study experiments in order to evaluate the framework and demonstrate its application.

4 FRAMEWORK BACKGROUND

4.1 Building Energy Simulation

As discussed in section 1.1, simulation studies in urban areas are needed to support planning and sustainably manage rapidly growing urban populations worldwide. One of the key planning areas for the urban environment is the management of energy resources. A substantial portion of energy use in the urban built environment is consumed by transportation and buildings. According to Eurostat (2019), energy consumed in the EU by households alone accounts for more than 27% of total energy demand in the EU, with transportation accounting for about 30%. Building Energy Modelling aims to forecast the total energy requirements for indoor heating, cooling, lighting, and powering appliances. According to surveys conducted by Foucquier et al. (2013), Reinhart and Davila (2016) and Johari *et al.* (2020), Urban Building Energy Modelling methods can be categorized into two main approaches: Statistical methods and Physical methods.

Statistical methods attempt to predict future building energy requirements based on historical usage. This approach is data-driven and requires substantial amounts of historical data to make accurate predictions. The statistical approach includes various regression methods and machine learning techniques.

Physical methods are simulations based on physics models of heat transfer. Physical Building Energy Simulations make use of thermal equations to forecast building energy requirements. Inputs to physical Building Energy Simulations include the location of buildings, properties of building materials, and weather data. Some physical Building Energy Simulation tools are designed to make energy forecasts for individual buildings e.g. EnergyPlus (Crawley *et al.*, 2001). Single building tools do not model interactions between different buildings in detail. As the focus is on a single building, detailed information is required about the simulated building in order to generate accurate predictions for its energy requirement. Other physical Building Energy Simulation tools are designed to make energy forecasts for a group of

buildings in close proximity to one another, such as the CitySim (Robinson 2012) tool. Multi building simulation tools model interactions between separate buildings in detail as these interactions are important for making accurate predictions for the group of buildings.

4.1.1 Physical Building Energy Simulation

This section provides an overview of the physical Building Energy Simulation process used by the CitySim tool. The inputs to CitySim include a *scene* file and a weather file. The scene file contains 3D representations of a group of buildings positioned in physical space, and the weather file contains data about weather patterns in the area. CitySim makes use of the information provided by these inputs to calculate the energy exchanges between buildings and the environment over a period of time. The results from these energy exchange calculations are used to estimate the building energy requirements over the simulated time period.

Each 3D building in a scene file is a collection of geometric *surfaces*. Types of surfaces in scene files include *wall surfaces*, *roof surfaces* and *floor surfaces*. These surfaces combine to create one or more internal thermal zones within each building which can exchange heat with one another. Building scenes may range widely in complexity. A simple building scene may contain a few shoebox-like buildings, each with four walls and a flat roof which join together to create a single internal thermal zone. A more complex building scene may contain several thousand buildings with complex surface structures and multiple internal thermal zones. Figure 4.1 shows an example of simple and complex building surface structures.

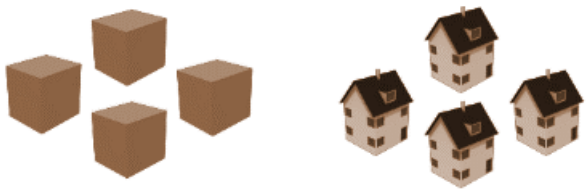

Figure 4.1: Simple and complex building surface structures.

CitySim accepts building scene files in a specific XML format. Zakhary *et al.*, (2016) extended CitySim to load building scenes created with the City Geographic Mark-up Language (CityGML) standard instead of a previous custom XML format.

CityGML (CityGML, 2020) is an XML-based standard for representing urban structures in 3D physical space.

As explained by Robinson (2012), CitySim calculates the radiation exchanges in a building scene during simulation execution. This includes interactions between one building surface and another, as well as interactions between building surfaces and the environment. These radiation exchanges contribute to indoor and outdoor temperatures and have an impact on heat transfer between building zones. The purpose of the radiation exchange calculations is to obtain a basis for estimating the energy demands required to suit building occupant comfort. Factors that contribute to occupant comfort include adjust indoor temperature via Heating, Ventilation and Air Conditioning systems (HVACS) as well as adjusting window blinds to let the sunshine in. Radiation exchange calculations performed by CitySim include different sets of equations for *shortwave* and *longwave* radiation exchange. Shortwave radiation exchange refers to the portion of solar radiation with wavelengths in the range of 0.3µm to 3µm that is absorbed by building surfaces. Longwave refers to infrared radiation exchanged between buildings in the wavelengths ranging from 3µm to 100µm.

Calculating these radiation exchanges in the urban setting is complicated by the fact that some building surfaces obstruct other building surfaces from directly exchanging radiation with one another. Building surfaces may also obstruct one another from directly exchanging radiation with the sun and sky. However, this can also serve to make the calculations more computationally tractable because not all pairs of surfaces in a large scene will need to exchange radiation with each other. In CitySim, the relationships between building surfaces that are not obstructed from one another and are therefore capable of exchanging radiation is represented using a sparse matrix for storage efficiency. Figure 4.2 provides a conceptual illustration of various shortwave and longwave radiation exchanges between buildings and the environment in an urban setting.

The details of the mathematical equations which form the basis for physical calculations of the radiation exchanges that occur between buildings and the environment is expanded upon in great detail by Robinson (2012). A brief summary is provided in the following sections.

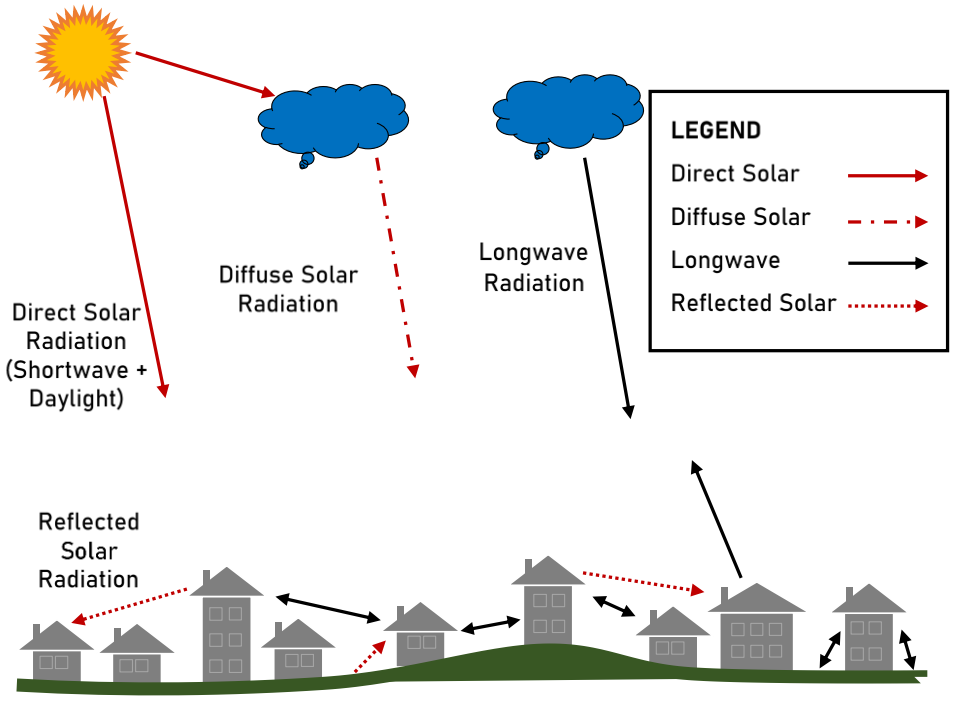

Figure 4.2: Illustration of some building radiation exchanges in an urban environment

- Shortwave Calculations:

  For each building surface, CitySim calculates the shortwave radiation that it receives from various sources. These sources include direct shortwave radiation received from the sun, diffuse solar radiation through the sky, and reflected solar radiation from other building surfaces. CitySim makes use of the Simplified Radiosity Algorithm (SRA) of Robinson and Stone (2006) to calculate the radiation exchanges in building scenes.

- Longwave Calculations:

  For each building surface, CitySim calculates the longwave radiation that is received from various sources. Sources of longwave radiation include the sky and the other surrounding building surfaces.

- Daylight Calculations:

  In addition to longwave and shortwave radiation exchange calculations, CitySim also performs calculations to determine the level of illuminance

within internal building zones received from sunlight. These are referred to as *daylight* calculations and provide useful measures which serve as a basis for determining the indoor lighting or shading requirements for occupant comfort.

- Thermal Zone Calculations:
  Finally, CitySim performs calculations for all internal zones, which includes determining the internal temperature that results from the radiation exchange calculations performed in previous steps. These are referred to as *thermal* calculations in CitySim.

Figure 4.3 provides an overview showing how the calculations described in the previous sections are performed by CitySim in a loop, with each loop iteration representing a single timestep representing one hour.

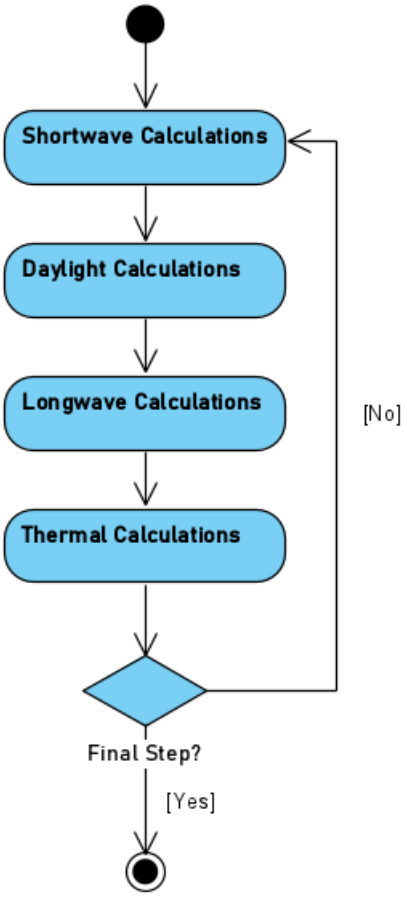

Figure 4.3: CitySim time-step loop

4.1.2 Building Occupancy Simulation

The energy demand in a building is regulated by its occupants. Building occupants use energy consuming HVACS to regulate indoor temperatures to suit their comfort. Building occupants also interact with building features such as windows and light switches to regulate indoor lighting. The bodies of occupants in buildings also exchange heat with their surroundings and consequently has an effect on indoor temperatures. Building energy consumption depends on the number of occupants and their activities within the building. Therefore, in order to produce more accurate energy predictions, it is useful to simulate the presence of occupants in buildings and their interaction with various building features.

However, creating a single monolithic simulation that performs Building Occupancy Simulation in addition to physical Building Energy Simulation is not ideal, and it is useful to both concerns. While physical Building Energy Simulation uses an approach based on physics equations, Building Occupancy Simulation lends itself to paradigms that model the behaviour of autonomous individuals interacting with their environment. For example, the MAS paradigm is used for Building Occupancy Simulation in the Nottingham Multi-Agent Stochastic Simulation (No-MASS) tool (Chapman, Siebers and Robinson, 2018).

4.1.3 Distributed Building Energy Simulation

Research work has previously been conducted in distributed Building Energy Simulation both for the purpose of interoperability with other simulations and for the goal of scaling up computing resources to execute large-scale Building Energy Simulations.

With the goal of performing distributed Building Energy Simulation for interoperability, Jain et al. (2016) use the HLA to integrate the urban land use model UrbanSim (Waddell, 2002) with the urban transportation model MATSim (Horni, Nagel and Axhausen, 2016). In another example, Menassa et al. (2014) employ the HLA to integrate the single Building Energy Simulation tool DOE2 (Rousset *et al.*, 2016) with a custom building occupancy ABS developed using the AnyLogic simulation toolkit. Wang, Siebers, and Robinson (2017) use the FMI to integrate the single Building Energy Simulation tool EnergyPlus with the Building Occupancy Simulation tool No-MASS.

Regarding the goal of performing distributed Building Energy Simulation to address the scalability issue, Hong *et al.* (2016) develop the City Building Energy Saver (CityBES) as a web-based platform to enable urban-scale Building Energy Simulation and provide analysis on potential energy savings from retrofitting measures. CityBES uses the CityGML standard for representing 3D building scenes. CityBES also makes use of the EnergyPlus tool as an engine for running physical Building Energy Simulations.

SimStadt is an urban energy simulation platform introduced by Nouvel *et al.*, (2015). Similar to CityBES, SimStadt also uses a CityGML representation for 3D building scenes. SimStadt allows a choice between different radiation exchange models including the Simplified Radiosity Algorithm and Perez Sky model that is also used by CitySim. SimStadt is also designed to be extensible by plug-in modules to allow different types of Urban Simulation studies to be conducted.

As discussed in Chapter 3, the content of the framework to be developed in this work will be informed by experiments conducted on HLA-based distributed simulations. The framework will enable the creation of meta-simulations that can be used to estimate the execution performance of specific distributed simulation applications. Although the framework will focus on HLA-based distributed simulations, it can potentially be extended to make performance estimations for other distributed simulation applications that are not based on the HLA, including the examples discussed in this section.

4.2 Performance Limiting Factors

Several factors play a part in determining the performance of distributed simulations. These factors can constitute potential bottlenecks during simulation execution. Some performance-limiting factors can influence both stand-alone simulations and distributed simulations. Some of these common factors include processor speed, memory size and I/O bandwidth (Park *et al.*, 2016; Hambrusch, Hameed and Khokhar, 1995). Other factors impact the performance of distributed simulations but are not performance concerns for stand-alone simulations. These additional factors include communication overheads, load balancing strategy, and the proportion of

computational workload which is inherently sequential and cannot be shared between multiple computing nodes (Lemeire, 2001). Among these factors, the communication overheads add on an obvious bottleneck that is absent from stand-alone simulations but can significantly influence the performance of distributed simulations (Kumar, 1992). As explained in Chapter 1, the performance framework developed from this work focuses on the dynamic relationship between computation and communication operations during distributed execution. Therefore, aspects of distributed performance related to communication are of particular interest. Chapter 3 has introduced various communication strategies which are to be applied in the experiments that will inform the development of the framework. However, it is evident that communication is not the only factor that influences execution performance. The following sections provide a general overview of factors that affect distributed performance and establish the place of communication among these other considerations. Using simplified scenarios, the influence of communication is compared with the influence of other performance-limiting factors. In the outlined scenarios, emphasis is placed on the amount of communication required to achieve ideal weak scaling. This approach is taken for convenience, as it makes it simpler to compare the different scenarios. However, it must be noted that ideal weak scaling is not the goal of the distributed simulation experiments which are to be conducted in Chapter 5.

4.2.1 Communication and Computation

The diagram in Figure 4.4, adapted from (Skillicorn and Talia, 1998), illustrates a simplified scenario showing how communication and computation together exert an influence on the total time required for distributed simulation execution. In this motivating example, the whole simulation is too large for a stand-alone computing node. It is estimated that the whole simulation could be completed in 120 hours of wall-clock time if stand-alone computing resources could support the large configuration. This 120-hour estimate is extrapolated from the performance of smaller simulation workloads. It is known that half of the full simulation workload can be completed on a stand-alone computing node in 60 hours of wall-clock time. Therefore, as shown in Figure 4.4, the full simulation workload has been divided between two identical computing nodes. In this scenario, communication is necessary during simulation execution to ensure correct outputs. To remove the

effect of other performance-limiting factors, the scenario incorporates the following simplifications:

- Perfectly balanced workloads.
- Half the simulation workload can be completed in half the time.
- The entire workload can be distributed. There is no portion of the workload that is inherently sequential.
- The impact of I/O operations are negligible.
- Communication does not overlap with computation.
- Time synchronization is free and does not add any extra costs.

From Figure 4.4, weak scaling is achieved at the point where the communication time equals computation time. At this point, the distributed completion time is equal to the stand-alone completion time. When communication time is less than computation time, performance improves, and the distributed completion time is less than the stand-alone completion time. On the other hand, when communication time exceeds computation time, performance degrades, and distributed completion time is longer than stand-alone completion time. From the simplified scenario, it is apparent that there is an upper limit on communication time if ideal weak scaling is to be achieved. The upper limit is equal to the difference between stand-alone computation time and distributed computation time.

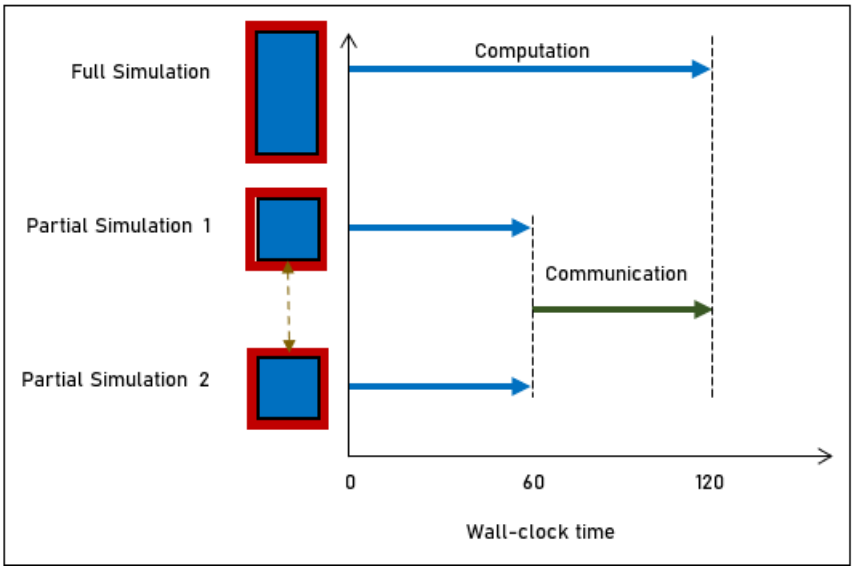

Figure 4.4: Communication and Computation

4.2.2 Communication and Load Balancing

Imbalanced workloads can negatively impact the performance of distributed simulations (Tan and Lim, 2004). The motivating example from the previous section can be extended to consider the impact of load balancing. Figure 4.5 illustrates a scenario in which assumption of perfectly balanced workloads no longer holds. The simulation workload is unevenly distributed. One computing node is responsible for 80% of the workload while the other handles 20%. Similar to the previous scenario, there is an upper limit on communication time if ideal weak scaling is desired. The upper limit is equal to the difference between stand-alone computation time and the computation time required for the largest distributed workload. As shown in Figure 4.5, a large workload imbalance can lead to a tight constraint on communication time. This motivating example has only considered the case where the workload remains static throughout simulation execution. However, dynamic load balancing is also a concern in distributed simulations where the computational workload can change on each node as the simulation progresses (De Grande and Boukerche, 2011). In such cases, the initial partitioning alone is not sufficient, and the workload needs to be continuously re-distributed to prevent performance from deteriorating during distributed execution.

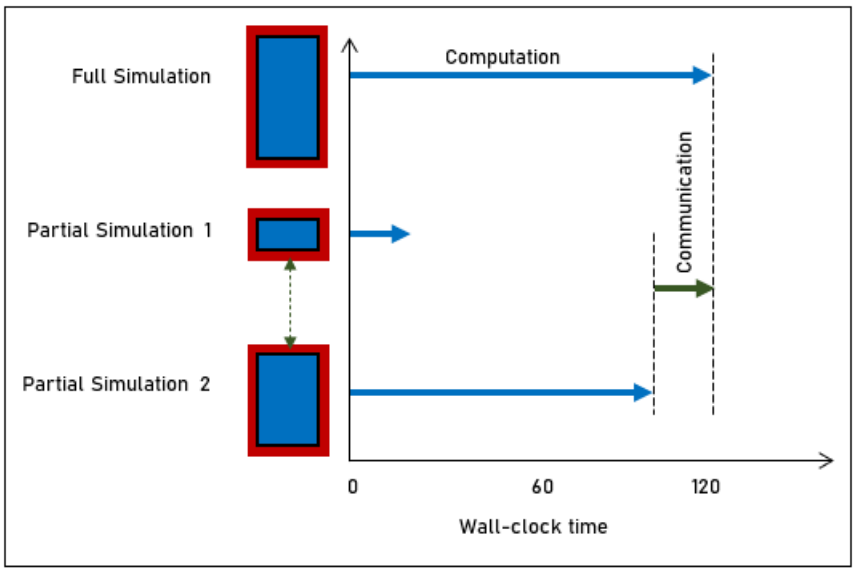

Figure 4.5: Communication and Load Balancing

4.2.3 Communication and Heterogeneous Workloads

Heterogeneous distributed simulations, where each computing node executes a different kind of simulation, can be considered as a special case for load balancing. As the different simulators perform different computations, their computation times may differ, even when considering the same set of simulation entities. Balancing the simulation workload in this case is less straightforward than the homogeneous case where simulation entities can be divided up between computing nodes. Figure 4.6 provides an illustration for this scenario. A stand-alone integrated simulator is introduced which combines the functionality of both simulators and completes the workload in the time required for both to complete. The impact on communication time is similar to the previous scenario with imbalanced workloads.

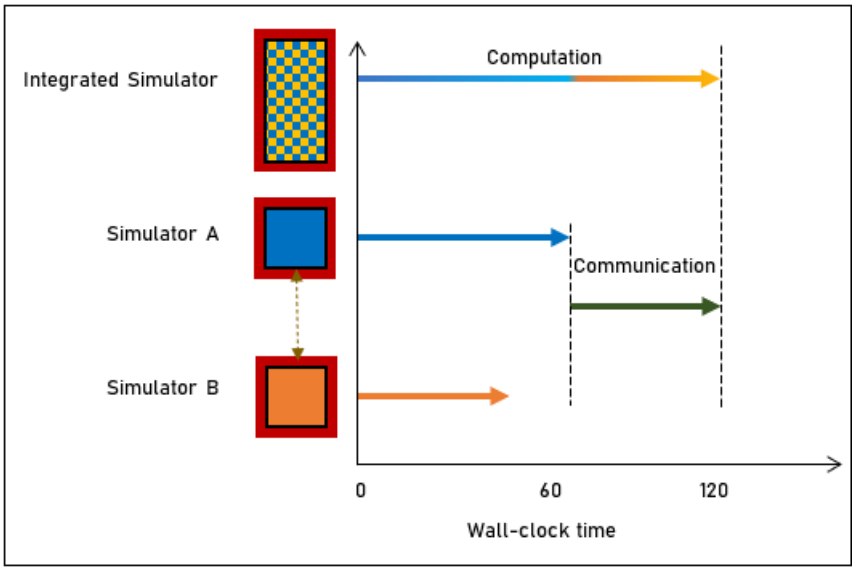

Figure 4.6: Communication and Heterogeneous Workloads

4.2.4 Communication and Sequential Workloads

As discussed in section 2.3, the inherently sequential portion of computational workload can significantly influence performance (Sun and Ni, 1993). The simple motivating example can also be extended to consider the impact of inherently sequential workloads. When a portion of the simulation workload cannot be distributed between computing nodes, that portion can be executed redundantly on all computing nodes. Alternatively, one node can perform the work and communicate the results to the others. However, this alternative strategy is not favourable as it adds

extra complexity and communication overheads. Figure 4.7 provides an illustration of a scenario in which the simulation workload includes an inherently sequential portion. The larger the sequential portion, the tighter the constraint on communication time.

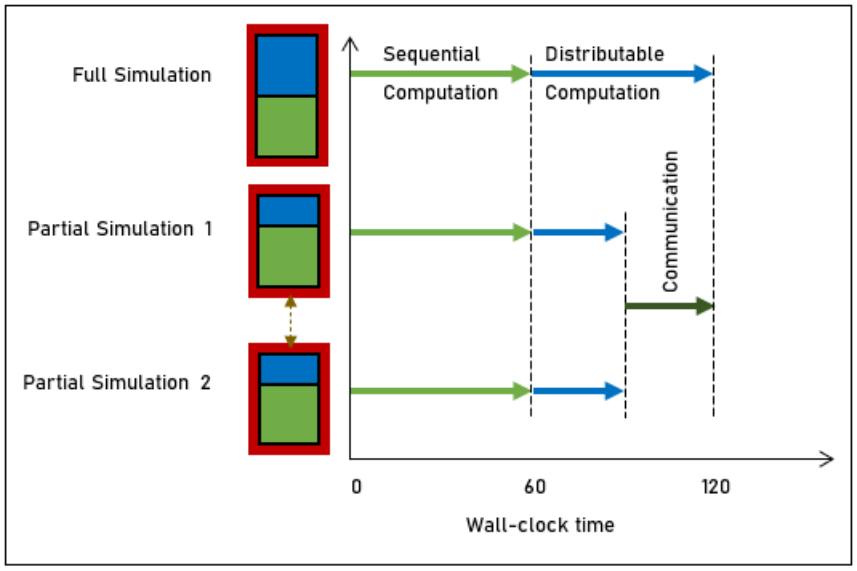

Figure 4.7: Communication and Sequential Workload

4.2.5 Communication and Time Synchronization

Time synchronization between nodes can incur additional overheads apart from the extra communication required to exchange timestamp information. The source of the additional overheads depends on the synchronization algorithm employed. As discussed in section 2.4, cascading rollbacks in optimistic algorithms can introduce significant overheads when out-of-order execution is detected and needs to be corrected by reverting the distributed simulation to an earlier point in time (Lubachevsky, Schwartz and Weiss, 1991). On the other hand, low lookahead values in conservative algorithms can significantly restrict parallel execution by holding some nodes back while waiting for others to reach a point where a safety guarantee can be provided (Nicol, 1993). The simplified diagram in Figure 4.8 illustrates this by adding synchronization costs in the motivating example. Here, a larger synchronization cost results in a tighter constraint on the communication time to achieve ideal weak scaling.

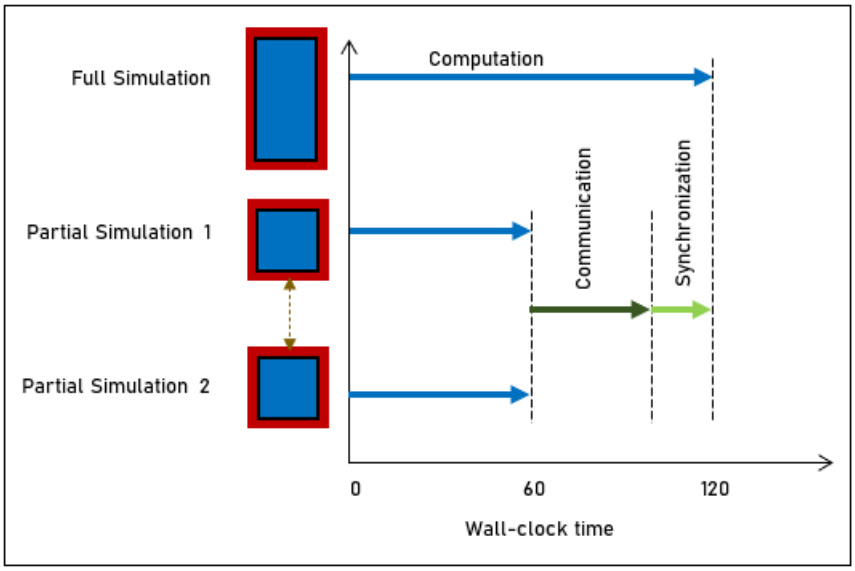

Figure 4.8: Communication and Time Synchronization

4.2.6 Communication and Latency Hiding

In cases where communication operations can be performed asynchronously, they can overlap with computations during distributed execution (Somani and Sansano, 1997). Using the latency hiding strategy, nodes can perform useful computational work while exchanging messages. This reduces time spent idling by nodes and lowers the impact of communication latency on performance. Figure 4.9 extends the motivating example to provide an illustration for the simplest scenario where communication and computation can completely overlap each other. As explained by Strumpen and Casavant (1994), the total distributed execution time in this scenario is the maximum of computation time and communication time. In cases where communication time is less than or equal to computation time, the communication can be completely hidden and incur no additional overheads.

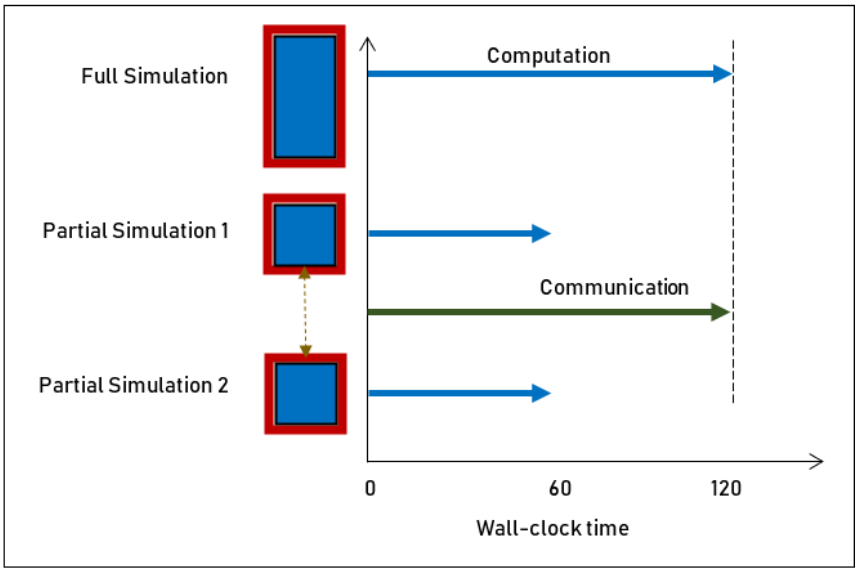

Figure 4.9: Communication and Latency Hiding

4.2.7 Communication and Number of Nodes

The number of computing nodes involved in a distributed simulation also influences communication time. Communication time tends to increase as the number of nodes increases and the number of communication paths involved goes up. This has been reported in experimental results such as those from Hambrusch, Hameed and Khokhar (1995) and Ponnusamy, Choudhary and Fox (1992). The relationship between communication time and the number of nodes depends on the properties of the interconnecting network. Network properties that influence communication time include bandwidth, latency, topology, and protocol. Communication time also depends on the distribution of communication workload among the nodes i.e., how much data each node needs to send and receive during distributed execution. This is in turn related to load balancing, as the workload partitioning strategy also determines for each node which data will be local and which will be external (De Grande and Boukerche, 2011).

4.2.8 Summary

The concerns discussed in the previous sections have presented the place of communication among other factors that are relevant for distributed simulation performance. This discussion will inform the development of the performance estimation framework, which will consider controlling various communication-related parameters and strategies.

5 CASE STUDIES

5.1 Overview

As discussed in Chapter 3, two case studies are selected from Urban Simulation domain for the purpose of experimentation. Case Study One is a homogeneous distributed simulation using the physical Building Energy Simulation tool, CitySim (Robinson 2012). Case Study Two is an extension to create a heterogeneous distributed simulation by adding the Building Occupancy Simulation tool, No-MASS. The simulation applications selected are normally used for stand-alone simulation experiments in their respective domains. Using these tools demonstrates that the approach employed in this work can be applicable to existing real-world simulation tools. The C++ source code of the CitySim tool is made available by the Sustaining Urban Habitats (SUH) project, of which this work forms a part. The open-source No-MASS simulation tool (Chapman, 2018) is also developed in C++ as part of the SUH project. Wang, Siebers, and Robinson (2017) have previously coupled No-MASS with the EnergyPlus Building Energy Simulation tool on a stand-alone computing node for small neighbourhood scale experiments.

5.2 Case Study One: Homogeneous Distributed Simulation

Case Study One is a distributed physical Building Energy Simulation using the CitySim tool. The HLA is employed to develop a distributed simulation composed of multiple computing nodes, each running an instance of CitySim, all working together in a single coordinated execution. As discussed in Chapter 2, the simulation objects in CitySim are building surfaces in an urban scene. Interactions exist between surfaces that have line-of-sight to one another. These surfaces are able to exchange various types of radiation during the simulation execution. Not all objects in the simulation have such interactions, and therefore some pairs of surfaces cannot exchange radiation with each other. In all the experiments conducted, it is assumed that the relationships between building surfaces remain static, and do not change

throughout simulation execution. This assumption implies, for example, that no new buildings are constructed, or changes made to existing buildings during the simulated period.

Building surface objects can be represented as vertices in a simple graph. Vertices connected by edges have interactions between them, while unconnected vertices cannot exchange radiation. As not all vertices can interact with each other, the graph is not fully connected, and connections between vertices are sparse. During distributed simulation, edges that cross boundaries between computing nodes represent communication paths between the nodes and determine which object attributes need to be exchanged. Edges weights indicate the size of data that needs to be exchanged. If a constant message size is assumed, the edges can be considered unweighted. The volume of communication between computing nodes during distributed simulation can be estimated by the number of edges between them. The direction of an edge signifies the direction of data transmission from sender to recipient. If it is assumed that two objects at opposite ends mutually require attribute updates from each other, the edges can be considered undirected as data transmission goes both ways. As the scenario is static, the set of vertices and edges remains the same throughout distributed simulation execution. Figure 5.1 provides an illustration of the graph representation for a scenario with four buildings, each having five surfaces. The large broken circles represent buildings, the small solid circles represent building surfaces, and the solid straight lines represent interaction relationships between surfaces. The illustration shows interactions between surfaces of the same building and interactions between surfaces of different buildings. If this scenario is to be distributed among two computing nodes, two buildings can be assigned to each computing node. The interactions between building surfaces assigned to different nodes will mark the data that needs to be exchanged between nodes. Each boundary-crossing interaction increases the total communication message count. For this scenario, the buildings can be partitioned into {A, C} on node one and {B, D} on node two. This would result in 6 boundary-crossing interactions, the minimum that can be achieved in this particular case.

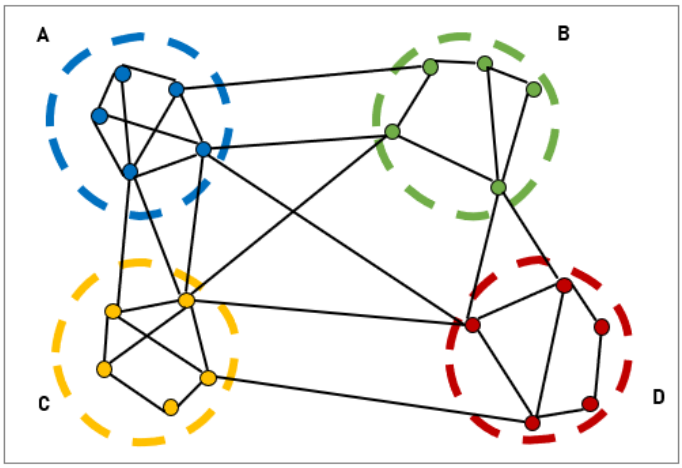

Figure 5.1: Illustration of building surface relationships

As described in Chapter 2, a CitySim building scene is a CityGML representation of a collection of 3D buildings, each composed of multiple surfaces (walls and roofs) and having one or more internal thermal zones (rooms). Obstructions in the building scene means that not all surface pairs are able to exchange radiation with one another. Therefore, radiation exchange relationships between building surfaces are captured in a sparse view-factor matrix. This sparse view-factor matrix is computed in a pre-processing step before starting the hourly time-stepped radiation exchange simulation that was illustrated by Figure 4.3. Consequently, the sparse view-factor matrix captures the relationships between building surfaces and can be represented with a graph as described in the previous section.

The graph of view-factor relationships can be used to facilitate the partitioning of the building scene for distribution among computing nodes. As the graph captures all the communication relationships that exist in the scene, it can be used as a guide to split groups of buildings into partitions of approximately equal size, while attempting to minimize the number of partition-crossing edges. This partitioning serves to balance the computational workload while attempting to minimize the number of messages that need to be exchanged between computing nodes. After initial partitioning, other methods can be employed to further reduce communication overheads during distributed execution. Some of the communication management strategies which address this concern have been discussed in Chapter 3.

Building scenes used for the experiments in this chapter make use of initial partitioning produced from the work of Zakhary et al. (2020) in which a Greedy Community Detection algorithm is employed. The surface interaction information from the sparse view-factor matrix is employed to group together surfaces that are densely connected, while minimizing the number of connections between different groups. While the actual interaction relationships are between building surfaces, partitioning is conducted at the building level. All surfaces belonging to the same building are assigned to the same partition. This constraint is important because surfaces in the same building tend to have strong relationships with one another. It helps to reduce the total number of cross-partition edges, and therefore reduce the communication message count. Figure 5.2 shows a building scene partitioned into 12 parts by Zakhary et al. (2020). The dots show the position of buildings in the scene, with separate colours for different partitions. The solid straight lines show examples where relationships exist between building surfaces.

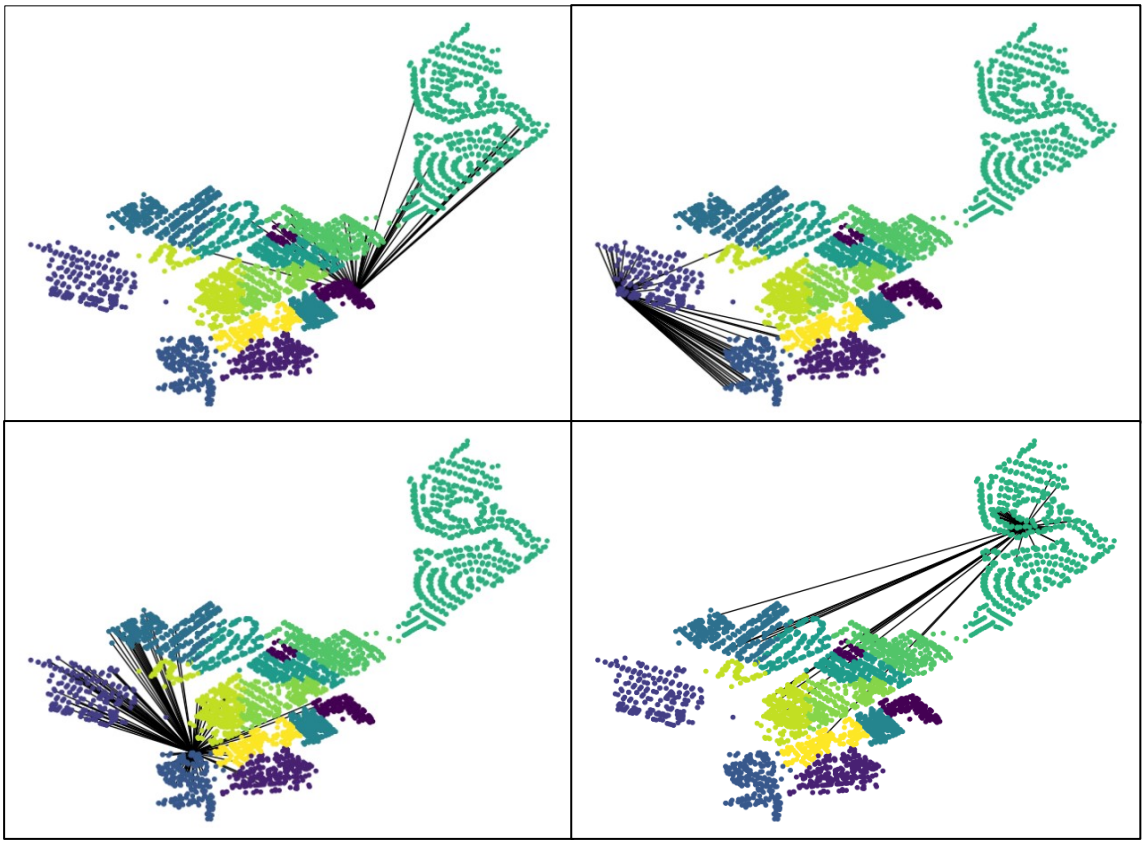

Figure 5.2: Building scene with 12 partitions scene showing some interactions

In the HLA experiments in this chapter, each partition will be assigned to a separate CitySim-Federate, which will own all the partition surfaces and be responsible for simulating them and publishing relevant attribute updates. To ensure correct local computations, each CitySim-Federate will keep track of external object attributes by attribute reflection received updates. Therefore, the area of interest of each CitySim-Federate will include local objects and external objects that interact with local objects. Figure 5.3 shows a conceptual illustration of the areas of interest of CitySim-Federates. The small solid circles represent surface objects, the thick solid lines indicate partition boundaries, and the thick broken lines show the extent of the area of interest of each CitySim-Federate. These areas of interest overlap depending on the interaction relationships between surface objects.

From Figure 5.3, the surface objects that fall within the overlapping areas of interest are the only ones that have an impact on communication overheads during distributed simulation execution. Surfaces that do not fall within the overlapping areas of interest only interact with other local surfaces. Therefore, their attributes do not need to be shared with other CitySim-Federates. Indeed, no record needs to be kept of them outside their own CitySim-Federate. Therefore, they need not be registered with the central HLA RTI. Only objects within overlapping areas of interest need to be registered with the RTI. This is necessary for establishing ownership, as well as allowing publishing or subscribing to relevant attribute updates.

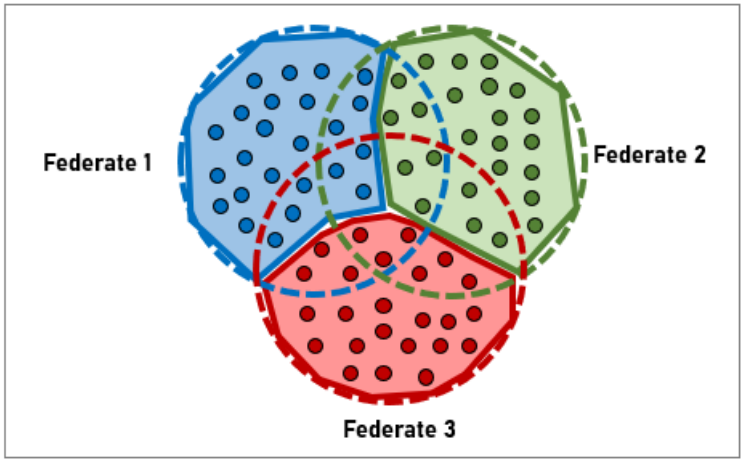

Figure 5.3: Conceptual illustration of CitySim-Federates' areas of interest

The conceptual diagram in Figure 5.4 illustrates the execution of a single time-step in the distributed CitySim HLA Federation. As discussed in Chapter 2, each hourly time-step in a CitySim simulation is composed of four calculations: shortwave calculations (SW), daylight calculations (DL), longwave calculations (LW) and thermal zone calculations (TH). These four have dependencies on one another in the order illustrated by Figure 5.4. Thermal zone calculations depend on results from longwave calculations within the same time-step. In turn, longwave calculations depend on shortwave computations within the same time-step. All calculations depend on the calculation results from the previous time-step, directly or indirectly. Because of these dependencies, data needs to be exchanged between CitySim-Federates within time-steps after partial calculations have been completed. This is necessary to ensure the correctness of the simulation outputs. As shown in Figure 5.4, data is exchanged between federates during the shortwave, daylight and thermal zone calculations. However, no data needs to be exchanged between CitySim-Federates during longwave calculations. This is an effect of the scene partitioning strategy. Thermal zone calculations only depend on the longwave results from surfaces in the same building. Consequently, data exchange after longwave calculations is not necessary for correct thermal zone calculations as all surfaces belonging to the same building are owned by the same CitySim-Federate.

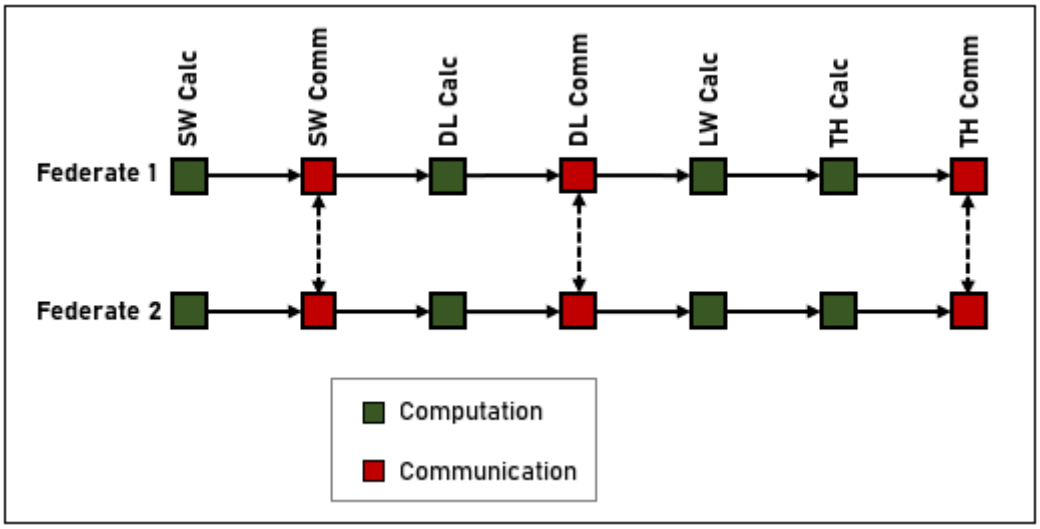

Figure 5.4: Conceptual diagram of CitySim HLA Federation execution

5.2.1 Case Study One: HLA Federation

The class diagram in Figure 5.5 shows a summary of the HLA object classes contained in the FOM of the CitySim Federation. This represents the data that is expected to be exchanged between federates during federation execution. This includes the single Surface object class. Surfaces are persistent objects which will exist throughout the federation execution. Messages concerning surface attributes will be exchanged from timestep to timestep. For these reasons, surfaces are represented by HLA object classes as opposed to HLA interaction classes. Interaction classes are normally used for transient messages which will not persist throughout the federation execution. The Surface object class includes three attributes, one for each of the properties of the scene surface objects that needs to be simulated. Messages containing updates of these attributes will be exchanged between federates during distributed execution. The FOM attributes include SW, DL and TH. The FOM does not include a LW attribute as these do need to be exchanged between CitySim-Federates.

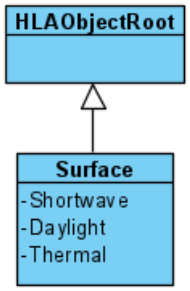

Figure 5.5: Class diagram showing HLA FOM objects for Case Study One

The sequence diagram in Figure 5.6 shows the exchanges between the RTI, the CitySim-Federate Ambassador and the CitySim-Simulator during distributed execution. The CitySim-Simulator is the component that performs the actual Building Energy Simulation, while the CitySim-Federate Ambassador is the component that coordinates message exchanges between the RTI and the CitySim-Simulator. The CitySim-Federate Ambassador sends and receives attribute updates to and from the RTI on behalf of the CitySim-Simulator. The CitySim-Federate Ambassador also performs all the signalling required for creating and initializing the federation, for time synchronization, and for destroying the federation at the conclusion of the distributed simulation. A CitySim-Federate is a single unit composed of a CitySim-Simulator and a CitySim-Federate Ambassador.

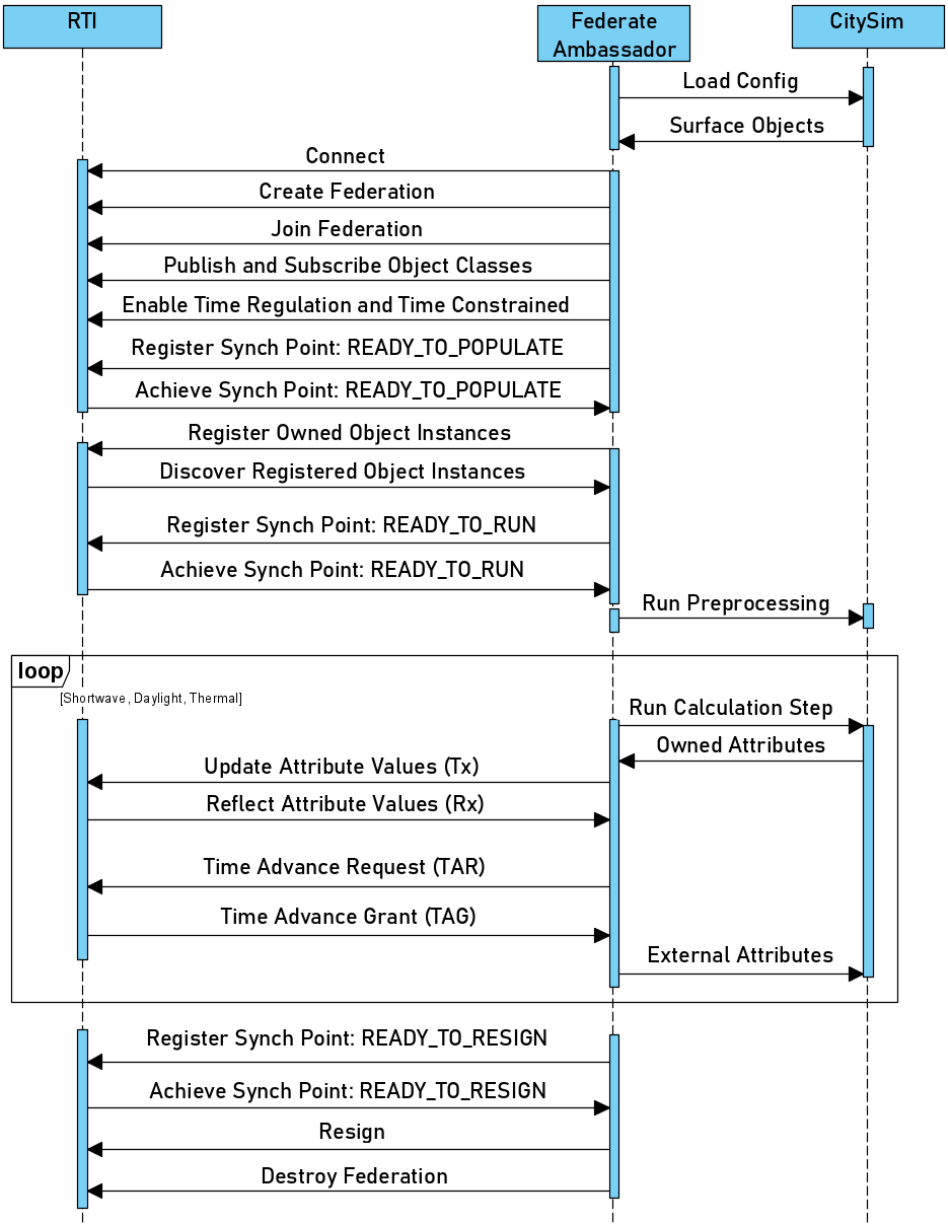

Figure 5.6: Sequence diagram of HLA federation execution for Case Study One

Although the sequence diagram in Figure 5.6 only shows one CitySim-Federate, a federation includes multiple instances of such units, passing messages in the same sequence described by the diagram. Certain functions such as creating and destroying the federation can only be performed once. The first CitySim-Federate to start performs the signalling required to create the federation, using the FOM. Subsequent CitySim-Federates need only to join the existing federation. The last federate to resign performs the signalling required to destroy the federation. The federation cannot be destroyed while other federates are still joined.

The signalling required for time synchronization involves sending Time Advance Requests (TARs) to the RTI and receiving Time Advance Grants (TAGs) in return. By this means, the RTI coordinates the execution of CitySim-Federates, granting TAGs when the CitySim-Federates can proceed safely to the next time step and all required messages have been delivered. Figure 5.7 shows the class diagram for the HLA CitySim-Federate.

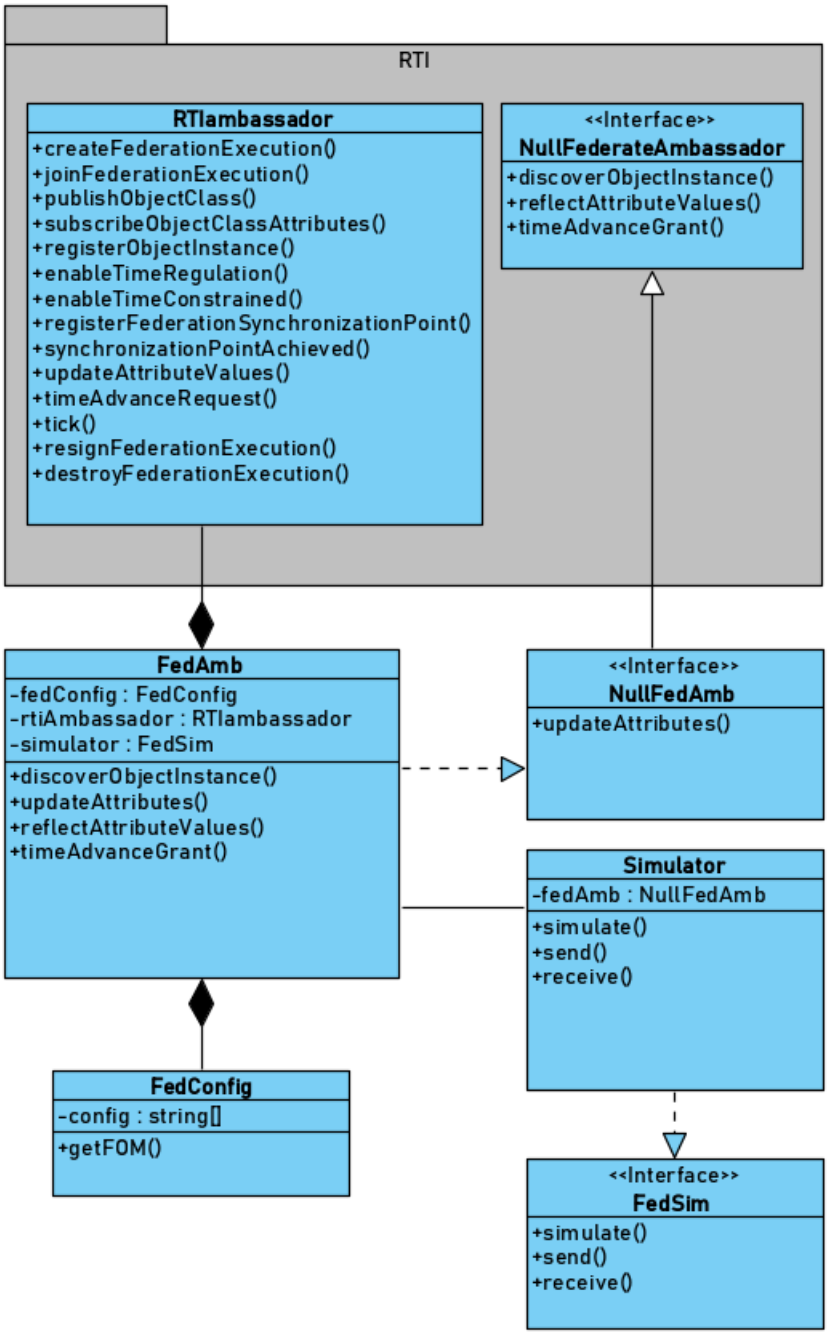

Figure 5.7: Class diagram for CitySim-Federate

5.2.2 Initial Experiments

5.2.2.1 Experimental Setup

Initial experiments carried out with the HLA CitySim Federation were conducted on a network of 12 virtual machines running CentOS Linux 7.4, each with 8GiB of RAM and 2x Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz. The virtual machines are connected to each other over a 10 Gigabit Ethernet LAN. The experiments make use of the open source Portico RTI (Portico, 2020) to support federation execution. As discussed in section 3.2, the Portico RTI implementation was selected for various reasons, including its open-source status and active development community. In the experiments, each computing node is responsible for executing a separate HLA CitySim-Federate. The maximum number of computing nodes involved in the experiments is 12 because the initial partitioning obtained from the work of Zakhary et al. (2020) uses 12 partitions.

5.2.2.2 Building Scenes

The initial experiments were conducted using a CityGML scene composed of 2,980 buildings representing an area of the township of Sneinton in Nottingham, UK. This CityGML scene is obtained from the work of Rosser *et al.* (2019) and is available in two types: a "simple" scene and a "complex" scene. The two types represent the same group of buildings with varying degrees of fidelity. The complex scene has about five times the number of surfaces in the simple scene. In the simple scene, all buildings are represented as simple shoebox-like shapes. This scene has a total of 25,514 building surfaces. The complex scene represents the building structures in more realistic detail, raising the total number of building surfaces to 122,847. To enable comparison between experiments, the same partitioning scheme is used for both the complex scene and the simple scene. As the partitioning is done at the building level and both scenes contain the same buildings, a partitioning created for the complex scene can also be used for the simple scene. Each CitySim-Federate is given responsibility for a different partition.

5.2.2.3 Experiment Run Configuration

The initial experiments were run for one year of simulation time. Every CitySim simulation run also includes an additional fifteen-day warm-up period that is discarded at the beginning. In these initial experiments, the number of partitions was

varied between 2 and 12. In experiments where fewer than 12 partitions were required, the original 12 were re-combined to produce the desired number of partitions, maintaining some of the original boundaries. The source code of the CitySim-Federate implementation was instrumented to measure the time spent on computation and total time for completion on each computing node. The extra processing overheads added by this code instrumentation was small, typically less than 1% of additional computation time. To illustrate how computation time and communication time are measured, Figure 5.8 shows the events that occur for a computation phase followed by a communication phase, assuming two CitySim-Federates with balanced workloads. Each CitySim-Federate completes its computation, transmits data to the RTI and waits to receive data from other federates via the RTI. After all required data has been exchanged, the RTI grants a time advance to all federates, allowing them to proceed. This sequence of a computation phase followed by a communication phase and a synchronization barrier follows the BSP execution pattern described in section 2.8.

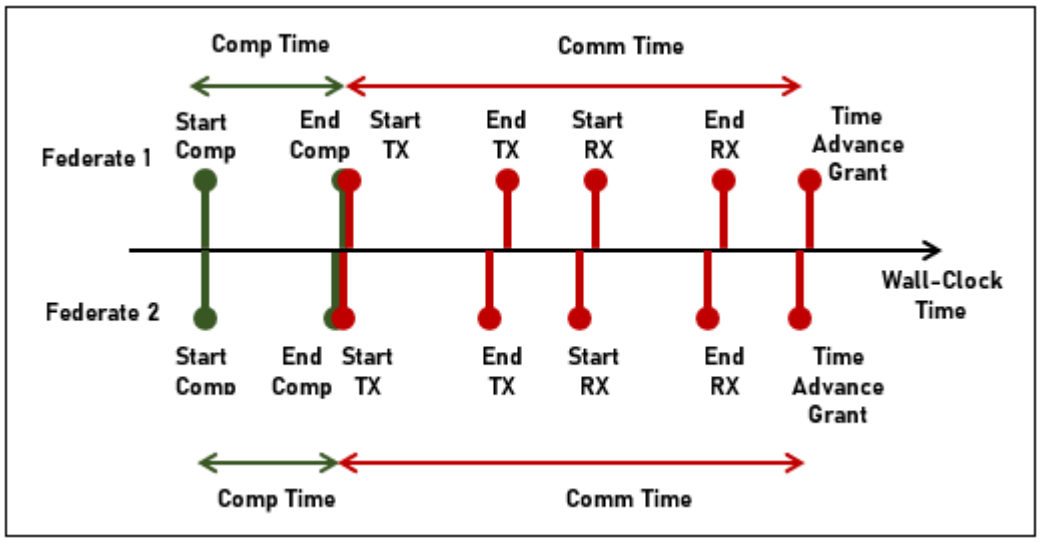

Figure 5.8: Computation and communication events within an hourly time-step

5.2.2.4 Initial Results

Figure 5.9 and Figure 5.10 present the results of the experiments for the simple scene and the complex scene. Figure 5.9 compares the measured computation time and communication time for the simple scene. The experiment runs vary the number of computing nodes from 2 to 12. Figure 5.10 shows the same comparison for the complex scene. For the complex scene, the memory requirements exceed the

resources available on 2 computing nodes. Therefore, the minimum number of nodes is 4 in this case. From the previous illustration in Figure 5.8, communication time is measured on each node from the point where computation ends until a TAG is received from the RTI. Error bars in Figure 5.9 and Figure 5.10 show differences in computation time between federates due to imbalanced workloads. Initial scene partitioning cannot always perfectly fulfil the multiple objectives under consideration, and therefore attempts to compromise for a satisfactory trade-off between balancing workloads and minimizing communication. In both figures, the results presented examine the variation between nodes during a single execution of the distributed simulation. Although both graphs have error bars, the differences between federate workloads are more visible in the complex scene.

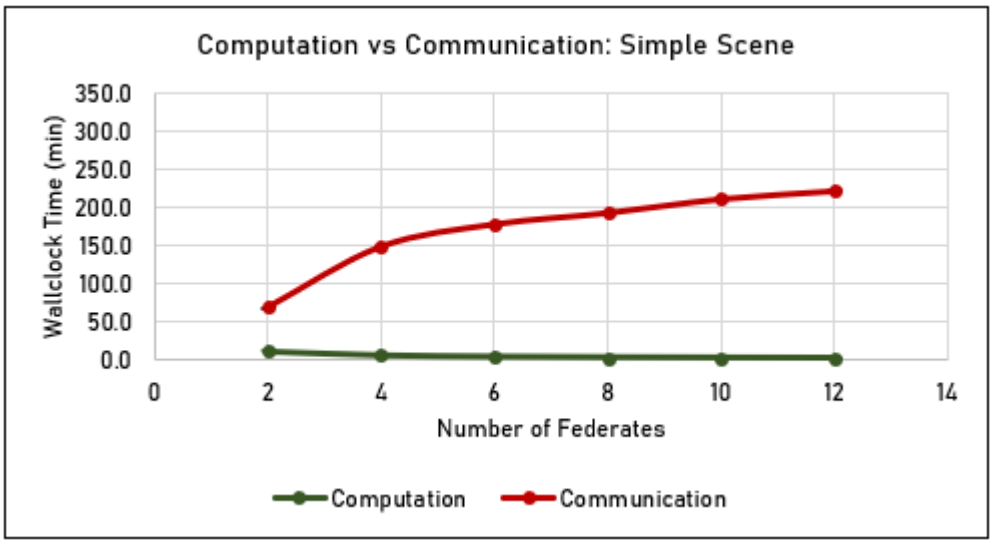

Figure 5.9: Computation vs communication wall-clock time (simple scene)

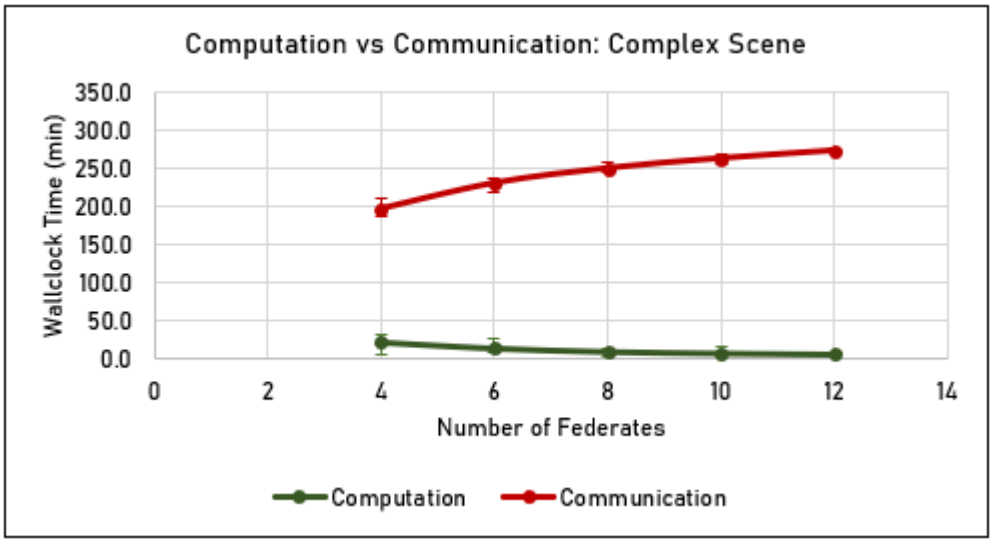

Figure 5.10: Computation vs communication wall-clock time (complex scene)

The trends from Figure 5.9 and Figure 5.10 show communication time increasing with the number of nodes while computation time decreases. The decreasing computation time is due to smaller workloads when more nodes are involved. In all cases, communication time exceeds computation time. As discussed, using the motivating examples in Chapter 4, this case is not ideal. However, it appears from the results that communication costs are lowest when the minimum number of nodes are involved: 2 nodes for the simple scene, and 4 nodes for the complex scene. The initial results suggest that when communication overheads are dominant, it is favourable to use the minimum number of nodes that can support the workload in order to avoid communication overheads. It is worth noting that while the number of surfaces has increased about five-fold from the simple scene to the complex scene, the communication overheads only grow by about 1.3 times. In the 4 federate experiment, the average communication time is 150 min for the simple scene and about 200 min for the complex scene. In the 12 federate experiment, the average communication time is about 220 minutes for the simple scene and 270 minutes for the complex scene.

For the same initial experiments, Figure 5.11 and Figure 5.12 show how computation time varies with assigned workload. The error bars indicate variations between federates. The trend confirms that computation time is proportional to the assigned workload for this simulation application.

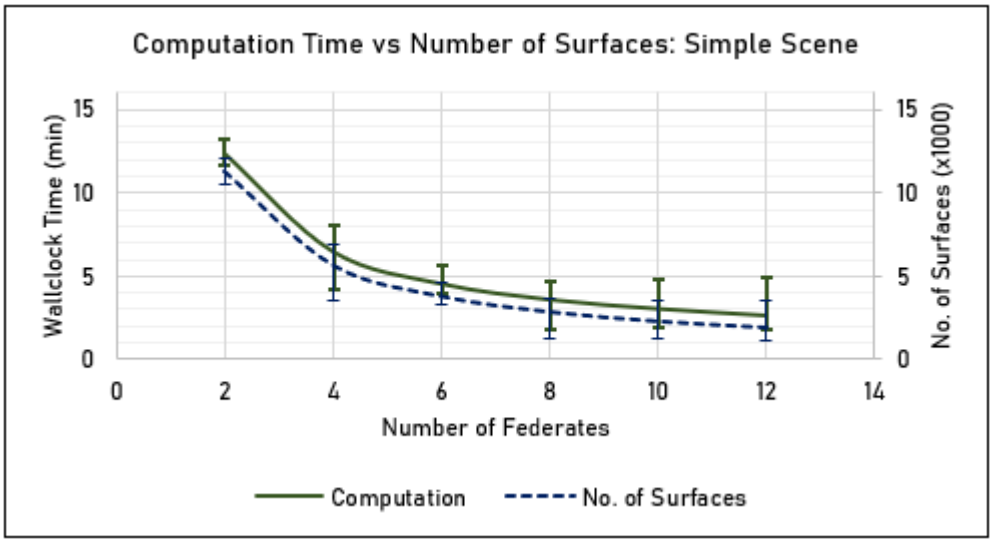

Figure 5.11: Computation wall-clock time vs number of surfaces (simple scene)

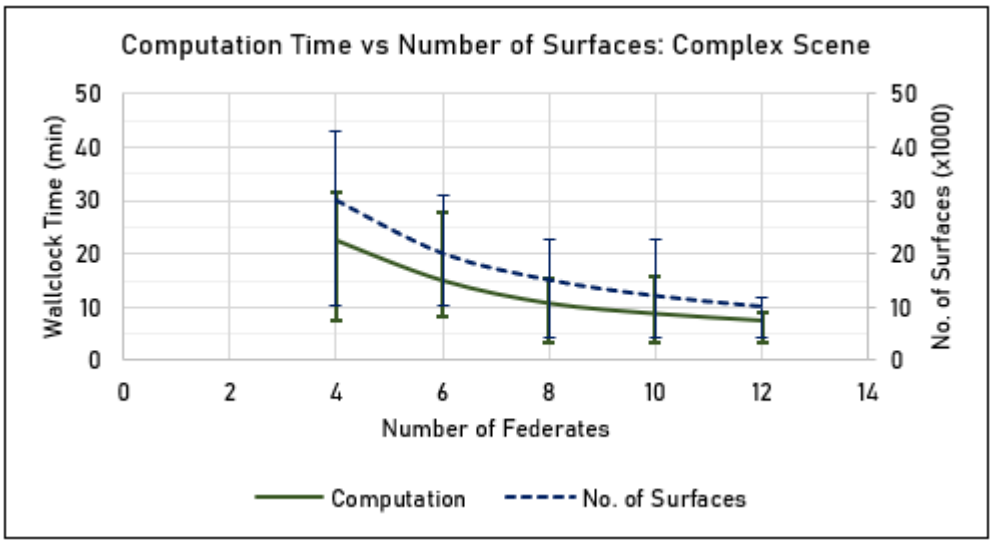

Figure 5.12: Computation wall-clock time vs number of surfaces (complex scene)

Figure 5.13 and Figure 5.14 show the trends in communication time, together with the number of inter-federate links (cross-node interactions) and total number of published surfaces (surfaces in the overlapping areas of interest). The number of inter-federate links is larger than the number of published surfaces as a single published surface can interact with more than one external surface.

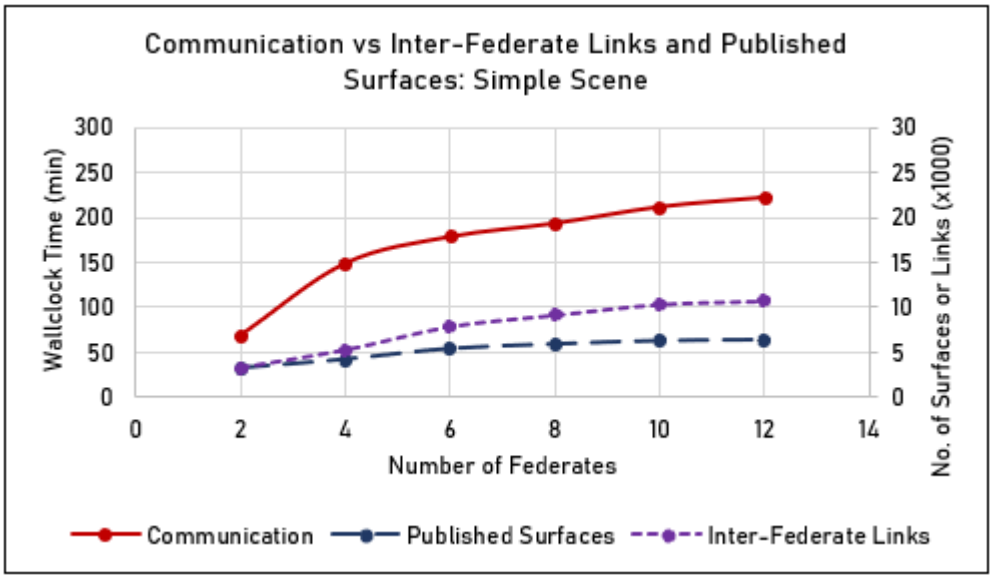

Figure 5.13: Communication vs published surfaces and inter-federate links (simple scene)

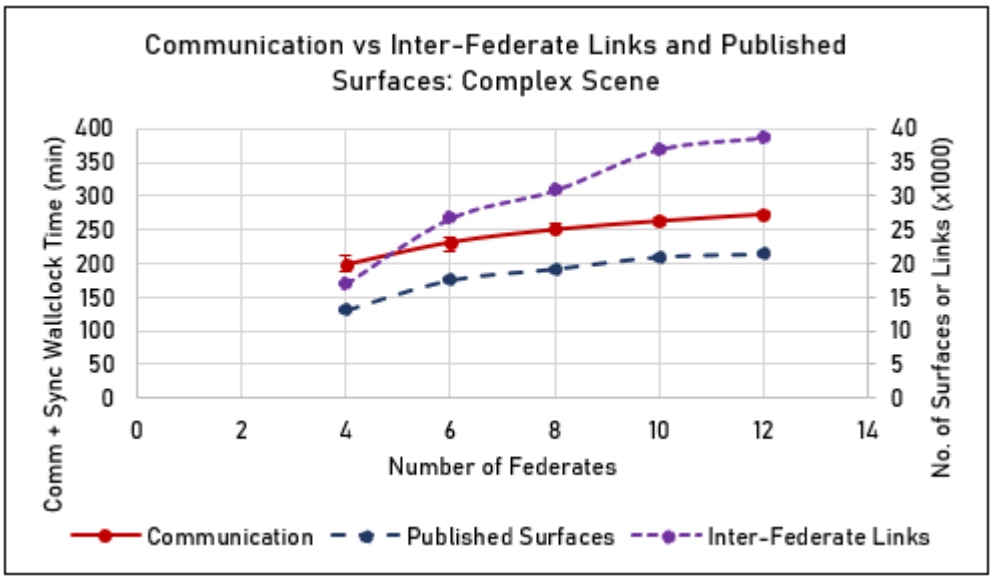

Figure 5.14: Communication vs published surfaces and inter-federate links (complex scene)

As previously displayed in Figure 5.8, the communication time for each node is measured as time not spent performing computation. This measurement incorporates multiple components:

1. Time spent actively transmitting attribute update messages to the RTI.
2. Time spent passively idling while waiting for other federates to complete their computational workloads.

3. Time spent actively receiving attribute reflection messages from the RTI.
4. Time spent actively sending and receiving signalling information to/from the RTI, such as for time advancement.

While actively sending and receiving messages (1, 2, 4) can be considered useful work, idling passively (3) cannot. Therefore, it is important to estimate the portion of the measured communication time that is actually spent idling. The boxplots in Figure 5.15 and Figure 5.16 show the variations between nodes in the 12-federate experiments for the simple and complex scenes. In the simple scene experiment, the range in computation times shows the slowest federate lagging behind the fastest by about 3.1 min, accumulated over the duration of the simulation. This lag can be used as an estimate for the idle time of the fastest federate, measured as part of its communication time. As the total measured communication time exceeds 220 min for all federates, the idle time of 3.1 min represents a small fraction (~1.4%) of the total communication time. This suggests that the majority of the measured communication time is spent actively sending and receiving messages. In the complex scene experiment, the lag in computation time is 5.8 min between the fastest and slowest federates. This is a small fraction (~2.1%) of the total communication time which exceeds 270 min for all federates. Similar to the simple scene, the major portion of the measured communication time is due to active communication, and only a minor portion can be explained by idling due to imbalanced workloads.

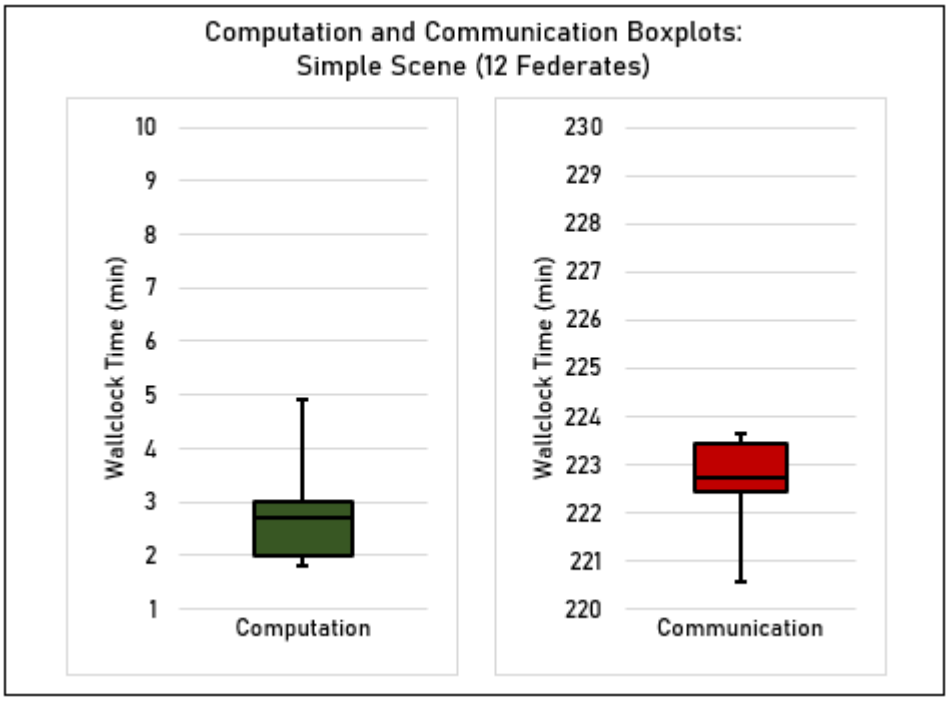

Figure 5.15: Computation and communication wall-clock time boxplots (simple scene)

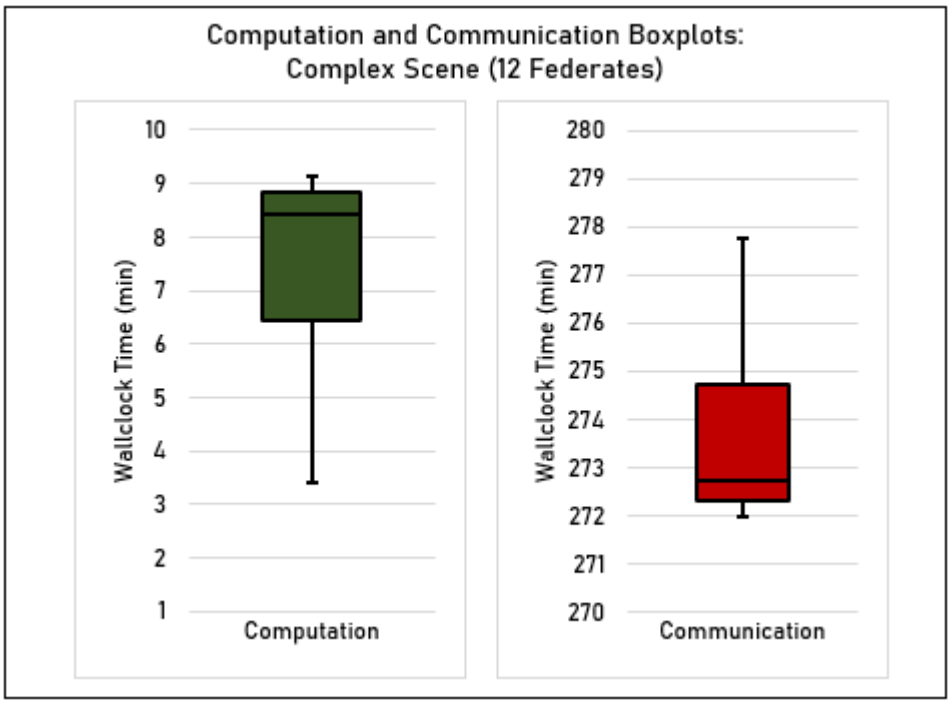

Figure 5.16: Computation and communication wall-clock time boxplots (complex scene)

As active communication time is the main performance bottleneck, the analysis of experimental results in subsequent sections will focus specifically on communication

time rather than total simulation time. Following this approach, it is useful to make direct comparisons between the measured communication times for various distributed simulation scenarios which involve different numbers of computing nodes or employ alternative communication strategies. As this explicitly accounts for communication time, it is more convenient than other measures, such as speedup, which implicitly account for communication time as part of total simulation time without separating it from computation time. Additionally, the definition of speedup requires the execution time on a single node as well as the execution time on multiple nodes. However, execution on a single node is not always possible when dealing with large-scale experiments.

5.2.3 Reduced Building Scene

The experiments in the previous section have shown that communication overheads can significantly influence the performance of distributed Building Energy Simulations. To further reduce communication overheads, subsequent experiments will apply the communication strategies discussed in Chapter 3, and measure any performance gains made. For communication strategies that can result in a loss of output accuracy, the resulting errors in simulation outputs will be measured as well. The initial simple and complex experiments in this section completed execution in 1.5 to 5 hours of wall-clock time. In order to allow more experiments to be conducted, the lengthy wall-clock time for each run needs to be reduced. For this purpose, a reduced building scene is introduced. This reduced building scene is a subset of the simple scene, consisting of 51 buildings with 320 surfaces. The run length of the experiments is also reduced from one year to four months. After running experiments using this reduced scene, the findings can be applied to the full scene of 2,980 buildings.

5.2.4 Message Elimination Experiments

For these experiments, the message elimination strategy introduced in Chapter 3 was implemented for Case Study One. The experiments were conducted on the reduced 51-building scene with 320 surfaces, distributed over two computing nodes using the experimental setup described in the previous section. The experiments begin with a

base case, in which communication occurs normally during every timestep. This is followed by an edge case experiment during which all attribute update messages are eliminated. Only communication related to time management is permitted. This edge case experiment serves to measure the amount of communication time that is not due to attribute updates. Subsequent experiments vary the period between successive attribute updates. The length of the period varies from 2 timesteps to 12 timesteps. The period values used in the experiments are chosen because they are factors that align with a day length of 24 hours. The results are presented in the following sections by comparing the normal case where communication occurs at every timestep with the other cases where communication is less frequent. As this is a lossy strategy, the comparisons made include the accuracy of the outputs as well as the wall-clock times involved, exploring the trade-off between these two concerns for this simulation.

Figure 5.17 shows the impact of the message elimination strategy on computation and communication time. The edge case with no communication is labelled *00* on the chart. The trend shows communication time reducing as the communication points grow farther apart due to larger update period values. As federates do not exchange attribute updates in time-steps between communication points, only synchronization messages are exchanged, resulting in fewer messages and a reduction in communication overheads. From Figure 5.17, the normal base case *01* has a communication wall-clock time of about 200 s. By contrast, the edge case *00*, which eliminates all attribute updates, has a communication wall-clock time of about 20 s. The edge case establishes a lower limit on the communication time necessary to execute the distributed simulation. The trend in Figure 5.17 shows that the communication roughly halves when the period between communication points doubles. From the normal (01) base case of 199 s to the 2-timestep period (02) case of 105 s, to the 4-timestep period (04) case of 60 s to the 8-timestep period (08) case of 30 s. This roughly reflects the fact that the number of attribute update messages has been reduced by half.

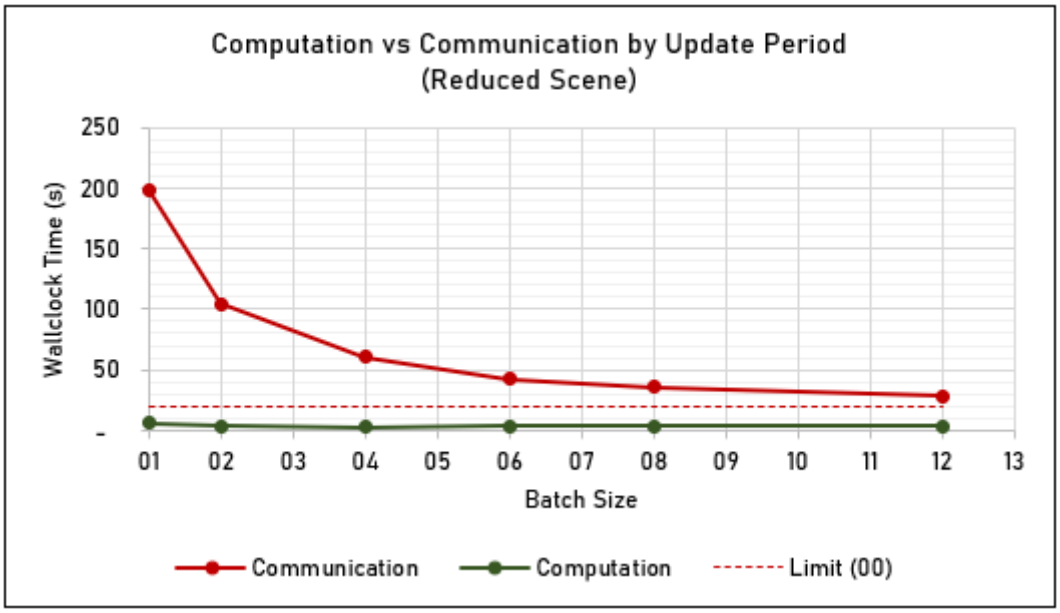

Figure 5.17: Effect of message elimination on communication time

Although the message elimination strategy improves execution performance, it also reduces simulation output accuracy due to its lossy nature. However, different simulation outputs are affected differently. The degree of error also varies from one surface to another. This section will examine how the four main outputs from the simulation are impacted: the surface shortwave (SW), daylight (DL) and longwave (LW) outputs as well as the thermal zone temperature (TH) outputs.

The boxplots in Figure 5.18 and Figure 5.19 show the overall distributions of the four main outputs for all surfaces in the top row. In the middle row, the boxplots show the absolute error of all surfaces in each experiment compared to the normal base case *01*. The bottom row provides a bar charts displaying the mean absolute error for all surfaces. The diagrams confirm that the largest output errors occur in the edge case experiment *00* where no attribute updates occur. This applies to all four outputs but is especially noticeable for the SW and DL outputs. Although the edge case *00* represents the best-case scenario for communication time, it also represents the worst-case scenario for simulation output accuracy. This confirms that communication between CitySim-Federates is important for ensuring accurate results.

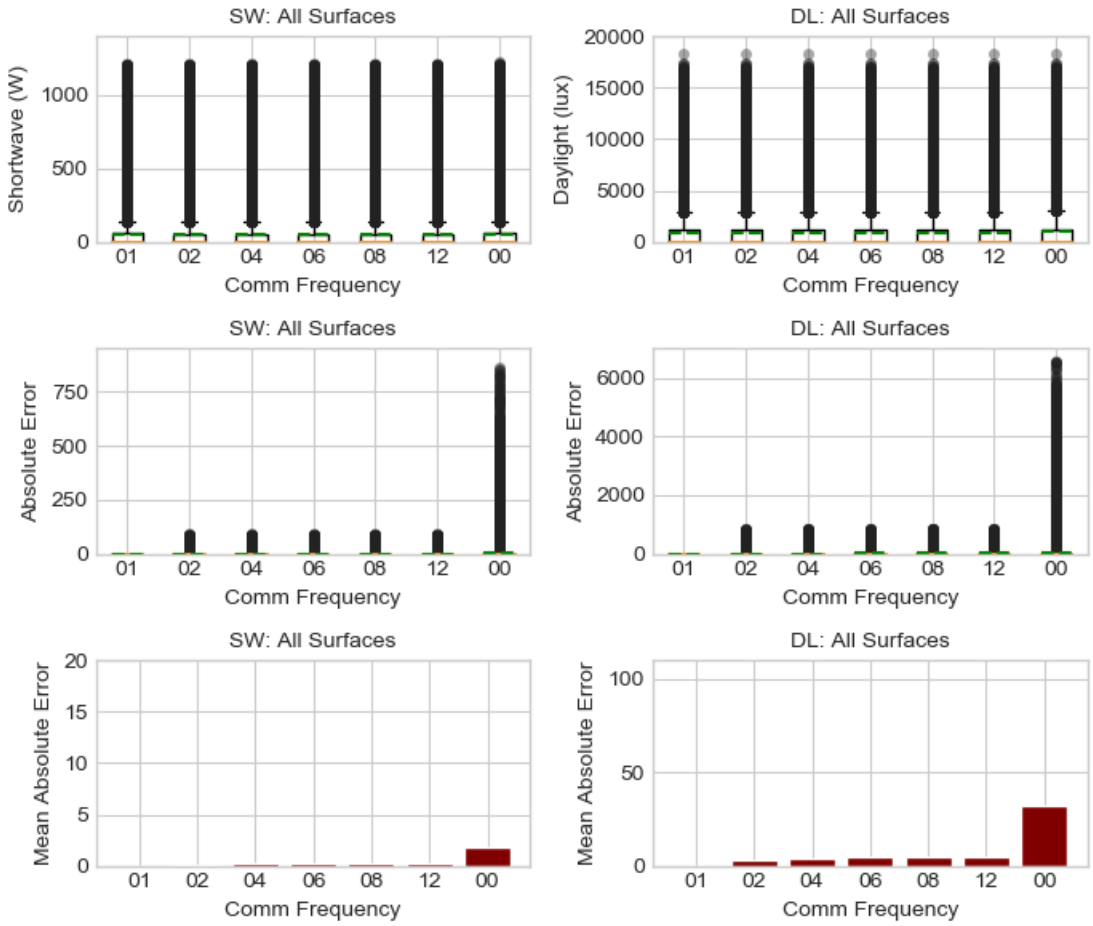

Figure 5.18: Summary of SW and DL output errors for whole scene

Apart from the edge case *00* which has no attribute updates, the mean absolute errors for the other experiments are small when averaged for all surfaces in the scenes. This is illustrated by the bar charts in the bottom rows of Figure 5.18 and Figure 5.19. However, the absolute error boxplots in the middle rows show that the absolute errors are significant for many individual surfaces in the scene. From these results it can be inferred that all surfaces in the scene are not equally affected by the eliminated attribute update messages. Whether the output errors for individual surfaces are important or not depends on the purpose for executing the distributed Building Energy Simulation. If only macro-level average values over the entire scene are required, then individual errors would only be significant to the extent to which they have an impact on the overall average. However, if the Building Energy Simulation is for a purpose such as exploring the impact of retrofitting options for individual buildings, the surface output errors can be considered important regardless of the negligible impact they have on the overall scene average. This discussion will assume that the output errors for individual surfaces are important.

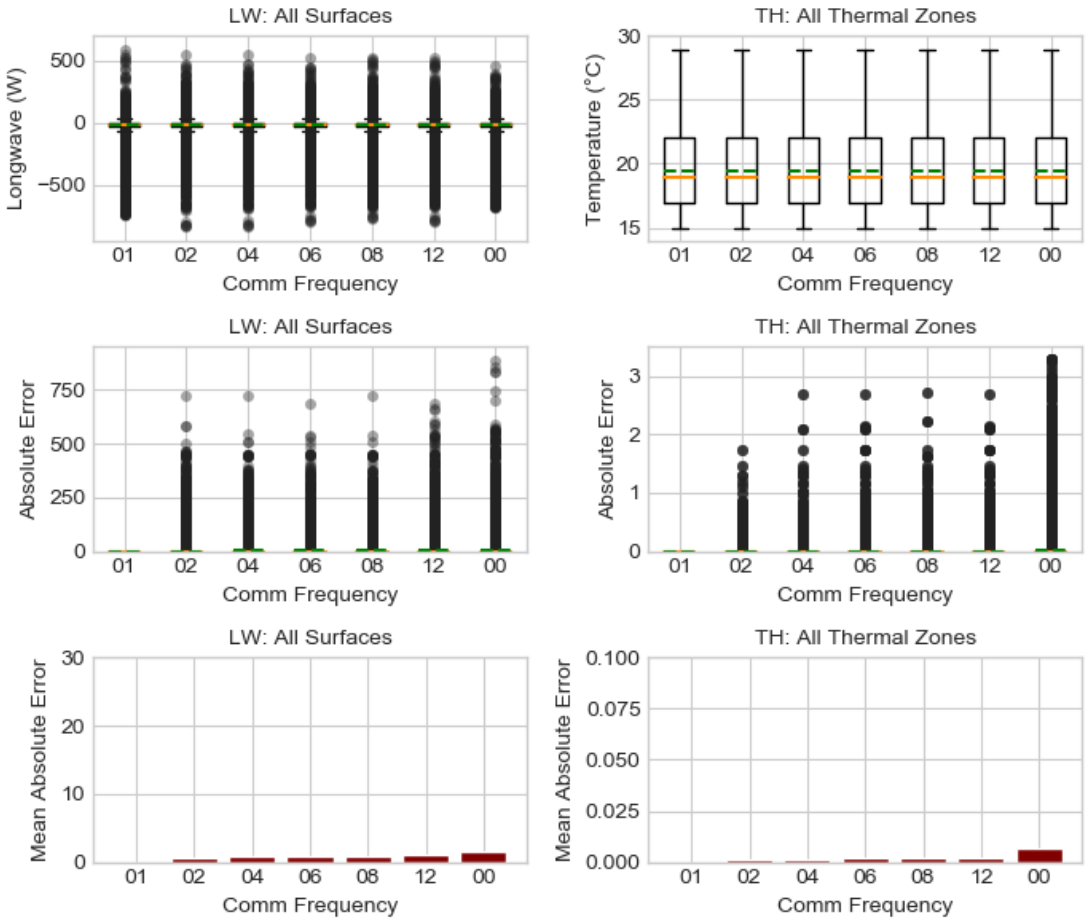

Figure 5.19: Summary of LW and TH output errors for whole scene

While some surfaces in the scene have small output errors and others have large output errors, there does not appear to be a simple rule to reliably predict which surfaces will produce large errors and which ones will not. Intuitively, it might be supposed that surfaces which have interactions with external surfaces rely on attribute updates for correct computations. Therefore, they are more likely to produce large errors when the required attribute updates have been eliminated. On the other hand, surfaces which only have local interactions should produce small errors. However, the experiments show that this intuition does not generally hold for all surfaces. Some of the surfaces that produce the largest errors have only local interactions, while some surfaces that have external interactions only produce small errors. The number of external interactions a surface has does not reliably predict whether its associated output errors will be large or small. However, the experimental results indicate that surfaces in the same building tend to have similar degrees of errors. If the errors produced in a surface are large, errors for the other surfaces in the building are usually large as well. However, this observation does not

explain all the results. In some buildings, some surfaces have large errors while others have small errors. The diagrams from diagrams from Figure 5.20 to Figure 5.23 illustrate two surfaces in the reduced scene. The surface on the left has external interactions but produces small output errors, while the surface on the right has only local interactions but produces large output errors.

From the whole scene boxplots in the middle rows of Figure 5.18 and Figure 5.19, the largest surface errors are significant even when the edge case *00* is excluded. This is true for all outputs. The SW absolute error boxplots show a maximum of about 800 for the edge case and 80 for the other cases. As shown in the top rows of Figure 5.18, SW values in the experiments range from 0 to approximately 1200. Therefore, absolute errors of 800 and 80 respectively correspond to 66% and 6% of the SW value range. Figure 5.20 compares the SW output errors for two surfaces with different error responses.

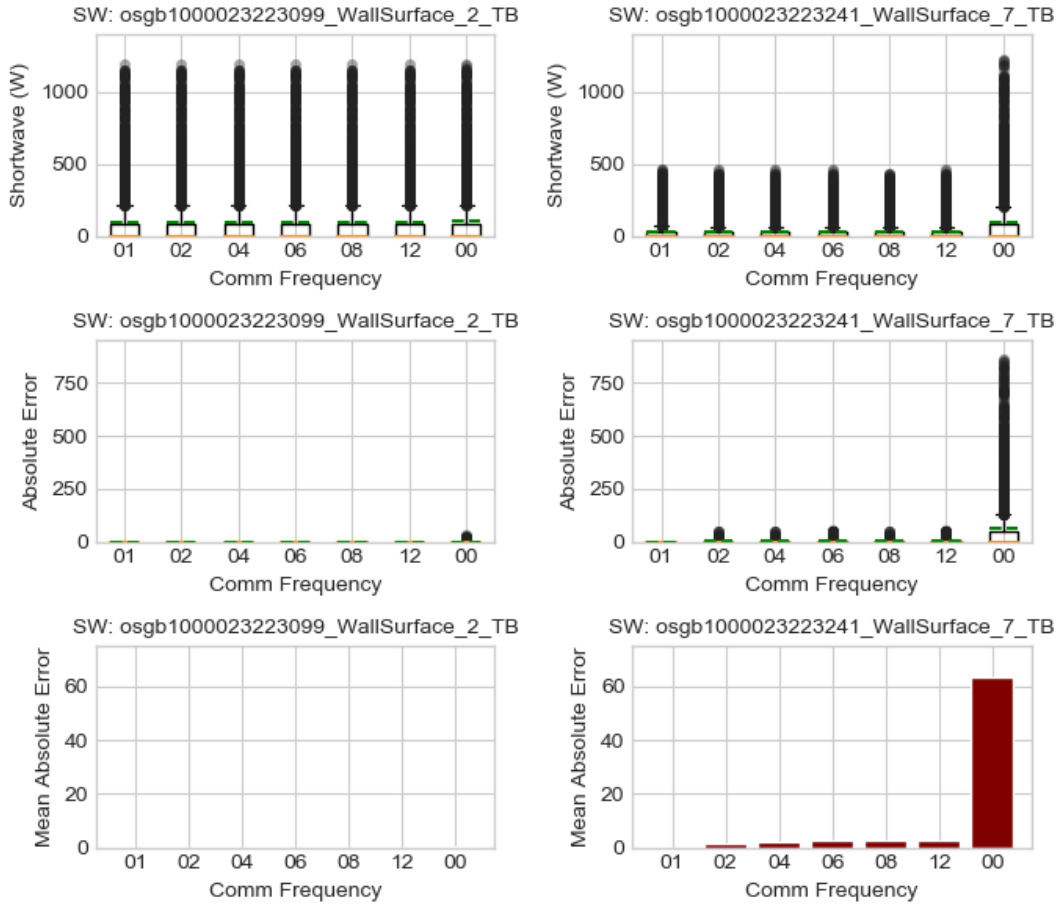

Figure 5.20: SW output errors for two surfaces: low (left) and high (right)

From Figure 5.18, the largest absolute error for DL is approximately 6500 for the edge case *00*, and 1000 for the other cases. The range for DL values is approximately 18000 in the experiments. Therefore, error of 6500 and 1000 respectively correspond to about 36% and 6% of the DL range respectively. Figure 5.21 compares the DL output errors for two surfaces with different error responses.

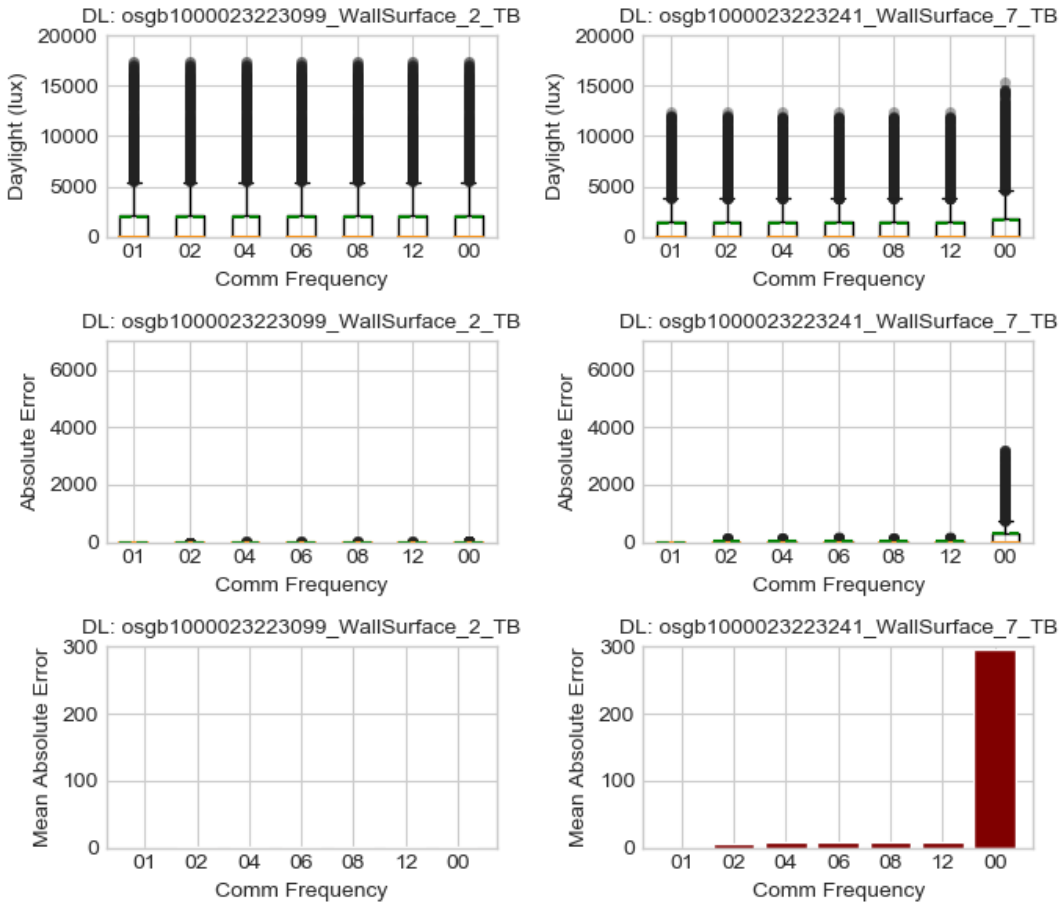

Figure 5.21: DL output errors for two surfaces: low (left) and high (right)

For the LW output, Figure 5.19 shows that the range of values is about 1200. The largest absolute error is approximately 600 for all cases. This corresponds to about 50% of the LW range. Figure 5.22 compares the LW output errors for two surfaces with different error responses.

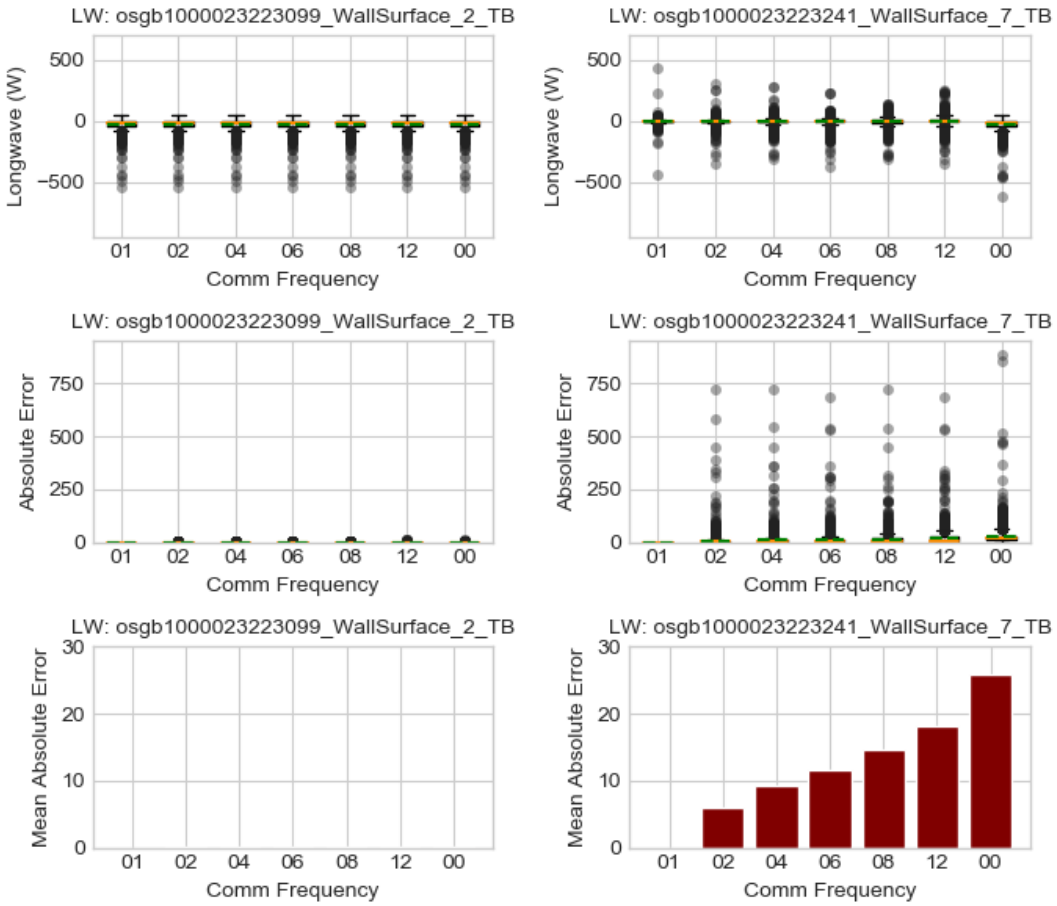

Figure 5.22: LW output errors for two surfaces: low (left) and high (right)

From Figure 5.19, the top row boxplots show that the value range for the TH output is about 14. The largest absolute error is approximately 3.3 for the edge case *00*, and about 1.8 for the other cases. Respectively, these values correspond to 24% and 13% of the TH range. Figure 5.23 compares the TH output errors for two surfaces with different error responses.

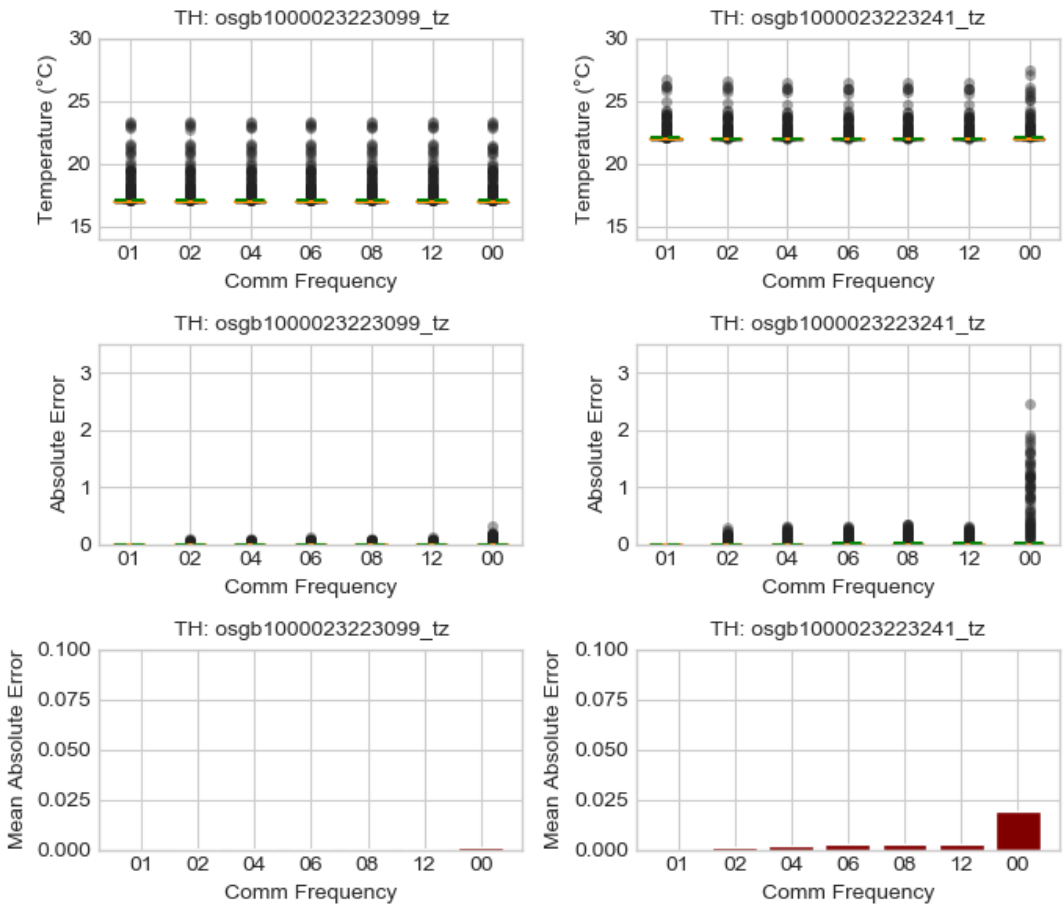

Figure 5.23: TH output errors for two surfaces: low (left) and high (right)

The experiments in this section have demonstrated that eliminating messages between CitySim-Federates can help to improve the performance of the distributed Building Energy Simulation. However, they have also shown, that communication is essential for the correctness of all outputs. In all cases, the errors produced due to reduced communication are significant for individual surfaces. However, the mean absolute errors over the entire scene are small. The results have shown that surfaces are not equally impacted by the eliminated attribute update messages. Although intuitive rationalizations have been explored, there does not appear to be a sufficient explanation to determine which of the surfaces will be highly impacted when messages are eliminated.

5.2.5 Batch Compression Experiments

In these experiments, the batching and compression strategy discussed Chapter 3 is implemented to evaluate its effectiveness on improving the communication performance of the federates in Case Study One. As this is a lossless method, the outputs produced from the distributed simulation in this case are the same as the original outputs. Therefore, there is no loss in accuracy even at the individual surface level, unlike the case of the message elimination experiments from the previous section.

The experiments make use of the reduced 51-building scene. The scene is distributed over two computing nodes using the experimental setup described for the initial experiments. As discussed in Chapter 3, this method attempts to reduce communication overheads by combining several update messages into a single batch and compressing the batched messages together to obtain a smaller number of packets for transmission over the network. During these distributed simulation experiments, the batch size is varied between 1 and 200. The batch size is the maximum number of messages which will be combined together into a single batch. The minimum batch size of 1 is the same as the normal case where individual messages are transmitted separately. As there are a total of 320 surface objects in the 51-building scene, neither CitySim-Federate in these experiments owns more than 200 surface objects. Therefore, the maximum batch size of 200 is set as a threshold for combining all updates into a single batch. Each message batch is compressed and transmitted to the RTI, which forwards it to the other federates. On arrival at the receiving federate, the packets are uncompressed to retrieve the original messages.

To aid the batching and compression method, the FOM was modified to facilitate sending attribute updates together for multiple surface objects. The class diagram in Figure 5.24 shows the modified objects. An object class has been added for each of the attributes that needs to be updated during federation execution, one each for SW, DL and TH. Each of the three new object classes has two attributes, both being lists. The first is a list of the ids of surface object instances included in the update and the second is a list of the corresponding attribute value updates for the surface objects in the list. The ids of the surface object instances are obtained from the unique integer handles assigned to the instances by the RTI when the instances are registered during federation initialization. In this modified FOM, the Surface object class no longer has

any attributes of its own. This is because federates no longer exchange updates directly for individual surface objects. However, individual surface object instances still need to be registered with the RTI in order to obtain the globally unique handles from which the ids are derived. Consequently, the Surface class is still required as these globally unique ids are depended upon for batching attribute updates together at the sending federate and unpacking the updates at the receiving federate. It should be noted that the attribute-oriented FOM in Figure 5.24 can be viewed as a generalization of the original FOM from Figure 5.5, as this new FOM permits sending and receiving attribute updates for individual surface objects. In the case where it is desired to send single attribute updates, all that is required is to create an id list with one item and a corresponding value list containing a single value. The two FOMS also differ in the options available for updating FOM object attributes. In the original FOM, an individual update can include more than one Surface attribute. For example, it is possible to deliver attribute updates for DL and SW together. However, the modified FOM only delivers updates for one attribute at a time, although it does this in bulk for multiple surfaces. Also, its object attribute updates need to include both the idList and valueList attributes in order for the update to have meaning.

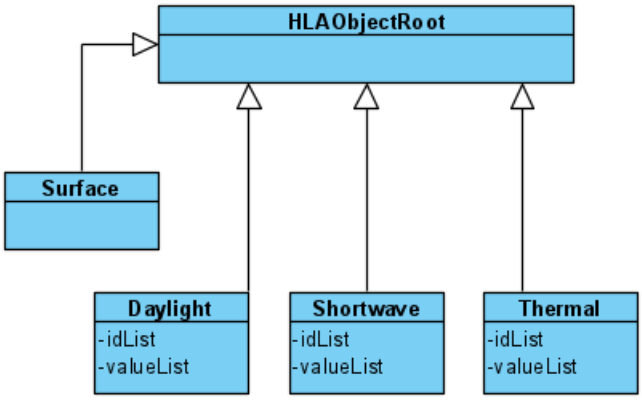

Figure 5.24: Modified class diagram with simplified HLA FOM objects for Case Study One

The graph in Figure 5.25 shows the results obtained from the batching and compression experiments, plotting a trend which illustrates how the distributed simulation wall-clock time reduces as larger numbers of messages are batched and compressed together. In these experiments, the communication wall-clock time continually reduces as the number of messages per batch is increased from 1 to 200. The largest gains occur between 1 and 100. Doubling the batch size from 100 to 200

only produces marginal gains. The largest communication time occurs for a batch size of 1, taking 184s. The minimum communication wall-clock time of 27s is achieved using a batch size of 200. This represents almost a seven-fold reduction in wall-clock time. From the previous section, the minimum achievable communication time for the reduced scene is about 20s when federates do not perform any attribute updates. The minimum of 20s is an indication of the overheads required for time synchronization between federates. From the results in this section, this implies that only 7s of additional communication overhead is added in the 200-batch experiment.

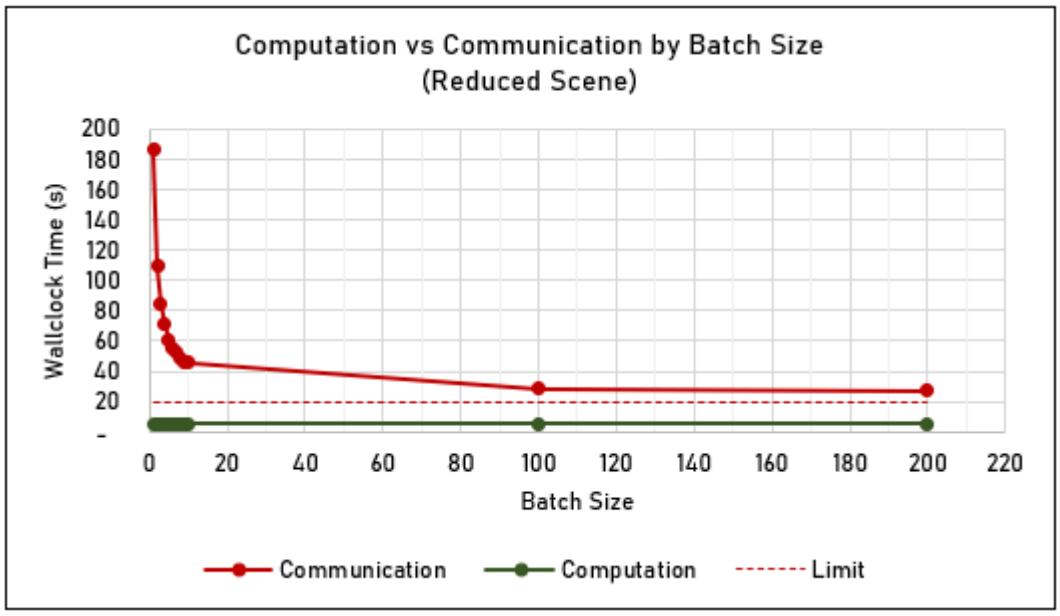

Figure 5.25: Batch compression results for reduced scene

Figure 5.26 shows the results from applying the batching and compression method to the full simple scene with 2,980 buildings. To boost the message count for these experiments, federates were configured to publish full attribute updates for all local surfaces, including updates for surfaces of no interest to other federates. However, the simulation run length was shortened to 4 months, as for the reduced scene. Two federates are employed for the batch compression experiments, varying, the batch size as 2, 10, 100 and 1000. The maximum number of surface objects owned by a single federate is about 12,000. The results show a similar pattern to those of the experiments for the reduced 51-building scene. Performance gains seem to diminish after a batch size of 100. However, the communication time reduction is significant from the 2-batch case to the 100-batch, falling from 2569s to 357s, which is a 7-fold improvement.

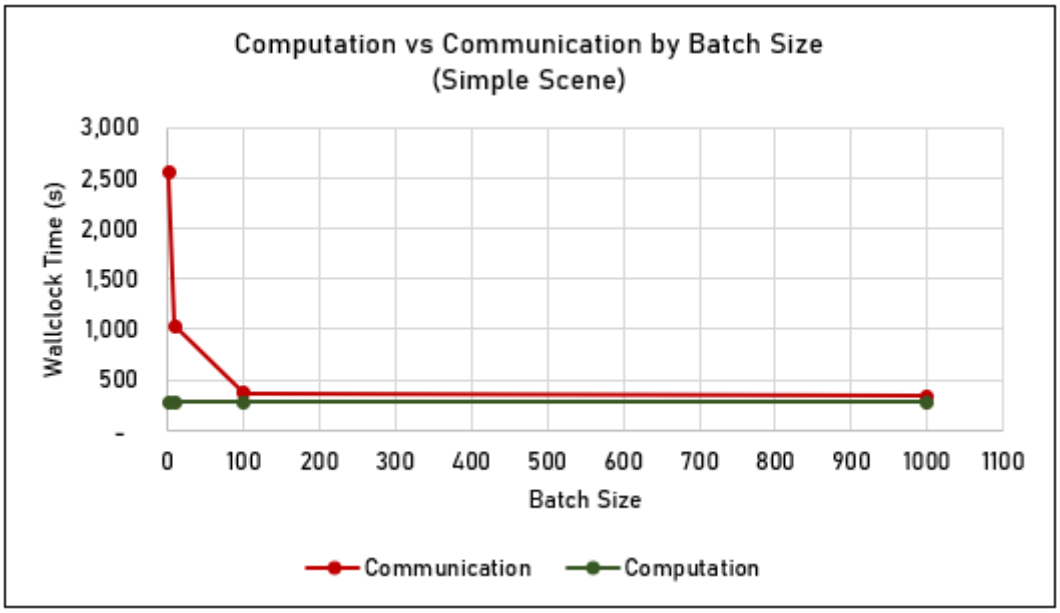

Figure 5.26: Batch compression results for simple scene

The experiments in this section have demonstrated how the batching and compression strategy can improve the performance of distributed Building Energy Simulations. As discussed in Chapter 3, the potential drawbacks of this strategy include additional time for compressing and de-compressing messages. In this case, however, it appears that these considerations are outweighed by the advantage of reduced communication overheads. The experiments have also shown that gains made can become marginal after a point, as the batch size increases. An important advantage of this strategy is that significant performance gains are made without having to sacrifice output accuracy.

5.3 Case Study Two: Heterogeneous Distributed Simulation

Case Study Two is a heterogeneous simulation model that couples the Building Energy Simulation from Case Study One, CitySim, with a Building Occupancy Simulation, No-MASS. A Building Energy Simulation can interoperate meaningfully with a Building Occupancy Simulation to obtain better energy use predictions fine-tuned by the activities of the building's occupants.

Case Study Two serves to present the additional concerns involved when the individual simulators involved in the distributed simulation are of different types. This section explores the impact on communication when maintaining interoperability in a heterogeneous arrangement. Communication patterns in the heterogeneous case can differ from patterns in the homogeneous case. For example, the points where communication occurs are the same for all federates in the homogeneous distributed simulation of Case Study One. However, the communication points in a heterogeneous distributed simulation may differ depending on the type of federate. This depends on the synchronization requirements for different federates and can possibly lead to a need for tighter coupling. The more types of federates that are added to the heterogeneous simulation, the greater the potential complexity regarding communication. Figure 5.27 illustrates a possible pattern of communication between four HLA federates, two federates of type A and two federates of type B. In this case, the communication between A and B is more loosely coupled than the communication between A federates or the communication between B federates.

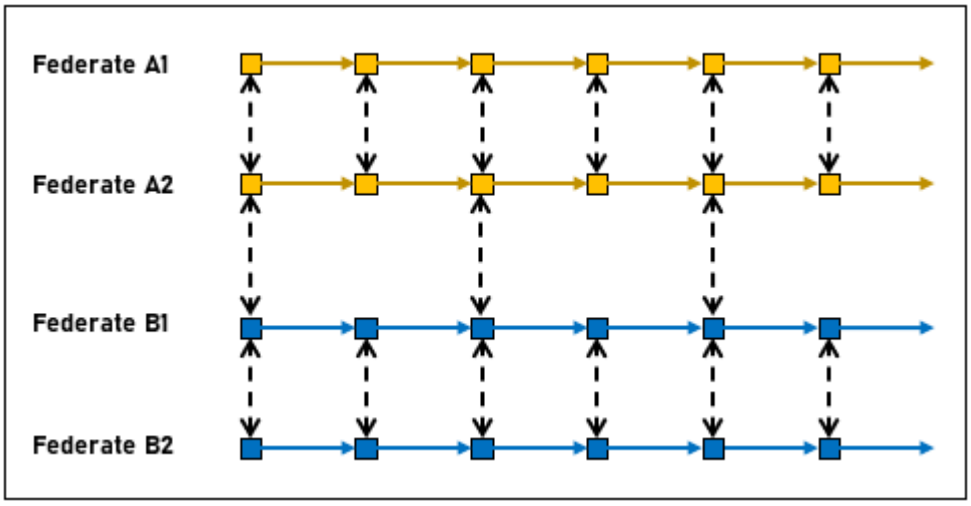

Figure 5.27: Example communication pattern between four federates of two types

5.3.1 Nottingham Multi-Agent Stochastic Simulation

The Nottingham Multi-Agent Stochastic Simulation (No-MASS) (Chapman, Siebers and Robinson, 2018) is a Building Occupancy Simulation designed to integrate with the Building Energy Simulation tool EnergyPlus (Crawley *et al.*, 2001). No-MASS couples with EnergyPlus using the FMI standard (Blochwitz *et al.*, 2011). Although EnergyPlus is a single building simulation tool, No-MASS can also be employed to simulate a multi-building scene. This is important as the experiments in this chapter all involve scenes with multiple buildings. No-MASS simulates each building separately and does not account for any interactions between occupants of different buildings. Consequently, no communication needs to occur between different No-MASS-Federates. Each No-MASS-Federate only needs to exchange data with the CitySim-Federates that simulate the same buildings. CitySim includes a simple model of stochastic occupant presence which uses stochastic parameters to represent factors such as the number of occupants present in a building, how occupants interact with windows and lighting. In a meaningful coupling with No-MASS, the stochastic occupant model can be replaced with parameters supplied by No-MASS. This is ideal as No-MASS is a specialized tool for Building Occupancy Simulation and offers better flexibility for modelling occupant behaviour using the ABS paradigm.

For the experiments in this section, the HLA federation from Case Study One will be extended to include No-MASS. Figure 5.28 shows a conceptual illustration of the CitySim / No-MASS HLA Federation. Each federate is run on a separate computing node. Each CitySim-Federate will share a scene partition with a partner No-MASS-Federate. Each CitySim-Federate / No-MASS-Federate pair will communicate to exchange relevant updates. As in Case Study One, City-Sim-Federates continue to exchange attribute updates with one another via the RTI. No-MASS-Federates, however, do not need to communicate with one another. Figure 5.28 illustrates this arrangement, pairing each No-MASS-Federate with a corresponding CitySim-Federate. While the CitySim-Federate simulates building energy interactions for the assigned partition, its partner No-MASS-Federate simulates occupant interactions for buildings in the same partition. Communication occurs at two levels, hourly timesteps and sub-hourly timesteps. CitySim-Federates exchange data as in Case Study One, with sub-hourly attribute updates after each computation. At hourly timesteps, however, CitySim-Federates and No-MASS-Federates exchange attribute

updates such as thermal zone (room) temperatures and occupant presence. It must be noted that No-MASS can also perform sub-hourly simulation computations. However, this level of resolution is not required for data exchange with CitySim. Sub-hourly timesteps in No-MASS-Federates do not need to be regulated by the RTI with TAGs as no data exchange is involved. Therefore, No-MASS-Federates can safely and independently manage their own internal sub-hourly timesteps.

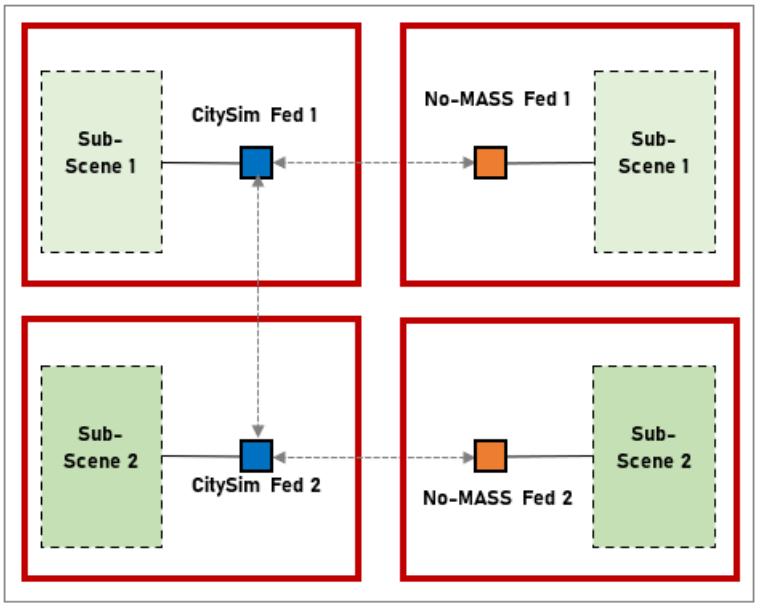

Figure 5.28: Conceptual illustration of interactions in the CitySim / No-MASS Federation

Synchronization in the heterogeneous simulation is complicated by the fact that No-MASS-Federates must ignore the sub-hourly CitySim attribute updates. These are regulated by the RTI due to the attribute updates involved. In the CitySim / No-MASS Federation implementation, the general hourly timestep is tracked by an integer, X. CitySim sub-timesteps are given incrementing fractional values such as X.1, X.2, X.3. City-Sim Federates advance their simulation time in these steps. However, No-MASS-Federates advance at an integral timestep, at the point where data exchange occurs between all the types of federates.

In summary, timestep management ensures that:

- CitySim-Federates continue to synchronize with one another at sub-hourly timesteps to exchange attribute updates after computations.
- During CitySim sub-timesteps, no data is exchanged between CitySim-Federates and No-MASS-Federates.

- No-MASS-Federates are allowed to execute sub-hourly timesteps which are not tracked by the RTI.
- At each hourly timestep, No-MASS-Federates and CitySim-Federates all synchronize with one another.

5.3.2 Case Study Two: HLA Federation

The class diagram in Figure 5.29 provides an overview of the HLA object classes in the FOM of the federation for Case Study Two. This case includes all the HLA Surface object class attributes from Case Study One, as this data is still exchanged between CitySim-Federates during federation execution. A new Zone object class has been added to the FOM in order to exchange building zone data between CitySim-Federates and No-MASS-Federates. Similar to surface objects, zone objects are persistent throughout federation execution. Therefore, they are represented as HLA object classes instead of interaction classes. The Zone object class includes four attributes: *Temperature*, *Illuminance*, *NumberOfOccupants* and *LightState*. Different federate types own different attributes, which differs from the federation in Case Study One. *Temperature* and *Illuminance* are owned by CitySim-Federates, while *NumberOfOccupants* and *LightState* are owned by No-MASS-Federates. In this arrangement, CitySim-Federates subscribe to *NumberOfOccupants* and *LightState* but cannot publish them. It is the responsibility of the No-MASS-Federates to publish the required attributed updates concerning those attributes. On the other hand, No-MASS-Federates subscribe to *Temperature* and *Illuminance* but cannot publish them, since those attributes are the responsibility of CitySim-Federates.

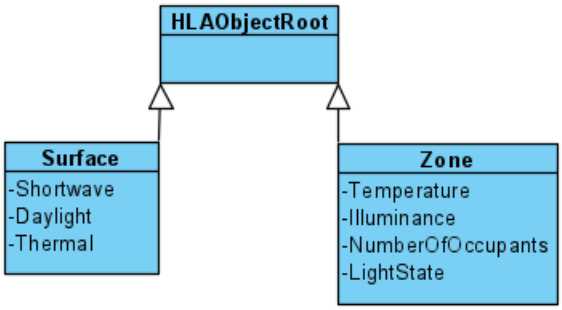

Figure 5.29: Class diagram showing simplified HLA FOM objects for Case Study Two

While Figure 5.29 provides a simplified view of the FOM, the actual FOM used in the distributed simulation experiments for Case Study Two is structured as shown in Figure 5.30. This structure is similar to the FOM that was created in Case Study One to enable attribute updates to be transmitted in batches. The attributes from the simplified FOM in Figure 5.29 are converted to HLA objects composed of pairs of id and value lists. As observed for the expanded FOM in Case Study One, sending independent attribute updates objects can be achieved by placing a single element in the id and value lists. The Surface and Zone classes are retained in the expanded FOM, similar to practice in Case Study One. This is important as individual surface and zone objects need to be registered with the RTI in order to supply the globally unique ids that will be used in the id lists.

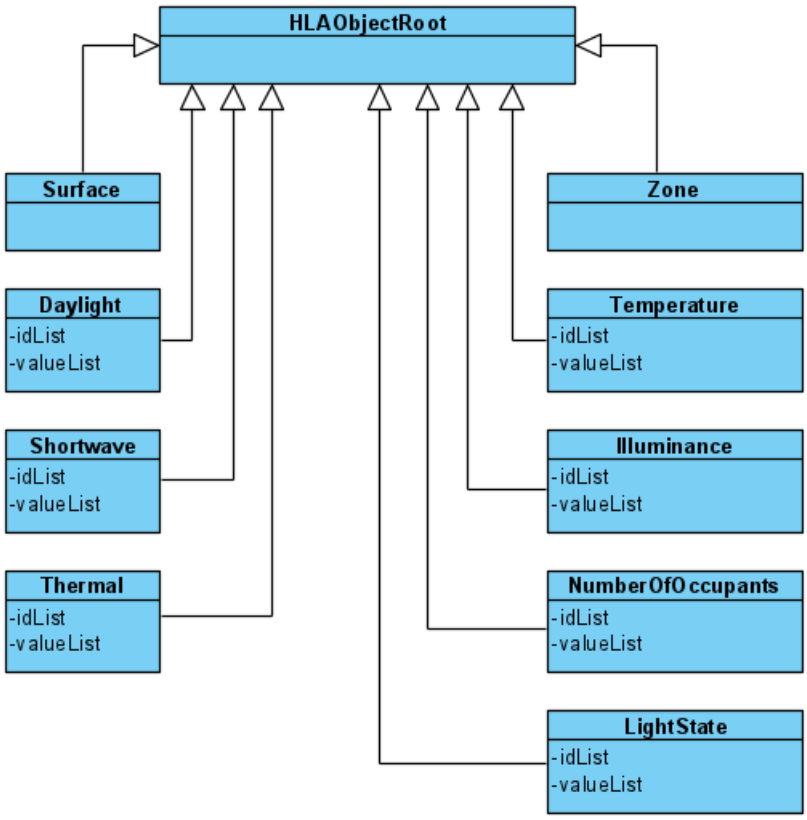

Figure 5.30: Class diagram showing expanded HLA FOM objects for Case Study Two

As discussed in this section, the data dependencies between CitySim-Federates and No-MASS-Federates have been identified and captured in the FOM. The purpose of the experiments conducted on this case study is to measure communication time. For

this purpose, it is adequate to implement attribute update exchanges and time synchronization between CitySim-Federates and No-MASS-Federates. As this is sufficient for enabling the measurement of communication time, it is not necessary to make further modifications to the internal workings of CitySim and No-MASS to utilise the data received for internal computations.

The sequence diagram in Figure 5.31 illustrates the process of message exchange between the RTI, CitySim-Federate and No-MASS-Federates during federation execution. Similar to CitySim-Federates, a No-MASS Federates is a unit composed of two parts, a No-MASS-Federate Ambassador and a No-MASS-Simulator. While the No-MASS-Federate Ambassador performs all message exchanges with the RTI, the No-MASS-Simulator performs the actual simulation. Similar to Case Study One, No-MASS-Federate Ambassadors work with the RTI to create, initialize and destroy the federation at the start and end of execution, and perform the signalling required for synchronization between federates. While the sequence diagram in Figure 5.31 only shows one Federate of each type, a CitySim / No-MASS Federation can include multiple CitySim-Federates and No-MASS-Federates, all working according to the sequence described by the diagram. Creating the federation is done only once, by the first federate to join, and destroying the federation is the responsibility of the last federate to resign.

Similar to the federation in Case Study One, time synchronization involves federates sending TARs to the RTI and waiting to receive TAGs before advancing to the next timestep. However, as discussed in the previous section, there are multiple levels of synchronization to consider for Case Study Two. On one level, hourly timestep synchronization is used between CitySim-Federates and No-MASS-Federates. On another level, CitySim-Federates also synchronize with one another at sub-hourly timesteps. No-MASS-Federates do not participate in the sub-hourly attribute updates but independently manage their own internal sub-hourly timesteps as these do not require attribute updates. From this, it is evident that heterogeneous coupling between No-MASS-Federates and CitySim-Federates is looser than homogeneous coupling between CitySim-Federates. Also, there is no coupling between No-MASS-Federates as they function independently of one another. The volume of heterogeneous communication between No-MASS-Federates and City-Sim-Federates is also lower than that between CitySim-Federates. This is due to the fact

that attribute update messages are exchanged at the higher zonal level rather than at the lower surface level. By definition, a zone is composed of multiple surfaces. This implies a smaller communication load between No-MASS-Federates and CitySim-Federates.

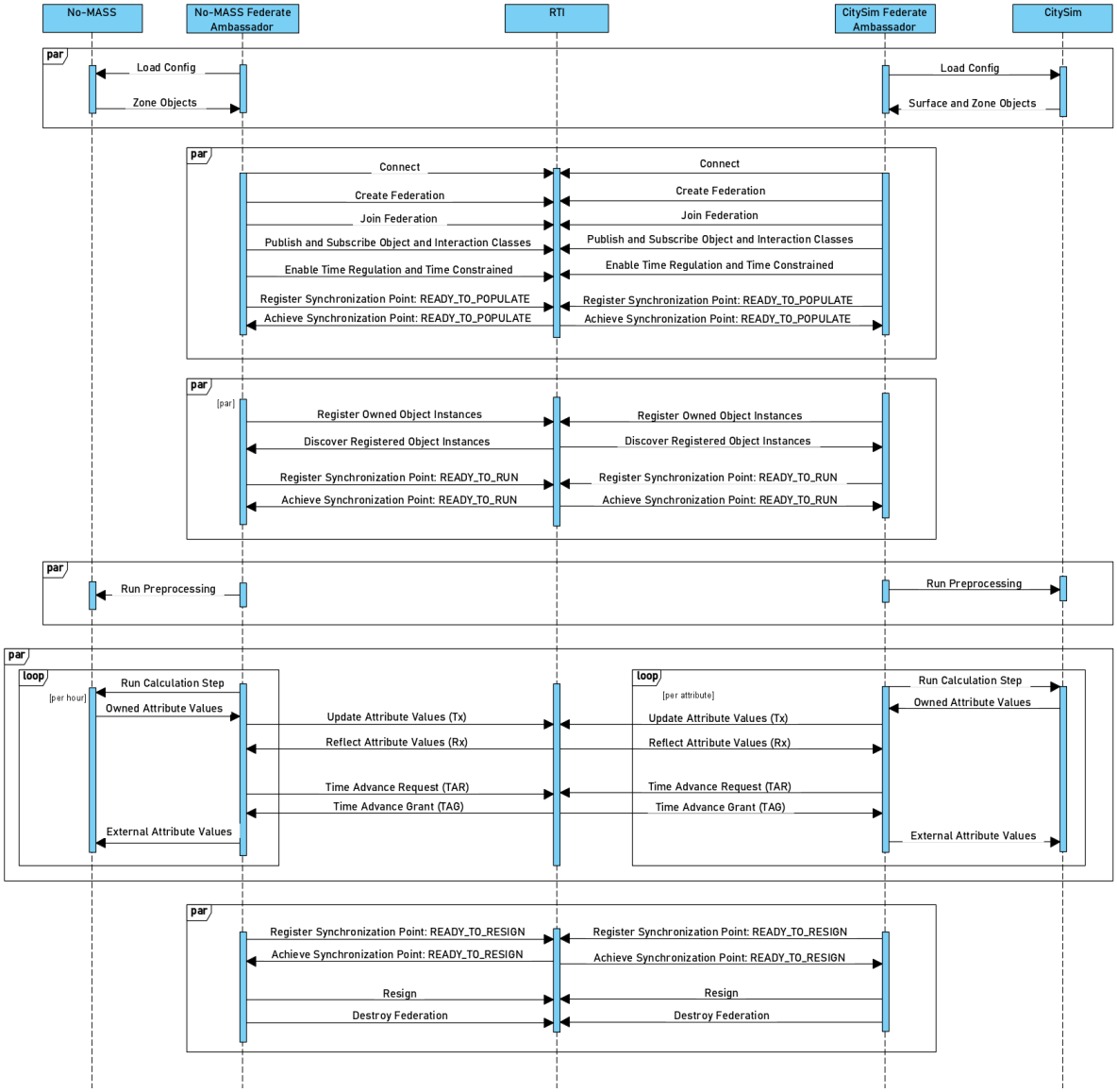

Figure 5.31: Sequence diagram of HLA federation execution for Case Study Two

5.3.3 Initial Experiments

The experiments in this section use the reduced 51-building scene distributed over two computing nodes, utilising the same experimental setup described for Case Study One. One computing node hosts a CitySim-Federate while the other hosts a No-MASS-Federate. Figure 5.32 shows the results from the initial experiments. Although they both simulate the same building scene, the computation time of the No-MASS-Federate significantly exceeds the computation time of the CitySim-Federate. While the CitySim simulation workload requires about 5s to complete, the No-MASS simulation workload requires about 15s to complete, which is three times longer. This is due to the fact that the heterogeneous simulators perform different computations even though they share the same scenario.

As discussed in previous sections, communication time is measured as the total time spent in activities other than processing the simulation workloads, including:

1. Actively transmitting attribute updates.
2. Passively idling.
3. Actively receiving attribute updates.
4. Actively sending and receiving TARs, TAGs, and other signalling messages.

As there is a three-fold difference between the computation times, a significant portion of measured communication time for the CitySim-Federate is due to passive idling while waiting for the No-MASS-Federate to complete its workload. From Figure 5.32, the measured communication clock time for the No-MASS-Federate is about 60s, while the communication time for the CitySim-Federate is about 70s. The 10s difference corresponds to the difference between their computation times, which indicates that the CitySim-Federate spends about 10s idling and 60s in active communication.

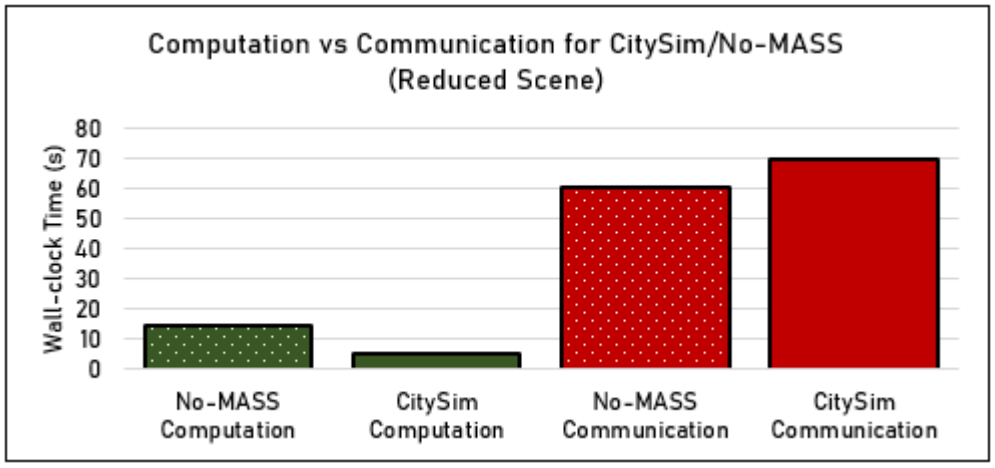

Figure 5.32: Initial experiment for CitySim /No-MASS Federation

The initial results raise concerns about computation time imbalances between heterogeneous simulators in distributed simulations. Large imbalances need to be addressed as they can have an impact on performance, judging from the motivating example discussions in Chapter 4. One possibility for reducing the workload disparity between heterogeneous simulator types would be to perform further partitioning. Following this approach, partitions can be sub-divided to reduce computation times for the slower simulator. This, however, will require additional computing nodes to handle each sub-partition. Figure 5.33 illustrates this arrangement using the CitySim/No-MASS Federation. As there are no communication paths between No-MASS-Federates, sub-partitioning in this case does not need to consider interactions between sub-partitions. This may not necessarily be the case for other heterogeneous simulations, and sub-partitioning may introduce additional communication.

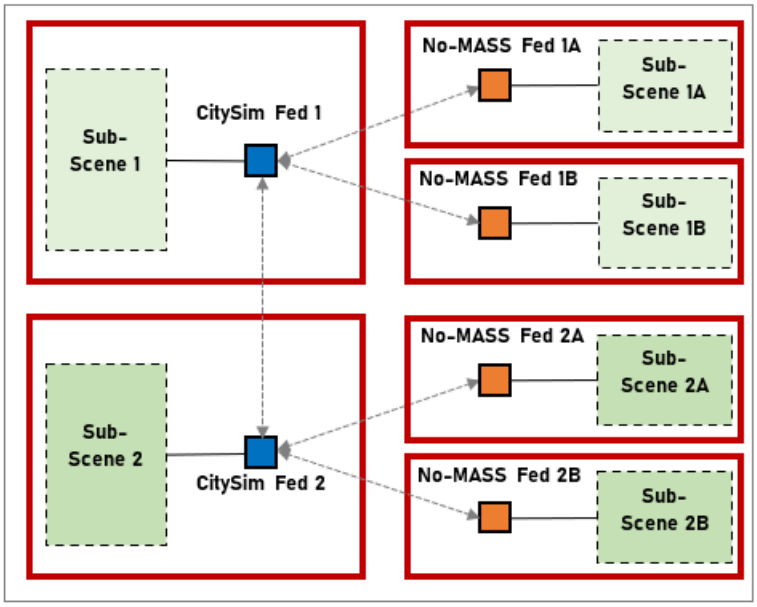

Figure 5.33: Sub-partitions for CitySim/No-MASS Federation

Another approach for reducing workload disparity would be to collocate heterogeneous simulators on the same computing node as shown in Figure 5.34. This arrangement is similar to a homogeneous distributed simulation, as each computing node can be considered as a composite of the same two simulators. In this case, initial partitioning is sufficient to balance the workload and sub-partitioning is not necessary.

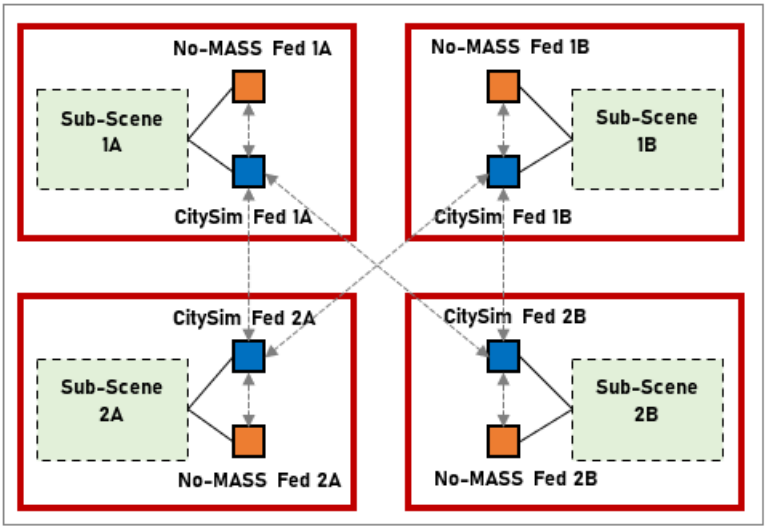

Figure 5.34: Collocated CitySim/No-MASS Federation

5.3.4 Batch Compression Experiments

The lossless batching and compression strategy is applied to the heterogeneous distributed simulation for Case Study Two, and the resulting communication time is measured. The batch size is set to the maximum of 200, which produced the largest communication time reduction for Case Study One. The chart in Figure 5.35 shows the results obtained from experiments. The communication wall-clock time of the No-MASS-Federate is 17s. As discussed in the previous section, the communication time measurement for the No-MASS-Federate does not include idle time. The new communication time is about 28% of the original 60s from the base case. Similar to Case Study One, the gains made by reduction in communication overheads outweigh the additional time required to compressing and decompress messages.

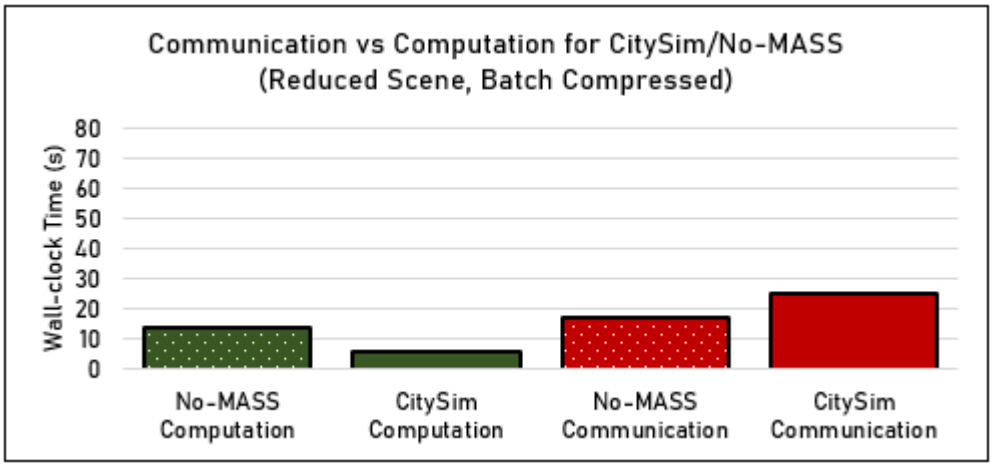

Figure 5.35: Batching compression experiment for CitySim/No-MASS Federation

5.4 Summary

In this chapter, the experiments for Case Study One have shown how communication time can be reduced by applying various communication strategies. They have demonstrated that for a lossy method, such as message elimination, performance gains are made at the expense of simulation accuracy. The message elimination experiments for CitySim have shown that although the average errors measured at system level may be negligible, the errors on individual objects can be significant. They have also shown that objects may not all be affected equally by the loss of messages. While this is the case, it may not be straightforward to establish a sufficient explanation for the error pattern. For the Case Study One experiments, the

most intuitive hypothesis failed to explain the pattern. The degree of error a surface generates appears to be unrelated to the number of external interactions it possesses.

The experiments show a significant reduction in communication time for the batching and compression strategy. This performance gain is obtained at no cost to simulation accuracy, as the strategy preserves all messages. In all cases, the additional overheads introduced by this strategy were outweighed by the gains made in communication time.

The experiments in this chapter have also examined time synchronization costs separately from the general communication costs. This is investigated by a special experiment in which no messages are exchanged except those required for time advancement. The measured synchronization time serves to establish a bound for the performance gains that can be achieved in the distributed simulation.

The experiments for Case Study Two have served to bring out the additional concerns that must be taken into account by the framework when different types of simulators are involved in a distributed simulation. These experiments have demonstrated that compared to homogeneous distributed simulations, more complex communication patterns can be introduced when heterogeneous simulators are coupled. They have also shown that load balancing can potentially be more complex for the heterogeneous case than the homogeneous case. For the heterogeneous experiments, the batching and compression strategy produced a similar performance improvement as in Case Study One.

6 FRAMEWORK

6.1 Overview

The framework proposed in this chapter forms the basis for implementation of meta-simulations designed to evaluate the performance of distributed simulations. The framework accounts for important performance-influencing factors related to communication. The difference in performance between alternative communication strategies is also an important consideration. The contents of the framework are informed by the two case studies presented in Chapter 5. As discussed in Chapter 3, the framework adopts the MAS paradigm.

6.2 Conceptual Model

The framework focuses on modelling the essential aspects of the federation that have significant bearing on communication time. It does not attempt to model the components of the HLA federation in great detail, such as all the services available on the interface between federates and the RTI. The following sections describe the essential elements of the federation execution that have been included in the framework, following the MAS paradigm.

6.2.1 Framework Components

- Coordinator Agent:

  The coordinator agent represents the component of the distributed simulation that coordinates execution. For an HLA federation, this is represented by the RTI. In a meta-simulation based on an HLA federation, this agent mimics the sub-set of RTI functionality that is required for executing the distributed simulation. For the case studies in Chapter 5, the Coordinator Agent can be labelled *RTI Agent*. The responsibilities of the RTI Agent for the two case studies include:

  a. Routing messages between federates based on the publish-subscribe paradigm.

b. Controlling federate time advancement during federation execution, ensuring all federates have received the required messages in each timestep before allowing Time Advance Grants (TAGs).

For distributed simulations based on the HLA, an implementation of the framework will normally include a single RTI Agent, as is usually the case with HLA federations. However, it is also possible to include more than one RTI Agent to manage separate federations that can interact with one another.

- Node Agent:

  Node Agents represent the components of the distributed simulation that perform the computations required to drive the simulation forward. A Node Agent can generate messages to share its local state with other Node Agents. For the case studies in Chapter 5, the Node Agents can be labelled *Federate Agents*. Federate Agents simulate the functions performed by federates in the case studies. The responsibilities of the Federate Agents include:

  a. Performing computations
  b. Publishing updates to the RTI Agent
  c. Receiving updates from the RTI Agent
  d. Sending Time Advance Requests to the RTI Agent and waiting to receive a TAG before proceeding to the next time-step.

  For distributed simulations based on the HLA, an implementation of the framework will normally include multiple Federate Agents that only interact directly with the RTI. Federate Agents can have properties to reflect the conditions of a real system. For example, a Federate Agent parameter can be added for the computation time per timestep. This can be set up as a stochastic parameter with values drawn from a triangular distribution with a min, max and mode. For homogeneous distributed simulations with imbalanced workloads, different min, max and mode values can be set for separate Federate Agents. Also, for heterogeneous distributed simulations, distinct types of Federate Agents can have different triangular distributions to reflect different computation times.

A parameter can also be added to the Federate Agent to represent the number of timesteps between consecutive updates. For a heterogeneous distributed simulation, this parameter that can be set differently according to the type of Federate Agent. This can account for the different communication patterns of distinct types of Federate Agents. This parameter is also useful for testing communication strategies which employ message elimination for some timesteps, by increasing the period between consecutive updates.

6.2.2 Experimental Factors

The experimental factors in the framework include various parameters that govern the distributed simulation execution and influence its performance. This includes global parameters that affect all agents, such as network latency and bit rate. It also includes parameters that are specific to Coordinator Agents and Node Agents. Table 6.1 provides a list of experimental factors considered in the framework. The list of experimental factors is informed by the case study experiments conducted in Chapter 5.

Table 6.1: Framework Parameters

Level / Component	Parameter	Description	Units
Global	Network Latency	One way latency associated with point-to-point packet transfer over the network.	ms
	Network Bit Rate	Data transfer rate over the network.	bits/s
	Payload Bytes	Size of data payload carried in each network packet.	bytes
	Header Bytes	Size of metadata appended to network packets.	bytes
	Max Timestep	Total simulation run length.	timesteps

Level / Component	Parameter	Description	Units
	Number of Nodes	Count of computing nodes co-operating in the distributed simulation.	number
Node Agent	Initialization Time	Computation time required for pre-processing on each node before starting the simulation execution. This parameter can be drawn from a specified triangular distribution with min, mode, and max values.	ms
	Computation Time per Timestep	Different values can be set for each node to reflect scenarios with imbalanced computation workloads. This parameter can be drawn from a specified triangular distribution with min, mode, and max values.	ms
	Message Bytes	Size of individual messages transmitted from the node. This parameter can be drawn from a specified triangular distribution with min, mode, and max values.	bytes
	Message Volume per Timestep	Total number of individual messages transmitted from the node in each timestep. Different values can be set for each node to reflect variations in communication requirements.	number
	Batch Size	When applying the batching and compression strategy, this parameter indicates the number of messages to combine in each batch. A setting of 1 means batching is not applied, and each message is transmitted separately.	number

Level / Component	Parameter	Description	Units
	Compression Ratio	Applies to the batching and compression strategy. A setting of 1 means no compression is applied.	number
	Update Period	Specifies the number of timesteps between subsequent message transmissions. This can be used to emulate the message elimination strategy.	timesteps
	Fixed Transmission Cost	A fixed delay due to operations required to establish the communication link e.g., handshaking.	ms
	Variable Transmission Cost	Variable delay due to operations required to process packets for transmission.	ms
Coordinator Agent	Initialization Time	Time required to set up the distributed simulation session, including signalling between nodes and the coordinator. This parameter can be drawn from a specified triangular distribution with min, mode, and max values.	ms

6.2.3 Responses

The measured responses of the framework track the performance of various aspects of the meta-simulation. The main response to be measured is the total execution time. This depends on the total communication time and total computation time for each Node Agents. As established in Table 6.1, the total computation time for each Node Agent depends on the specified input parameter setting for *Computation Time* per timestep. However, the total communication time is not specified directly by an input parameter. Instead, it is determined indirectly by executing the meta-simulation, and

depends on the settings of several input parameters including *Message Bytes*, *Message Volume*, *Batch Size,* and *Network Latency*, among others. Similar to the approach used for the case study experiments in Chapter 5, the total communication time for each Node Agent is determined as time not spent performing computation. This is measured as a sum of three parts:

(1) Active Transmit Time
(2) Active Receive Time
(3) Passive Idling Time + Synchronization Time.

Table 6.2 provides an overview of the responses measured by the framework.

Table 6.2: Framework Responses

Level / Component	Output	Description	Units
Node Agent	Total Communication Time	The total time spent on each node to perform operations other than computation of its simulation workload. This includes time for packet transmission and receipt as well as for idle time.	ms
	Total Transmit Time	The total time spent actively transmitting messages by each node.	ms
	Total Receive Time	The total time spent actively receiving messages by each node.	ms
	Total Idle Time	Time spent by each node while neither performing computation nor active communication operations.	ms
	Total Packets Transmitted	The total number of data packets transmitted by each node during distributed simulation execution. This may differ from the number of messages depending on factors such as *Payload Bytes*, *Message Bytes*, *Batch Size*, and *Compression Ratio*.	number

Level / Component	Output	Description	Units
	Total Packets Received	The total number of data packets received by each node during distributed simulation execution.	number
	Total Computation Time	The total computation time for each node. This depends directly on the input parameter for *Computation Time* per timestep.	ms
Global	Total Execution Time	The total time elapsed from the start of the distributed simulation to completion.	ms

6.3 Model Content

6.3.1 Node Agent

Node Agents are modelled using state charts with transition rules that are governed by the node parameters set out in Table 6.1. As illustrated by Figure 6.1, Node Agent state charts include the following states:

- Initialize:

  In this state, Node Agents perform the required signalling with the Coordinator Agent to set up the distributed simulation. Node Agents also perform pre-processing computations that are required before starting the simulation timesteps. For an HLA distributed simulation, this state represents the time taken by Federate Agents to join the federation, register objects to establish ownership, declare class attributes they intend to publish, and specify class attributes to which they wish to subscribe. From Figure 6.1, the *Initialize* state transitions to the *Compute* state. This transition is triggered by a timeout which is determined by the Node Agent parameter settings for *Initialization Time*.

- Compute:

  In this state, Node Agents perform the simulation computations for one timestep. From Figure 6.1, the *Compute* state transitions to the *Send* state. This transition is triggered by a timeout determined by the Node Agent parameter settings for *Computation Time*. For scenarios where the simulation workload is highly imbalanced, the *Computation Time* parameter settings will vary significantly from one Node Agent to another. Sequential workloads can be implicitly accounted for in the parameter settings for different Node Agents. When comparing what-if scenarios for different numbers of Node Agents, the *Computation Time* parameter will need to be adjusted accordingly.

- Send:

  In the *Send* state, Node Agents transmit data packets. For an HLA distributed simulation, the Federate Agents will transmit data packets to the Coordinator Agent. The number of data packets sent by each Node Agent is determined parameter settings such as *Message Size*, *Number of Messages*, *Batch Size*, and *Compression Ratio*. The time required to send data packets is determined by global parameter settings for network *Latency*, *Bit Rate*, and *Payload Size*. From Figure 6.1, the *Send* state transitions to the *Receive* state. This transition is triggered by a timeout. The timeout is calculated on each Node Agent as the time required to send all of its data packets.

- Receive:

  In the *Receive* state, Node Agents wait to receive data packets. Node Agents remain in this state until they receive a TAG from the Coordinator Agent. On first entering the *Receive* state, a Node Agent sends a TAR message to the Coordinator Agent to indicate that it is ready to proceed to the next timestep. The Coordinator Agent will not respond with a TAG message until it is safe to move forward. While in the *Receive* state, each Node Agent keeps track of the time spent actively receiving data packets and the time spent idling.

  For an HLA distributed simulation, Federate Agents receive data packets which the RTI Agent forwards from other Federate Agents. The active receive time depends on the number of data packets received from the RTI

Agent in this state. The remaining time is recorded as idling time. The transition out of the *Receive* state is triggered when a TAG message is received from the RTI Agent. The RTI Agent only sends a TAG when the Federate Agent has acquired all relevant messages in the current timestep and can therefore proceed safely forward to the next timestep.

From Figure 6.1, the *Receive* state can either transition back to the *Compute* state, or transition forward to the *Finish* state. In both cases, the transition out of the Receive state is triggered when the Node Agent receives a TAG message from the Coordinator Agent. If the global setting for the *Max Timestep* parameter has not yet been reached, the Node Agent transitions back to the *Compute* state. If the final timestep has been completed, however, the Node agent transitions forward to the *Finish* state.

- Finish:

  In the *Finish* state, Node Agents perform post-processing operations and carry out the signalling necessary for ending the distributed simulation session. For an HLA distributed simulation, Federate Agents resign from the federation execution and signal the RTI Agent to destroy the federation if no other Federate Agents are still joined.

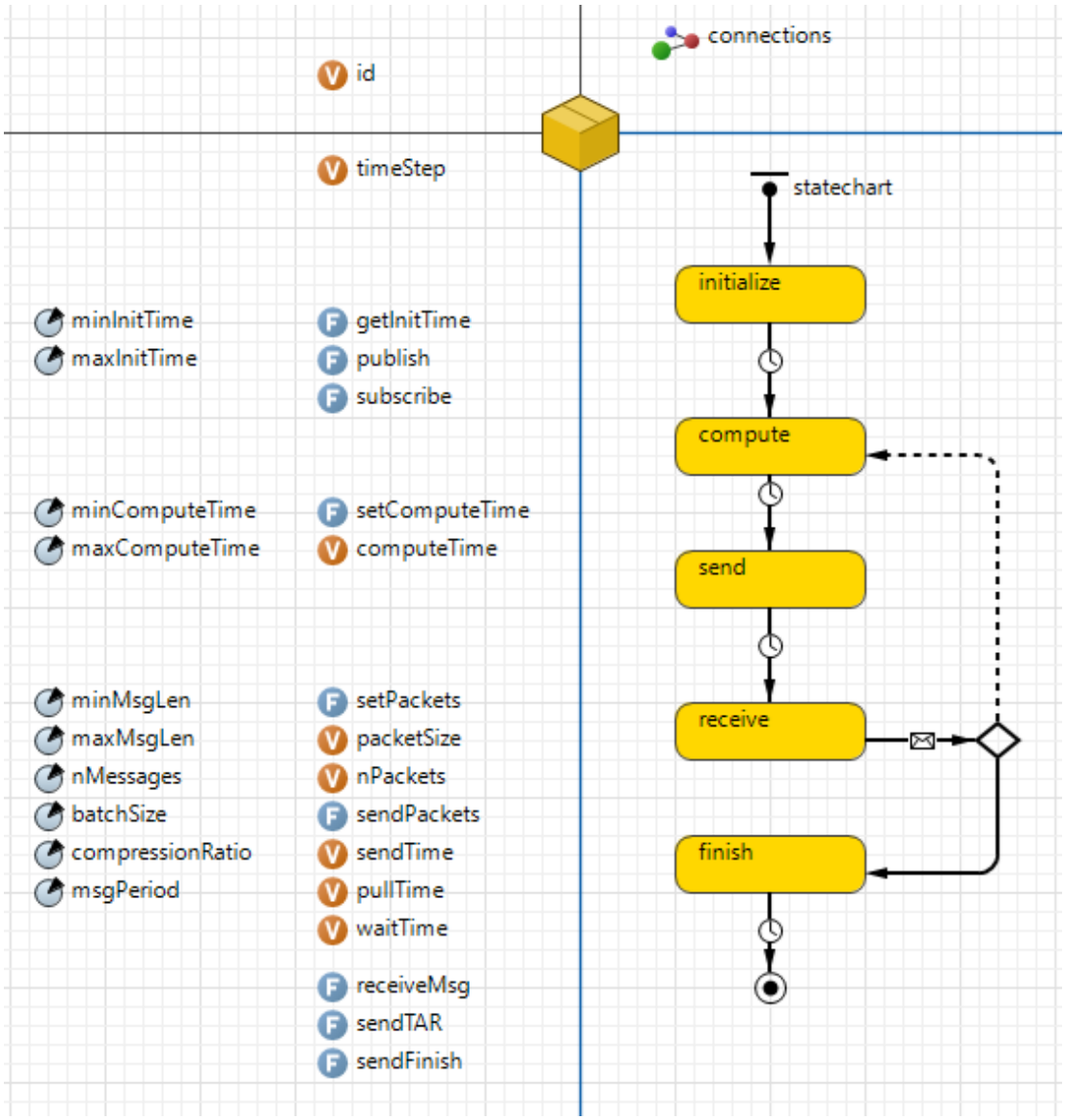

Figure 6.1: Node Agent State Chart

6.3.2 Coordinator Agent

Similar to the Node Agent, the Coordinator Agent also has an internal state-chart that determines its behaviour. As illustrated by Figure 6.2, the Coordinator Agent state-chart includes the following states:

- Initialize:

  Similar to the *Initialize* state in the Node Agent, this state represents the period of setting up the distributed simulation. For an HLA distributed simulation, Federate Agents exchange messages with the Coordinator Agents

to create the federation, register objects, and declare publication and subscription intentions. From Figure 6.2, the *Initialize* state transitions to the *Wait* state. This transition is determined by a timeout which depends on the *Initialization Time* parameter setting.

- Wait:

  In the *Wait* state, the Coordinator Agent waits to receive messages from the Node Agents. For an HLA distributed simulation, these messages can include data packets and signalling messages. Examples of signalling messages an RTI Agent may receive include TARs and resignation notifications from Federate Agents. To manage different types of messages, a Coordinator Agent can keep separate queues for each type. For example, an RTI Agent can have one queue for data packets that need forwarding, a second queue for TARs, and a third queue for resignation notifications. When the RTI Agent receives a message, it places it in the appropriate queue. Each data packet can be kept in the queue until it has been forwarded to all eligible subscribers, after which it can be removed from the queue. TARs can be retained in their own queue until all Federate Agents have sent in their own TARs. If the data packet queue is empty TARs have been received from all Federate Agents, the RTI Agent can safely grant TAGs to the Federate Agents.

  From Figure 6.2, the *Wait* state can either transitions to the *Process* state or the final state. The transition to the *Process* state is triggered whenever the Coordinator Agent receives any message. It is also triggered by a periodic timeout in order for the Coordinator Agent to continuously monitor the message queues. The transition to the final state is triggered when the distributed simulation is completed.

- Process:

  In the *Process* state, the Coordinator Agent processes the messages stored in its message queues. For example, in an HLA distributed simulation the RTI Agent can use the following procedure to process its data packet queue and honour publish-subscribe relationships:

- Forward data packets to appropriate Federate Agents if the subscribers are currently in the *Receive* state but have not already been sent the data packet.
- Keep track of which data packets have been forwarded to which Federate Agents.
- Remove a data packet from the queue if it has been forwarded to all relevant subscribers.

The RTI Agent can then use the following procedure to process its TAR queue:

- Check if the data packet queue is empty.
- Check if all the Federate Agents have sent in TARs and are currently in the *Receive* state.
- If the two conditions hold, sends TAG messages to all Federate Agents, and flush the TAR queue.

Data packets transmitted by the RTI Agent while in the *Process* state contribute to the active receive time of the subscribing Federate Agents in the *Receive* state. If data packets are transmitted by multi-cast, the RTI Agent only needs to send data packet once each time it enters the *Process* state. The RTI Agent keeps track of the total number of data packets transmitted to all Federate Agents during each round of processing. In each processing round, idle time on each Federate Agent in the *Receive* state can accumulate based on the difference between the number of data packets the Federate Agent receives, and the total number sent by the RTI Agent in that round.

From Figure 6.2, the transition from the *Process* state to the *Wait* state is triggered by a timeout. This timeout is determined by the time required for the Coordinator Agent to complete the previous round of processing. For an HLA distributed simulation, this is the time required for the RTI Agent to transmit all data packets in the previous round of processing. The transition is also triggered by a periodic timeout in order for the Coordinator Agent to continuously cycle between the *Wait* state and the *Process* state while monitoring the message queues.

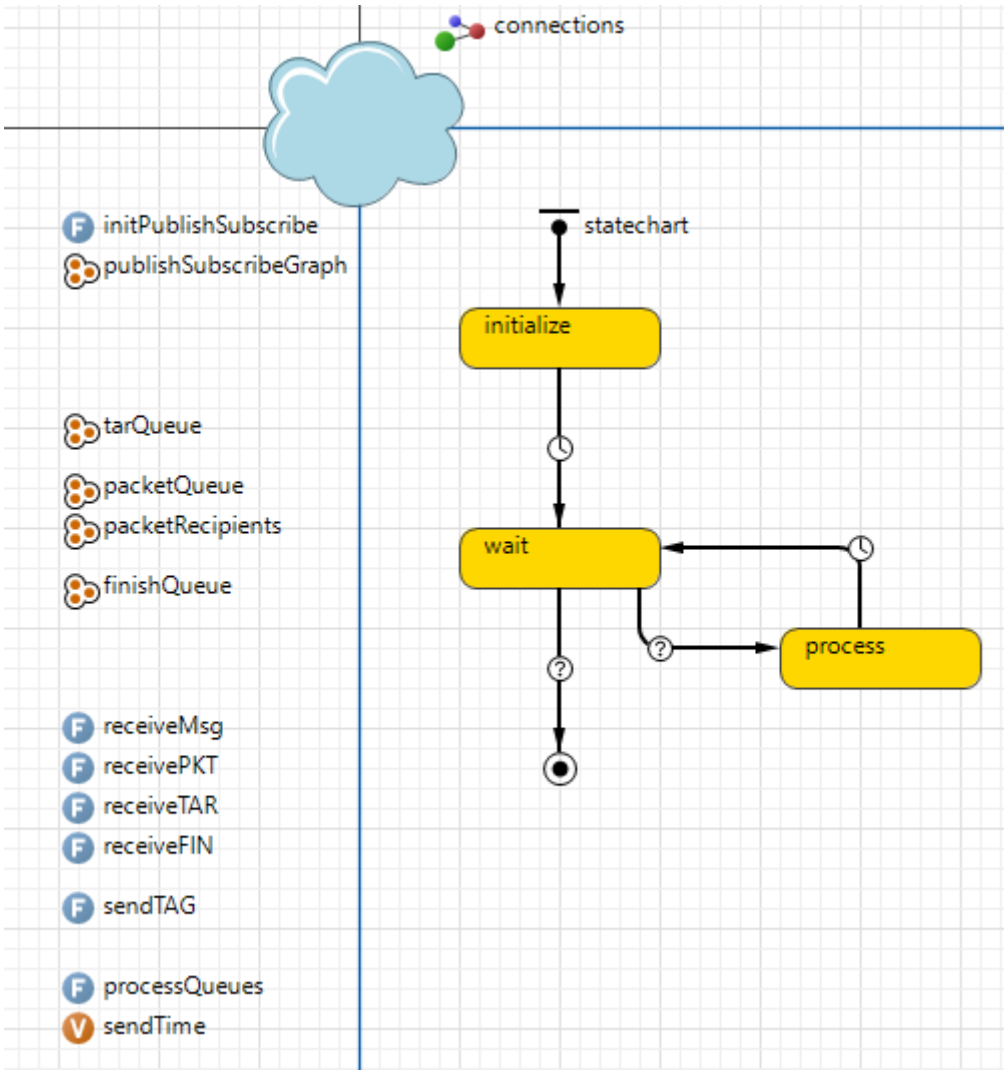

Figure 6.2: Coordinator Agent State Chart

6.3.3 Packet Transmission Time

For each Node Agent, the number of packets transmitted in the *Send* state is calculated based on the following message parameters from Table 6.1 as follows:

- Message Volume
- Message Bytes
- Batch Size
- Compression Ratio
- Payload Bytes

$$\text{Number of Batches} = \left\lceil \frac{\text{Message Volume}}{\text{Batch Size}} \right\rceil$$

$$\text{Bytes per Batch} = \left\lceil \frac{\text{Message Bytes} \times \text{Batch Size}}{\text{Compression Ratio}} \right\rceil$$

$$\text{Number of Packets} = \left\lceil \frac{\text{Bytes per Batch}}{\text{Payload Bytes}} \right\rceil \times \text{Number of Batches}$$

To calculate the transmission time for each data packet over the network, the simplified cost model used by Thakur, Rabenseifner and Gropp (2005) is applied:

$$\text{Packet Transmit Time} = \alpha + n \times \beta$$

Where α, β, and n correspond to framework parameters from Table 6.1:

- α is the *Network Latency* parameter.
- β is the transfer time per byte, which is the inverse of the *Bit Rate* parameter.
- n is the number of bytes in the packet, which consists of *Payload Bytes* and *Header Bytes*.

6.4 Framework Evaluation

6.4.1 Parameter Settings

In this section, the framework is evaluated by implementing a meta-simulation with the AnyLogic simulation toolkit which follows the descriptions of components and interactions discussed in the previous sections. As this concrete implementation of the framework is based on the case studies from Chapter 5, the implemented behaviour and interactions of the agents follows the HLA approach to distributed simulation. The Coordinator Agent is labelled *RTI Agent*, and the Node Agent is labelled *Federate Agent*. Table 6.3 provides a listing of general parameter settings for the meta-simulation. These settings are derived from the characteristics of the experiments conducted in Chapter 5.

Table 6.3: General parameter settings for meta-simulation

Level / Component	Parameter	Value	Remarks
Global	Network Latency	15 µs	Derived from empirical measurement of one-way network latency using the qperf tool.
	Network Bit Rate	10 Gbps	Network type is 10 Gigabit Ethernet. Confirmed empirically with qperf tool.
	Payload Bytes	536 bytes	Default Maximum Segment Size (MSS) for TCP.
	Header Bytes	40 bytes	Default TCP header (20 bytes) + default IP header (20 bytes).
	Fixed Transmission Cost	1 ms	Determined by calibration.
	Variable Transmission Cost	25 µs	Determined by calibration.
Federate Agent	Message Bytes	Triangular (5, 7, 9) bytes	Estimated from case study experiments.

From Table 6.3, settings for the *Fixed Transmission Cost* and *Variable Transmission Cost* parameters are obtained by calibrating the meta-simulation to the results observed for the reduced scene experiment and the batch compression experiment in Chapter 5. In order to perform this calibration, the remaining meta-simulation parameters are set according to the values Table 6.4, which are based on the experiments from Chapter 5. For example, *Message Volume* can be determined from the scene partitioning and *Computation Time* per timestep is estimated from the measured computation time in the original experiments.

Table 6.4: Parameter settings for reduced scene and compression experiment

Level / Component	Parameter	Reduced Scene	Reduced Scene + Batch Compression
Global	Max Timestep	3264 This corresponds to a simulated period of four months plus a 15-day warm-up period, using hourly timesteps.	3264
	Number of Nodes	2	2
Federate Agent	Initialization Time	Triangular (6, 7, 8) s	Triangular (6, 7, 8) s
	Computation Time per Timestep	Triangular (1, 2, 3) ms	Triangular (1, 2, 3) ms
	Message Volume per Timestep	Triangular (200, 270, 340)	Triangular (200, 270, 340)
	Batch Size	1 (no batching)	200
	Compression Ratio	1 (no compression)	2
	Update Period	1 (no skipped timesteps)	1 (no skipped timesteps)

6.4.2 Calibration Results

Using the parameter values from Table 6.3 and Table 6.4, the meta-simulation was calibrated to find appropriate settings for the *Fixed Transmission Cost* and *Variable Transmission Cost* parameters. The calibration was performed by parameter variation over multiple runs for both scenarios. Table 6.5 presents the meta-simulation results for the chosen values of *Fixed Transmission Cost = 1ms* and *Variable Transmission Cost = 25 µs*. Compared to the original experiment, the difference in communication time for the batch compression scenario is 1s. For the normal reduced scene experiment, the difference from the original communication time is 59s.

Table 6.5: Calibration results for meta-simulation

Experiment	Actual Communication Time (s)	Meta-Simulation Communication Time (s)
Reduced Scene	199	125
Reduced Scene + Batch Compression	27	28

6.4.3 Test Results

Further meta-simulation runs were conducted to test the calibrated parameters on the simple scene and the complex scene experiments from Chapter 5. Table 6.6 provides a listing of the scenario-specific parameters used, which are based on the characteristics of the original experiments.

Table 6.6: Parameter settings for simple scene and complex scene

Level / Component	Parameter	Simple Scene	Complex Scene
Global	Max Timestep	9120 This corresponds to a simulated period of one year plus a 15-day warm-up period, using hourly timesteps.	9120
	Number of Nodes	2	4
Federate Agent	Initialization Time	Triangular (250, 300, 350) s	Triangular (0.14×10^6, 1×10^6, 1.33×10^6) s
	Computation Time per Timestep	Triangular (70, 80, 90) ms	Triangular (50, 150, 200) ms
	Message Volume per Timestep	Triangular (2700, 3200, 3600)	Triangular (2900, 5900, 7700)

Level / Component	Parameter	Simple Scene	Complex Scene
	Batch Size	1 (no batching)	1 (no batching)
	Compression Ratio	1 (no compression)	1 (no compression)
	Update Period	1 (no skipped timesteps)	1 (no skipped timesteps)

Table 6.7 presents the results of meta-simulation runs for the simple scene and the complex scene. In the case of the simple scene, the meta-simulation produces 61 min of communication time compared to 70 min for the original case, showing a difference of 9 min. For the complex scene, the meta-simulation produces 240 min compared to the actual result of 198 min, which is a difference of 42 min.

Table 6.7: Meta-simulation results for simple scene and complex scene

Experiment	Actual Communication Time (min)	Meta-Simulation Communication Time (min)
Simple Scene	70	61
Complex Scene	198	240

Another set of meta-simulation runs was conducted for the simple scene experiments for which the batch compression strategy was applied in Chapter 5. Table 6.8 lists the parameter settings used for the meta-simulation, reflecting the conditions of the original experiments. As discussed in Chapter 5, these experiments were run for four months of simulation time. Also, message counts were boosted by allowing federates to transmit attribute updates for all local surface objects, regardless of whether other federates were interested in those updates or not.

Table 6.8: Parameter settings for simple scene with batch compression

Level / Component	Parameter	Simple Scene (Batch Size = 2)	Simple Scene (Batch Size = 1000)
Global	Max Timestep	3264	3264
	Number of Nodes	2	2
Federate Agent	Initialization Time	Triangular (0.7×10^6, 0.75×10^6, 0.8×10^6) s	Triangular (0.7×10^6, 0.75×10^6, 0.8×10^6) s
	Computation Time per Timestep	Triangular (70, 80, 90) ms	Triangular (70, 80, 90) ms
	Message Volume per Timestep	Triangular (18k, 19k, 20k)	Triangular (18k, 19k, 20k)
	Batch Size	2	1000
	Compression Ratio	2	2
	Update Period	1	1

Table 6.9 presents the results of meta-simulation runs for the simple scene with batch compression. For a batch size of 2, the meta-simulation resulted in 63 min of communication time compared to 42 min in the actual experiment, a difference of 21 min. For a batch size of 1000, the meta-simulation shows a communication time of about 1.5 min compared to the actual result of 6 min, which is a difference of 4.5 min.

Table 6.9: Meta-simulation results for simple scene with batch compression

Experiment	Actual Communication Time (min)	Meta-Simulation Communication Time (min)
Simple Scene (Batch Size = 2)	42	63
Simple Scene (Batch Size = 1000)	6	1.5

The communication times predicted by the meta-simulation in the test runs from the previous sections are not exact. Regardless of this, the results provide a good indication of the performance expectation for large-scale simulations, based on parameters calibrated by small-scale experiments. The experiments have also demonstrated how a communication strategy applied to a small-scale experiment can be tested for a large-scale experiment without the need to set up and execute the large-scale scenario.

6.5 Summary

This chapter has provided details of a framework for evaluating performance trends in distributed simulations. The proposed framework is based on the lessons learned from development and experimentation with the Urban Simulation case studies from Chapter 5. These have informed the components of the framework and their interactions with one another, as well as the communication parameters and strategies considered by the framework. An implementation of the framework can be developed for specific distributed simulations in order to evaluate the effect that various communication-related parameters can have on distributed performance. In this chapter, this has been demonstrated by the implementation of a meta-simulation which reproduces the components and interactions described by the framework. The meta-simulation was calibrated with the reduced scene experiments from Chapter 5. Using the calibrated parameters, further meta-simulation runs were conducted to demonstrate the usefulness of the proposed approach for making performance evaluations for distributed simulations.

7 CONCLUSION

7.1 Contribution

The work in this thesis has primarily been concerned with the development of a framework for estimating the execution performance of distributed simulation applications. The framework developed is a contribution to the research field of Parallel and Distributed Simulation. It proposes a meta-simulation approach based on the MAS paradigm to enable performance evaluation for distributed Urban Simulation applications. The proposed framework focuses on aspects of the dynamic relationship between communication and computation that can significantly influence execution performance. It sets out the main components of the meta-simulation and defines the interactions between components during distributed execution. Using the approach proposed by the framework, custom meta-simulations can be created for specific distributed simulation applications. The performance estimates produced from such meta-simulations can support decisions to commit time and computing resources to developing and executing large-scale distributed simulations. Meta-simulations based on the framework can also be employed to investigate the effect that various communication strategies and parameters can have on the distributed simulation under investigation. Comparing the results of various what-if scenarios can aid the search for optimal parameters and communication strategies for the distributed simulation application under investigation. A demonstration of the framework has been provided in Chapter 6 by the implementation of a meta-simulation based on the experiments conducted in Chapter 5. The concrete meta-simulation serves to provide an example of the framework in use and evaluate its usefulness for the purpose of estimating distributed simulation performance. While the case studies chosen for the experiments in Chapter 5 were developed in C++, the framework that has been presented is not specific to the implementation details of the selected simulations. Simulations that are implemented in different programming languages may exhibit variations in execution performance which can be accounted for in a meta-simulation by adjusting the computation time

parameters to appropriate values for each simulation. Similarly, the framework is not specific to the computing nodes or the network employed in the experiments, but can account for differences between platforms by adjusting relevant parameters such as network latency and bit rate.

7.2 Achievement of Aim and Objectives

The aim of the work in this thesis, as set out in Chapter 1, has been achieved by the development of the framework described in Chapter 6. As stated in the aim, the proposed framework considers communication related parameters and strategies that can have significant impact on distributed simulation performance. The aim has been fulfilled by completing the list of objectives that was set out in Chapter 1:

- Literature Review:
  Chapters 2 has covered the relevant research literature on large-scale distributed Urban Simulation.
  Chapter 3 has provided justification for the methodology selected, considering various approaches that have been applied in related work. It has also proposed various communication strategies for consideration in the framework.
  Chapter 4 has explored the relationships between communication and other relevant distributed performance considerations. It has also discussed existing approaches to distributed performance estimation.

- Homogeneous Experimentation:
  Chapter 3 has introduced the selected physical Building Energy Simulation and provided justification for the selection as Case Study One.
  Chapter 5 has discussed Case Study One in more detail. An HLA distributed simulation has been developed for Case Study One and experiments have been conducted, applying communication strategies previously discussed in Chapter 3.

- Heterogeneous Experimentation:
  Chapter 3 has introduced the Building Occupancy Simulation selected for Case Study Two and provided justification for the choice.
  Chapter 5 has detailed the development of a heterogeneous HLA distributed simulation for Building Energy Simulation and Building Occupancy Simulation. Differences in communication patterns between the Case Study One and Case Study Two have been discussed. From these discussions, it has been established that the homogeneous arrangement is a special case of the heterogeneous arrangement, which has more variation in communication patterns. Experiments have been conducted for Case Study Two, and results have been discussed and compared with Case Study One.

- Framework Development:
  Chapter 6 outlines a meta-simulation framework for estimating distributed simulation performance, based on the lessons learned from development and experimentation with the case studies in Chapter 5. The proposed framework makes it possible to investigate the effects that various communication parameters and strategies can have on distributed simulation performance. Due to its reliance on a meta-simulation approach, the proposed framework enables the process of selecting optimal parameters by comparing what-if scenarios. The framework has been demonstrated using an implementation based on experiments conducted in Chapter 5.

7.3 Limitations

The simplifications listed in the following sections are features of the framework which constrain the scope within which it can be applied.

- Reliable Communication:
  It is assumed that a reliable communication protocol such as TCP is used, and that the network connection is 100% reliable. Therefore, the framework does not account for dropped packets and the additional delays that could be introduced by the need to re-transmit corrupted or lost packets.

- Network Congestion:
  The potential impacts of network congestion on communication performance in high traffic scenarios has been simplified. It is assumed that the input parameters are adequate to represent the network performance under any load conditions.

- Communication Topology:
  The network structure has been based on the general structure of the HLA, assuming a star topology with a central RTI and peripheral Federates. Although the HLA experiments conducted in Chapter 5 form the basis of the framework, the framework can be adapted or extended to suit other communication topologies.

- Barrier Synchronization:
  The framework uses conservative barrier synchronization based on the experiments from the case studies in Chapter 5. As other time synchronization approaches have not been considered, the framework will need to be modified or extended to account for the difference if other synchronization approaches are to be employed.

- Output Accuracy:
  The framework focuses on distributed performance dynamics related to exchanging messages between nodes over the network. However, it does not have a means to account for the loss in output accuracy that may result from the use of lossy communication strategies such as those introduced in Chapter3.

- Communication Method:
  The framework assumes that data exchange between computing nodes is accomplished via message passing over the shared network. It does not cover cases in which other communication methods are used, such as shared memory or one-sided communication.

- Size of Experiments:

  The case studies in Chapter 5 have experimented on building scenes containing various numbers of buildings ranging from 50 to 3,000. However, experiments have not been conducted at true city-scale with hundreds of thousands of buildings to simulate.

- Static Interactions:

  For both the homogeneous and heterogeneous case studies in Chapter 5, the publish-subscribe relationships between federates do not change during simulation execution. In other distributed simulations, this may not be the case and message exchange relationships between computing nodes can change dynamically as execution progresses. In such cases, dynamic load balancing to transfer ownership of objects between federates may have important implications for performance.

- Other Performance Bottlenecks

  As discussed in Chapter 4, the performance of distributed simulations can be influenced by other factors apart from communication. The proposed framework focuses on communication and does not address bottlenecks that may arise as a result of other factors.

7.4 Future Work

The framework proposed in this thesis has been developed based on the HLA approach to distributed simulation. This is a practical consequence of the methodology which was suitably employed for the experimental work conducted in Chapter 5. Although this approach has been useful for the development of the framework, some of the limitations listed in the previous section shed light on other considerations which have not yet been fully accounted for by the framework. Future work will involve gradually expanding the framework to consider these other factors where it is deemed useful. The future expansion will also enable the framework to

cover other distributed simulation approaches apart from the HLA, such as the alternatives discussed in Chapter 3. The MAS paradigm, which is essential to the framework, has not been fully exploited for the case studies examined in this work, especially with regard to “intelligent” agents. This concept can potentially be useful in future expansions of the framework. For example, co-operating “intelligent nodes” or an “intelligent co-ordinator” can be introduced in cases where interaction relationships are not static and alternative dynamic load balancing strategies need to be tested. In this case the “intelligent nodes” and “intelligent co-ordinator” can co-operate to dynamically shift workloads during distributed execution. In summary, future work will include expansions to reduce limitations, enrich the framework and enable the creation of useful tools for evaluating the execution performance of large-scale distributed simulations.

REFERENCES

Alexandrov, A., Ionescu, M. F., Schauser, K. E. and Scheiman, C. (1995) 'LogGP: Incorporating Long Messages into the LogP Model', in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures - SPAA '95*. New York, New York, USA: ACM Press, pp. 95–105. doi: 10.1145/215399.215427.

Amdahl, G. M. (1967) 'Validity of the single processor approach to achieving large scale computing capabilities', in *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*. doi: 10.1145/1465482.1465560.

Anagnostou, A., Nouman, A. and Taylor, S. J. E. (2013) 'Distributed hybrid agent-based discrete event emergency medical services simulation', *Proceedings of the 2013 Winter Simulation Conference - Simulation: Making Decisions in a Complex World, WSC 2013*, pp. 1625–1636. doi: 10.1109/WSC.2013.6721545.

Antelmi, A., Cordasco, G., Spagnuolo, C. and Vicidomini, L. (2015) 'On evaluating graph partitioning algorithms for distributed agent based models on networks', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. doi: 10.1007/978-3-319-27308-2_30.

Appuswamy, R., Gkantsidis, C., Narayanan, D., Hodson, O. and Rowstron, A. (2013) 'Scale-up vs scale-out for hadoop: Time to rethink?', *Proceedings of the 4th Annual Symposium on Cloud Computing, SoCC 2013*. doi: 10.1145/2523616.2523629.

Awais, M. U., Palensky, P., Elsheikh, A., Widl, E. and Matthias, S. (2013) 'The High Level Architecture RTI as a master to the functional mock-up interface components', *2013 International Conference on Computing, Networking and Communications, ICNC 2013*, pp. 315–320. doi: 10.1109/ICCNC.2013.6504102.

Barnes, P. D., Carothers, C. D., Jefferson, D. R. and LaPre, J. M. (2013) 'Warp Speed: Executing Time Warp on 1,966,080 Cores', *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation - SIGSIM-PADS '13*, p. 327. doi: 10.1145/2486092.2486134.

Bauer, D. W., Carothers, C. D. and Holder, A. (2009) 'Scalable time warp on blue gene supercomputers', *Proceedings - Workshop on Principles of Advanced and Distributed Simulation, PADS*, pp. 35–44. doi: 10.1109/PADS.2009.21.

Bell, C., Bonachea, D., Cote, Y., Duell, J., Hargrove, P., Husbands, P., Iancu, C., Welcome, M. and Yelick, K. (2003) 'An evaluation of current high-performance networks', *Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2003*, 00(C). doi: 10.1109/IPDPS.2003.1213106.

Birkin, M. and Wu, B. (2012) 'A Review of Microsimulation and Hybrid Agent-Based Approaches', in *Agent-Based Models of Geographical Systems*. Dordrecht: Springer Netherlands, pp. 51–68. doi: 10.1007/978-90-481-8927-4_3.

Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauss, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.-V. and Wolf, S. (2011) 'The Functional Mockup Interface for Tool independent Exchange of Simulation Models', in *Proceedings from the 8th International Modelica Conference, Technical Univeristy, Dresden, Germany*. Dresden, Germany: Linköping University Electronic Press, pp. 105–114. doi: 10.3384/ecp11063105.

Borshchev, A. and Filippov, A. (2004) 'From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools', *22nd International Conference of the System Dynamics Society, 25-29 July 2004*, p. 45.

Bourguignon, F. and Spadaro, A. (2006) 'Microsimulation as a tool for evaluating redistribution policies', *Journal of Economic Inequality*, 4(1), pp. 77–106. doi: 10.1007/s10888-005-9012-6.

Bryant, R. E. (1977) *Simulation of Packet Communication Architecture Computer Systems*, *Tr-188*.

Buchholz, M., Bungartz, H. J. and Vrabec, J. (2011) 'Software design for a highly parallel molecular dynamics simulation framework in chemical engineering', *Journal of Computational Science*, 2(2), pp. 124–129. doi: 10.1016/j.jocs.2011.01.009.

Buyya, R. and Murshed, M. (2002) 'GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing', *Concurrency and Computation: Practice and Experience*, 14(13–15), pp. 1175–1220. doi: 10.1002/cpe.710.

Carothers, C. D., Bauer, D. and Pearce, S. (2002) 'ROSS: A high-performance, low-memory, modular time warp system', *Journal of Parallel and Distributed Computing*, 62(11), pp. 1648–1669. doi: 10.1016/S0743-7315(02)00004-7.

Carothers, C. D., Perumalla, K. S. and Fujimoto, R. M. (1999) 'Efficient Optimistic Parallel Simulations Using Reverse Computation', *ACM Transactions on Modeling and Computer Simulation*, 9(3), pp. 224–253.

Carothers, C. and Perumalla, K. S. (2010) 'On deciding between conservative and optimistic approaches on massively parallel platforms', in *Proceedings of the 2010 Winter Simulation Conference*. IEEE, pp. 678–687. doi: 10.1109/WSC.2010.5679119.

Chandy, M. K. and Misra, J. (1979) 'Distributed Simulation: A Case Study in Design and Verification of Distributed Programs', *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 5(5), pp. 440–452.

Chandy, M. K. and Misra, J. (1981) 'Asynchronous Distributed Simulation via a sequence of Parallel computations', *Communications of the ACM*, 24(11), pp. 198–20. doi: 10.1016/0010-4655(86)90224-9.

Chapman, J. (2018) *No-MASS Repository*, *Github repository*. Available at: https://github.com/jacoblchapman/No-MASS (Accessed: 4 September 2021).

Chapman, J., Siebers, P.-O. and Robinson, D. (2018) 'On the multi-agent stochastic simulation of occupants in buildings', *Journal of Building Performance Simulation*, 11(5), pp. 604–621. doi: 10.1080/19401493.2017.1417483.

Chen, D., Wang, L., Zomaya, A. Y., Dou, M. G., Chen, J., Deng, Z. and Hariri, S. (2015) 'Parallel simulation of complex evacuation scenarios with adaptive agent

models', *IEEE Transactions on Parallel and Distributed Systems*, 26(3), pp. 847–857. doi: 10.1109/TPDS.2014.2311805.

CityGML (2020) *CityGML Standard*. Available at: http://www.citygml.org/ (Accessed: 19 October 2020).

Coakley, S., Richmond, P., Gheorghe, M., Chin, S., Worth, D., Holcombe, M. and Greenough, C. (2016) 'Large-Scale Simulations with FLAME', *Intelligent Agents in Data-intensive Computing*, 14, pp. 1–20. doi: 10.1007/978-3-319-23742-8.

Collier, N. and North, M. (2012) 'Parallel agent-based simulation with Repast for High Performance Computing', *Simulation*, 89(November), pp. 1215–1235. doi: 10.1177/0037549712462620.

Collier, N., Ozik, J. and Macal, C. M. (2015) 'Large-scale agent-based modeling with repast HPC: A case study in parallelizing an agent-based model', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9523, pp. 454–465. doi: 10.1007/978-3-319-27308-2_37.

Cordasco, G., De Chiara, R., Mancuso, A., Mazzeo, D., Scarano, V. and Spagnuolo, C. (2013) 'Bringing together efficiency and effectiveness in distributed simulations: The experience with D-Mason', *Simulation*, 89(10), pp. 1236–1253. doi: 10.1177/0037549713489594.

Cordasco, G., Spagnuolo, C. and Scarano, V. (2017) 'Work partitioning on parallel and distributed agent-based simulation', in *Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2017*. doi: 10.1109/IPDPSW.2017.87.

Crawley, D. B., Lawrie, L. K., Winkelmann, F. C., Buhl, W. F., Huang, Y. J., Pedersen, C. O., Strand, R. K., Liesen, R. J., Fisher, D. E., Witte, M. J. and Glazer, J. (2001) *EnergyPlus: Creating a new-generation building energy simulation program*, *Energy and Buildings*. doi: 10.1016/S0378-7788(00)00114-6.

Culler, D., Karp, R., Patterson, D., Sahay, A., Erik Schauser, K., Santos, E., Subramonian, R. and Von Eicken, T. (1993) 'LogP: Towards a realistic model

of parallel computation', *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP*, Part F1296, pp. 1–12. doi: 10.1145/155332.155333.

Das, S., Fujimoto, R. M., Panesar, K., Allison, D. and Hybinette, M. (1994) 'GTW: a time warp system for shared memory multiprocessors', *Proceedings of Winter Simulation Conference*, pp. 1332–1339.

Dementiev, R., Kettner, L. and Sanders, P. (2008) 'STXXL: standard template library for XXL data sets', *Software: Practice and Experience*, 38(6), pp. 589–637. doi: 10.1002/spe.844.

Epstein, J. M. (2008) 'Why Model?', *Jasss*, 11(4), p. 12. doi: 10.1080/01969720490426803.

Epstein, J. M. and Axtell, R. (1996) *Growing artificial societies: social science from the bottom up*.

Eurostat (2019) *Energy, Transport and environment statistics. 2019 edition*. doi: 10.2785/660147.

Ferenci, S. L., Perumalla, K. S. and Fujimoto, R. M. (2000) 'An Approach for federating parallel simulators', *Proceedings of the Workshop on Parallel and Distributed Simulation, PADS*, pp. 63–70. doi: 10.1109/pads.2000.847145.

Ferscha, A. and Tripathi, S. K. (1998) *Parallel and distributed simulation of discrete event systems*. doi: 10.1.1.19.6226.

Flores-Contreras, J., Duran-Limon, H. A., Chavoya, A. and Almanza-Ruiz, S. H. (2020) *Performance prediction of parallel applications: a systematic literature review*, *Journal of Supercomputing*. Springer US. doi: 10.1007/s11227-020-03417-5.

Forrester, J. (1970) 'Urban dynamics', *IMR; Industrial Management Review (pre-1986)*.

Forrester, J. W. (1968) 'Industrial Dynamics—A Response to Ansoff and Slevin', *Management Science*, 14(9), pp. 601–618. doi: 10.1287/mnsc.14.9.601.

Fortune, S. and Wyllie, J. (1978) 'Parallelism in random access machines', in *Proceedings of the tenth annual ACM symposium on Theory of computing -*

STOC '78. New York, New York, USA: ACM Press, pp. 114–118. doi: 10.1145/800133.804339.

Foucquier, A., Robert, S., Suard, F., Stéphan, L. and Jay, A. (2013) 'State of the art in building modelling and energy performances prediction: A review', *Renewable and Sustainable Energy Reviews*, 23, pp. 272–288. doi: 10.1016/J.RSER.2013.03.004.

Fujimoto, R. M. (1990a) 'Parallel discrete event simulation', *Commun. ACM*, 33, pp. 30–53. doi: 10.1145/84537.84545.

Fujimoto, R. M. (1990b) 'Performance of time warp under synthetic workloads', *Simulation Series*, pp. 23–28.

Fujimoto, R. M. (1998) 'Time Management in the High Level Architecture', *Scenario*, 0280(6), pp. 388–400. doi: 10.1177/003754979807100604.

Fujimoto, R. M. (2000) *Parallel and distributed simulation systems*. New York: Wiley.

Fujimoto, R. M. (2015) 'Parallel and distributed simulation', in *2015 Winter Simulation Conference (WSC)*. IEEE, pp. 45–59. doi: 10.1109/WSC.2015.7408152.

Fujimoto, R. M. (2016) 'Research Challenges in Parallel and Distributed Simulation', *ACM Trans. Model. Comput. Simul. Article*, 26(29). doi: 10.1145/2866577.

Fujimoto, R. M. and Hoare, P. (1998) 'HLA RTI Performance in High Speed LAN Environments', in *Proceedings of the Fall Simulation Interoperability Workshop*.

Fujimoto, R. M., Perumalla, K., Park, A., Wu, H., Ammar, M. H. and Riley, G. F. (2003) 'Large-scale network simulation: How big? How fast?', in *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, pp. 116–123. doi: 10.1109/MASCOT.2003.1240649.

Georgia Tech (2001) *Federated Simulations Development Kit*. Available at: https://www.cc.gatech.edu/computing/pads/fdk.html (Accessed: 4 September 2021).

De Grande, R. E. and Boukerche, A. (2011) 'Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems', *Journal of Parallel and Distributed Computing*. doi: 10.1016/j.jpdc.2010.04.001.

Gropp, W., Gropp, W. D., Lusk, E., Skjellum, A. and Lusk, A. D. F. E. E. (1999) *Using MPI: portable parallel programming with the message-passing interface*. MIT press.

Gustafson, J. L. (1988) 'Reevaluating amdahl's law', 31(5), pp. 532–533.

Gutlein, M., Baron, W., Renner, C. and Djanatliev, A. (2020) 'Performance Evaluation of HLA RTI Implementations', *Proceedings of the 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2020*. doi: 10.1109/DS-RT50469.2020.9213641.

Hambrusch, S. E., Hameed, F. and Khokhar, A. A. (1995) 'Communication operations on coarse-grained mesh architectures', *Parallel Computing*, 21(5), pp. 731–751. doi: 10.1016/0167-8191(94)00110-V.

Hapner, M., Burridge, R., Sharma, R., Fialli, J. and Stout, K. (2002) 'Java Message Service', *Sun Microsystems Inc., Santa Clara, CA*, 9.

Heath, B. and Hill, R. (2010) 'Some insights into the emergence of agent-based modelling', *Journal of Simulation*, 4, pp. 163–16916. doi: 10.1057/jos.2010.16.

Helbing, D., Hennecke, A., Shvetsov, V. and Treiber, M. (2002) 'Micro- and macro-simulation of freeway traffic', *Mathematical and Computer Modelling*, 35(5–6), pp. 517–547. doi: 10.1016/S0895-7177(02)80019-X.

Henderson, T. R. and Riley, G. F. (2006) 'Network Simulations with the ns-3 Simulator', *Proc. Sigcomm*, p. 527.

Ho, N. M., Thoai, N. and Wong, W. F. (2015) 'Multi-agent simulation on multiple GPUs', *Simulation Modelling Practice and Theory*. doi: 10.1016/j.simpat.2015.06.008.

Hodson, D. D. and Baldwin, R. O. (2009) 'Characterizing, Measuring, and Validating the Temporal Consistency of Live--Virtual--Constructive Environments', *Simulation*, 85(10), pp. 671–682. doi: 10.1177/0037549709340732.

Hodson, D. D. and Hill, R. R. (2014) 'The art and science of live, virtual, and constructive simulation for test and analysis', *Journal of Defense Modeling and Simulation: Applications, Methodology Technology*, 11(2), pp. 77–89. doi: 10.1177/1548512913506620.

Hofer, R. C. and Loper, M. L. (1995) 'DIS Today', *Proceedings of the IEEE*, 83(8), pp. 1124–1137. doi: 10.1109/5.400453.

Hong, T. (LBNL), Chen, Y. (LBNL), Lee, S. H. (LBNL), Piette, M. P. (LBNL), Chen, Y. (LBNL) and Piette, M. P. (LBNL) (2016) 'CityBES: A web-based platform to support city-scale building energy efficiency', *5th International Urban Computing Workshop, At San Francisco*, (August), p. 10.

Horni, A., Nagel, K. and Axhausen, K. W. (2016) *The Multi-Agent Transport Simulation MATSim*. United Kingdom: Ubiquity Press. doi: 10.5334/baw.

Huang, Q., Huang, Z., Werstein, P. and Purvis, M. (2008) 'GPU as a general purpose computing resource', in *Parallel and Distributed Computing, Applications and Technologies, PDCAT Proceedings*. doi: 10.1109/PDCAT.2008.38.

IEEE (2010a) 'IEEE Std 1516-2010, High Level Architecture (HLA) Framework and Rules'. doi: 10.1109/IEEESTD.2010.5553440.

IEEE (2010b) 'IEEE Std 1516.1-2010, High Level Architecture (HLA) Federate Interface Specification'. doi: 10.1109/IEEESTD.2010.5557728.

IEEE (2010c) 'IEEE Std 1516.2-2010, High Level Architecture (HLA) Object Model Template (OMT) Specification'. doi: 10.1109/IEEESTD.2010.5557731.

Imgrund, M. and Arth, A. (2017) 'Rambrain - a library for virtually extending physical memory', *SoftwareX*. doi: 10.1016/j.softx.2017.07.004.

Improbable (2020) *Improbable SpatialOS*. Available at: https://improbable.io/ (Accessed: 4 September 2021).

Ino, F., Fujimoto, N. and Hagihara, K. (2001) 'LogGPS: A parallel computational model for synchronization analysis', *SIGPLAN Notices (ACM Special Interest Group on Programming Languages)*, 36(7), pp. 133–142. doi: 10.1145/568014.379592.

Jafer, S., Liu, Q. and Wainer, G. (2013) 'Synchronization methods in parallel and

distributed discrete-event simulation', *Simulation Modelling Practice and Theory*, 30, pp. 54–73. doi: 10.1016/j.simpat.2012.08.003.

Jain, A., Robinson, D., Dilkina, B. and Fujimoto, R. (2016) 'An approach to integrate inter-dependent simulations using HLA with applications to sustainable urban development', in *2016 Winter Simulation Conference (WSC)*. IEEE, pp. 1218–1229. doi: 10.1109/WSC.2016.7822178.

Jefferson, D. (1990) 'Virtual Time II: Storage Management in Conservative and Optimistic Systems', *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pp. 75–89. doi: 10.1145/93385.93403.

Jefferson, D., Beckman, B., Wieland, F., Blume, L., Di Loreto, M., Hontalas, P., Laroche, P., Sturdevant, K., Tupman, J., Warren, V., Wedel, J., Younger, H. and Bellenot, S. (1987) 'Distributed simulation and the time warp operating system', *Proceedings of the 11th ACM Symposium on Operating Systems Principles, SOSP 1987*, pp. 77–93. doi: 10.1145/41457.37508.

Jefferson, D. R. (1985) 'Virtual time', *ACM Transactions on Programming Languages and Systems*, 7(3), pp. 404–425. doi: 10.1145/3916.3988.

Jefferson, D. R. and Sowizral, H. (1982) *Fast Concurrent Simulation Using the Time Warp Mechanism. Part I. Local Control*. RAND CORP SANTA MONICA CA.

JGroups (2020) *JGroups: a toolkit for reliable multicast communication*. Available at: http://www.jgroups.org/ (Accessed: 4 September 2021).

Johari, F., Peronato, G., Sadeghian, P., Zhao, X. and Widén, J. (2020) 'Urban building energy modeling: State of the art and future prospects', *Renewable and Sustainable Energy Reviews*, 128(September 2019). doi: 10.1016/j.rser.2020.109902.

Karp, A. H. and Flatt, H. P. (1990) 'Measuring Parallel Processor Performance', *Communications of the ACM*, 33(5), pp. 539–543. doi: 10.1145/78607.78614.

Kirkwood, C. (1998) 'System dynamics methods', *College of Business Arizona State University*.

Ko, S. and Han, W.-S. (2018) 'TurboGraph++', pp. 395–410. doi: 10.1145/3183713.3196915.

Komann, M., Kauhaus, C. and Fey, D. (2005) 'Calculation of Single-File Diffusion Using Grid-Enabled Parallel Generic Cellular Automata Simulation', in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pp. 528–535. doi: 10.1007/11557265_67.

Kuhl, F., Weatherly, R. and Dahmann, J. (1999) 'Creating computer simulation systems: an introduction to the high level architecture'.

Kumar, D. (1992) 'Systems with Low Distributed Simulation Overhead', *IEEE Transactions on Parallel and Distributed Systems*, 3(2), pp. 155–165. doi: 10.1109/71.127257.

Kyrola, A., Blelloch, G. and Guestrin, C. (2012) 'GraphChi', *USENIX Symposium on operating systems design and implementation*, 10(31), pp. 31–46.

Law, A. M. and Kelton, W. D. (1984) 'Confidence Intervals for Steady-State Simulations: I. a Survey of Fixed Sample Size Procedures.', *Operations Research*, 32(6), pp. 1221–1239. doi: 10.1287/opre.32.6.1221.

Law, A. M. and Kelton, W. D. (2000) *Simulation Modelling and Analysis*. McGraw-Hill.

Lees, M. H., Logan, B., Oguara, T. and Theodoropoulos, G. (2003) 'Simulating Agent-Based Systems with HLA: The case of SIM_AGENT -- Part II', *2003 European Simulation Interoperability Workshop*.

Lemeire, J. (2001) 'Performance factors in parallel discrete event simulation', in *Proc. of the 15th European Simulation Multiconference*. Prague.

Lin, K.-C., Blair, J. L. and Woodyard, J. M. (1997) 'Study on Dead-Reckoning Translation in High-Level Architecture', *SIMULATION*, 69(2), pp. 103–109. doi: 10.1177/003754979706900203.

Lin, Y.-B. and Edward, D. L. (1991) 'A Study of Time Warp Rollback Mechanisms', *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 1(1), pp. 51–72. doi: 10.1145/102810.102813.

Lin, Y. C. and Snyder, L. (2009) *Principles of Parallel Programming*. Pearson.

Lubachevsky, B., Schwartz, A. and Weiss, A. (1991) 'An Analysis of Rollback-Based Simulation', *ACM Transactions on Modeling and Computer Simulation*

(TOMACS), 1(2), pp. 154–193. doi: 10.1145/116890.116912.

Lysenko, M. and D 'Souza, R. M. (2008) 'A Framework for Megascale Agent Based Model Simulations on Graphics Processing Units', *Journal of Artificial Societies and Social Simulation*, 11(4).

Macal, C. and North, M. (2015) 'Introductory tutorial: Agent-based modeling and simulation', in *Proceedings - Winter Simulation Conference*. doi: 10.1109/WSC.2014.7019874.

Malinga, L. and Le Roux, W. H. (2009) 'HLA RTI performance evaluation', *SISO European Simulation Interoperability Workshop 2009, EURO SIW 2009*, pp. 37–42.

Menassa, C. C., Kamat, V. R., Lee, S., Azar, E., Feng, C. and Anderson, K. (2014) 'Conceptual Framework to Optimize Building Energy Consumption by Coupling Distributed Energy Simulation and Occupancy Models', *Journal of Computing in Civil Engineering*, 28(1), pp. 50–62. doi: 10.1061/(ASCE)CP.1943-5487.0000299.

Meyer, R. (2014) 'Event-Driven Multi-agent Simulation', in *International Workshop on Multi-Agent Systems and Agent-Based Simulation*. Springer. doi: 10.1007/978-3-319-14627-0.

Milgram, P. and Kishino, F. (1994) 'A taxonomy of mixed reality visual displays', *IEICE Transactions on Information Systems*, 77(12), pp. 1321–1329.

Miller, D. C. and Thorpe, J. A. (1995) 'SIMNET: the advent of simulator networking', *Proceedings of the IEEE*, 83(8), pp. 1114–1123. doi: 10.1109/5.400452.

Minson, R. and Theodoropoulos, G. (2008) 'Distributing RePast agent-based simulations with HLA', *Concurrency and Computation: Practice and Experience*, 20(10), pp. 1225–1256. doi: 10.1002/cpe.1280.

Misra, J. (1986) 'Distributed Discrete-event Simulation', *ACM Comput. Surv.*, 18(1), pp. 39–65. doi: 10.1145/6462.6485.

Morecroft, J. and Robinson, S. (2005) 'Explaining Puzzling Dynamics: Comparing the Use of System Dynamics and Discrete- Event Simulation', *Proceedings of the 23rd International Conference of the System Dynamics Society*, pp. 1–32.

Mostaccio, D., Suppi, R. and Luque, E. (2005) 'Simulation of Ecologic Systems Using MPI', in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pp. 449–456. doi: 10.1007/11557265_57.

Nagel, K. and Schreckenberg, M. (1992) 'A cellular automaton model for freeway traffic', *J. Phys. I France*, 2, pp. 2221–2229. doi: 10.1051/jp1:1992277.

Neema, H., Gohl, J., Lattmann, Z., Sztipanovits, J., Karsai, G., Neema, S., Bapty, T., Batteh, J., Tummescheit, H. and Sureshkumar, C. (2014) 'Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems', *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, 96, pp. 235–245. doi: 10.3384/ecp14096235.

Nicol, D. M. (1993) 'The Cost of Conservative Synchronization in Parallel Discrete Event Simulations', *Journal of the ACM (JACM)*, 40(2), pp. 304–333. doi: 10.1145/151261.151266.

North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M. and Sydelko, P. (2013) 'Complex adaptive systems modeling with Repast Simphony'. doi: 10.1186/2194-3206-1-3.

Nouvel, R., Brassel, K.-H., Bruse, M., Duminil, E., Coors, V., Eicker, U. and Robinson, D. (2015) *SIMSTADT, a New Workflow-driven Urban Energy Simulation Platform for CityGML City Models*, *CISBAT International conference*. doi: 10.5075/epfl-cisbat2015-889-894.

Oguara, T., Theodoropolous, G., Dan, C., Logan, B. and Lees, M. (2007) *PDES-MAS: A Unifying Framework for the Distributed Simulation of Multi-Agent Systems*.

Orcutt, G. H. (1957) 'A new type of socio-economic system', *Review of Economics and Statistics*, 39(2), pp. 116–123. doi: 10.2307/1928528.

Ouro, P., Fraga, B., Lopez-Novoa, U. and Stoesser, T. (2019) 'Scalability of an Eulerian-Lagrangian large-eddy simulation solver with hybrid MPI/OpenMP parallelisation', *Computers and Fluids*, 179, pp. 123–136. doi: 10.1016/j.compfluid.2018.10.013.

Owens, J. D., Luebke, D., Govindraju, N., Harris, M., Kruger, J., Lefohn, A. E. and

Purcell, T. J. (2006) 'A Survey of General Purpose Computation on Graphics Hardware', *Computer Graphics Forum*, 26(1), pp. 80–113. doi: 10.1111/j.1467-8659.2007.01012.x.

Park, E., Eidenbenz, S., Santhi, N., Chapuis, G. and Settlemyer, B. (2016) 'Parameterized benchmarking of parallel discrete event simulation systems: Communication, computation, and memory', in *Proceedings - Winter Simulation Conference*. IEEE, pp. 2836–2847. doi: 10.1109/WSC.2015.7408388.

Parry, H. R. and Bithell, M. (2012) 'Large Scale Agent-Based Modelling: A Review and Guidelines for Model Scaling', in *Agent-Based Models of Geographical Systems*. Dordrecht: Springer Netherlands, pp. 271–308. doi: 10.1007/978-90-481-8927-4_14.

Perumalla, K. S. (2005) 'μsik — A Micro-Kernel for Parallel/Distributed Simulation Systems', in *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*. IEEE, pp. 59–68. doi: 10.1109/PADS.2005.1.

Perumalla, K. S. (2006) 'Parallel and distributed simulation: Traditional techniques and recent advances', in *Proceedings - Winter Simulation Conference*. doi: 10.1109/WSC.2006.323041.

Perumalla, K. S. (2007) 'Scaling time warp-based discrete event execution to 10^4 processors on a Blue Gene supercomputer', *Proceedings of the 4th international conference on Computing frontiers - CF '07*, p. 69. doi: 10.1145/1242531.1242543.

Peschlow, P. and Martini, P. (2007) 'Efficient Analysis of SimultaneousEv ents in Distributed Simulation', in *Proceedings - IEEE International Symposium on Distributed Simulation and Real-Time Applications, DS-RT*, pp. 244–251. doi: 10.1109/DS-RT.2007.21.

Plesser, H. E., Eppler, J. M., Morrison, A., Diesmann, M. and Gewaltig, M.-O. (2007) 'Efficient Parallel Simulation of Large-Scale Neuronal Networks on Clusters of Multiprocessor Computers', in *European Conference on Parallel Processing*, pp. 672–681. doi: 10.1007/978-3-540-74466-5_71.

Ponnusamy, R., Choudhary, A. and Fox, G. (1992) 'Communication overhead on the

CM5: an experimental performance evaluation', in *[Proceedings 1992] The Fourth Symposium on the Frontiers of Massively Parallel Computation*. IEEE Comput. Soc. Press, pp. 108–115. doi: 10.1109/FMPC.1992.234899.

Portico (2020) *Portico Repository*. Available at: https://github.com/openlvc/portico (Accessed: 14 October 2020).

Rajkumar, R., Lee, I., Sha, L. and Stankovic, J. (2010) 'Cyber-physical systems: The next computing revolution', *Proceedings - Design Automation Conference*, pp. 731–736. doi: 10.1145/1837274.1837461.

Reinhart, C. F. and Davila, C. C. (2016) 'Urban building energy modeling - A review of a nascent field', *Building and Environment*, 97, pp. 196–202. doi: 10.1016/j.buildenv.2015.12.001.

Reynolds, C. W. (1987) 'Flocks, herds and schools: A distributed behavioral model', *ACM SIGGRAPH Computer Graphics*, 21(4), pp. 25–34. doi: 10.1145/37402.37406.

Richmond, P. (2015) 'Complex Systems Simulation with CUDA'.

Richmond, P. and Chimeh, M. K. (2017) 'Flame GPU: Complex system simulation framework', in *Proceedings - 2017 International Conference on High Performance Computing and Simulation, HPCS 2017*. doi: 10.1109/HPCS.2017.12.

Righter, R. and Walrand, J. C. (1989) 'Distributed simulation of discrete event systems', *Proceedings of the IEEE*, 77(1), pp. 99–113. doi: 10.1109/5.21073.

Robinson, D. (2012) 'The Urban Radiant Environment', in *Computer Modelling for Sustainable Urban Design*. Routledge, pp. 35–74.

Robinson, S. (2014) *Simulation: the practice of model development and use*. Palgrave Macmillan.

Rosser, J. F., Long, G., Zakhary, S., Boyd, D. S., Mao, Y. and Robinson, D. (2019) 'Modelling Urban Housing Stocks for Building Energy Simulation using CityGML EnergyADE', *ISPRS International Journal of Geo-Information*, 8(4), p. 163. doi: 10.3390/ijgi8040163.

Rousset, A., Herrmann, B., Lang, C. and Philippe, L. (2016) 'A survey on parallel

and distributed multi-agent systems for high performance computing simulations', *Computer Science Review*, 22(2), pp. 27–46. doi: 10.1016/j.cosrev.2016.08.001.

Schelling, T. C. (1971) 'Dynamic models of segregation†', *The Journal of Mathematical Sociology*, 1(2), pp. 143–186. doi: 10.1080/0022250X.1971.9989794.

Schriber, T. J. (2009) 'Simulation for the masses: Spreadsheet-based Monte Carlo simulation', in *Proceedings of the 2009 Winter Simulation Conference (WSC)*. IEEE, pp. 1–11. doi: 10.1109/WSC.2009.5429310.

Schriber, T. J., Brunner, D. T. and Smith, J. S. (2017) 'Inside discrete-event simulation software: How it works and why it matters', in *2017 Winter Simulation Conference (WSC)*. IEEE, pp. 735–749. doi: 10.1109/WSC.2017.8247828.

Siebers, P.-O., Macal, C. M., Garnett, J., Buxton, D. and Pidd, M. (2010) 'Discrete-event simulation is dead, long live agent-based simulation!', *Journal of Simulation*, 4(3), pp. 204–210. doi: 10.1057/jos.2010.14.

Skillicorn, D. B. and Talia, D. (1998) 'Models and Languages for Parallel Computation', *ACM Computing Surveys*, 30(2), pp. 123–169. doi: 10.1145/280277.280278.

Snyder, L. (1986) 'Type Architectures, Shared Memory and the Corollary of Modest Potential', pp. 289–317.

Somani, A. K. and Sansano, A. M. (1997) *Minimizing Overhead in Parallel Algorithms Through Overlapping Communication/Computation*. Institute for Computer Applications in Science and Engineering HAMPTON VA.

Strassburger, S., Schulze, T. and Fujimoto, R. M. (2008) 'Future Trends in Distributed Simulation and Distributed Virtual Environments: Results of a Peer Study', *Proceedings of the 2008 Winter Simulation Conference*, pp. 777–785. doi: 10.1109/WSC.2008.4736140.

Strumpen, V. and Casavant, T. L. (1994) 'Exploiting communication latency hiding for parallel network computing: model and analysis', *Proceedings of the Internatoinal Conference on Parallel and Distributed Systems - ICPADS*, pp.

622–627. doi: 10.1109/icpads.1994.590409.

Sun, X. H. and Ni, L. M. (1993) 'Scalable Problems and Memory-Bounded Speedup', *Journal of Parallel and Distributed Computing*, 19(1), pp. 27–37. doi: 10.1006/jpdc.1993.1087.

Sutter, H. (2005) *The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software*.

Tan, G. and Lim, K. C. (2004) 'Load Distribution Services in HLA', in *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*. IEEE, pp. 133–141. doi: 10.1109/DS-RT.2004.27.

Tay, S. C. and Teo, Y. M. (2001) 'Performance optimization of throttled time-warp simulation', *Proceedings. 34th Annual Simulation Symposium*, pp. 211–218. doi: 10.1109/SIMSYM.2001.922134.

Taylor, S. J. E. (2019) 'Distributed simulation: state-of-the-art and potential for operational research', *European Journal of Operational Research*, 273(1), pp. 1–19. doi: 10.1016/j.ejor.2018.04.032.

Thakur, R., Rabenseifner, R. and Gropp, W. (2005) 'Optimization of collective communication operations in MPICH', *International Journal of High Performance Computing Applications*, 19(1), pp. 49–66. doi: 10.1177/1094342005051521.

Theodoropoulos, G. and Logan, B. (1999) 'A framework for the distributed simulation of agent-based systems', in *Proceedings of the 13th European Simulation Multiconference (ESM'99)*, pp. 58–65.

Top500 (2020) *Top500 Supercomputer Sites*. Available at: https://www.top500.org/lists/top500/2021/06/ (Accessed: 4 September 2021).

Topçu, O. and Oğuztüzün, H. (2017) *Guide to Distributed Simulation with HLA*, *Guide to Distributed Simulation with HLA*. Springer. doi: 10.1007/978-3-319-61267-6.

United Nations (2014) 'World Urbanization Prospects: The 2014 Revision, Highlights. Department of Economic and Social Affairs', *Population Division, United Nations*.

Valiant, L. G. (1990) 'A bridging model for parallel computation', *Communications of the ACM*. doi: 10.1145/79173.79181.

Waddell, P. (2002) 'UrbanSim: Modeling Urban Development for Land Use, Transportation, and Environmental Planning', *Journal of the American Planning Association*, 68(3), pp. 297–314. doi: 10.1080/01944360208976274.

Wang, K., Siebers, P. and Robinson, D. (2017) 'Towards Generalized Co-simulation of Urban Energy Systems', in *Procedia Engineering*. doi: 10.1016/j.proeng.2017.07.092.

White, K. P. and Ingalls, R. G. (2016) 'Introduction to simulation', *Proceedings - Winter Simulation Conference*, 2016-Febru, pp. 1741–1755. doi: 10.1109/WSC.2015.7408292.

Wilson, A. L. and Weatherly, R. M. (1994) 'Aggregate level simulation protocol: an evolving system', *Winter Simulation Conference Proceedings*, pp. 781–787. doi: 10.1109/wsc.1994.717433.

Wolfram, S. (2002) *A new kind of science*. Wolfram media Champaign, IL.

Wooldridge, M. and Jennings, N. R. (1995) 'Intelligent agents: Theory and practice', *The Knowledge Engineering Review*. doi: 10.1017/S0269888900008122.

Zakhary, S., Allen, A., Siebers, P. and Robinson, D. (2016) 'A computational workflow for urban micro-simulation of buildings' energy performance', in *Urban Transitions Global Summit*.

Zakhary, S., Rosser, J., Siebers, P.-O., Mao, Y. and Robinson, D. (2020) 'Using unsupervised learning to partition 3D city scenes for distributed building energy microsimulation', *Environment and Planning B: Urban Analytics and City Science*, p. 239980832091431. doi: 10.1177/2399808320914313.

Zehe, D., Knoll, A., Cai, W. and Aydt, H. (2015) 'SEMSim Cloud Service: Large-scale urban systems simulation in the cloud'. doi: 10.1016/j.simpat.2015.05.005.

Zhang, T., Siebers, P.-O. and Aickelin, U. (2012) 'A three-dimensional model of residential energy consumer archetypes for local energy policy design in the UK', *Energy Policy*, 47, pp. 102–110. doi: 10.1016/j.enpol.2012.04.027.

Zhu, X., Han, W. and Chen, W. (2015) 'Gridgraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning', *Proceedings of the 2015 USENIX Annual Technical Conference, USENIX ATC 2015*, pp. 375–386.