Department of Computer Science
University of Pretoria
Pretoria
South Africa

Migrating to a Real-Time Distributed Parallel Simulator Architecture

by

Bernardt Duvenhage

June 18, 2008

Submitted in partial fulfilment of the requirements for the degree
Masters of Science (Computer Science) in the Faculty
of Engineering, Built Environment and Information Technology

University of Pretoria
Pretoria, South Africa

Superviser: Professor D. G. Kourie

Acknowledgements

I would like to thank my supervisor and adviser, Professor Derrick G. Kourie. His support and excellent critical review of my research have brought me much appreciated insight and have certainly improved on the contents of this dissertation.

Many thanks to the my colleagues at the Council for Scientific and Industrial Research (CSIR)—especially Jan Roodt, Cobus Nel and Herman le Roux—with whom I have had many discussions surrounding modelling and simulation, our simulation capability and the various simulators that we have worked on. Herman le Roux is the technical team leader for the Systems Modelling simulation group and I would like to single him out and thank him for what I have learned from him.

The funding provided by the Armaments Corporation of SA (ARMSCOR), the South-African DoD, the CSIR, the University of Pretoria and the National Research Foundation is very much appreciated. Without their funding the research would not have been possible.

Special thanks goes out to the modelling and simulation community for the work they are all doing and for their gargantuan shoulders that have helped me to reach where I am now.

Finally, I would like to thank my very lovely wife with all my heart for her patience and understanding while I was working long hours to finish off this dissertation. Also to the rest of my family and my friends, I would like you to know that I will be resurfacing in society soon and look forward to seeing you all more often.

Abstract

The South African National Defence Force (SANDF) currently requires a system of systems simulation capability for supporting the different phases of a Ground Based Air Defence System (GBADS) acquisition program. A non-distributed, fast-as-possible simulator and its architectural predecessors developed by the Council for Scientific and Industrial Research (CSIR) was able to provide the required capability during the *concept and definition* phases of the acquisition life cycle. The non-distributed simulator implements a 100Hz logical time Discrete Time System Specification (DTSS) in support of the existing models. However, real-time simulation execution has become a prioritised requirement to support the development phase of the acquisition life cycle.

This dissertation is about the ongoing migration of *the non-distributed simulator* to a practical simulation architecture that supports the real-time requirement. The simulator simulates a synthetic environment inhabited by interacting GBAD systems and hostile airborne targets.

The non-distributed simulator was parallelised across multiple Commodity Off the Shelf (COTS) PC nodes connected by a commercial Gigabit Ethernet infrastructure. Since model reuse was important for cost effectiveness, it was decided to reuse all the existing models, by retaining their 100Hz logical time DTSSs.

The large scale and event-based High Level Architecture (HLA), an IEEE standard for large-scale distributed simulation interoperability, had been identified as the most suitable distribution and parallelisation technology. However, two categories of risks in directly migrating to the HLA were identified. The choice was made, with motivations, to mitigate the identified risks by developing a specialised custom distributed architecture.

In this dissertation, the custom discrete time, distributed, peer-to-peer, message-passing architecture that has been built by the author in support of the parallelised simulator requirements, is described and analysed. It reports on empirical studies in regard to performance and flexibility. The architecture is shown to be a suitable and cost effective distributed simulator architecture

for supporting *a speed-up of three to four times* through parallelisation of the 100 Hz logical time DTSS. This distributed architecture is currently in use and working as expected, but results in a *parallelisation speed-up ceiling irrespective of the number of distributed processors*.

In addition, a hybrid discrete-time/discrete-event modelling approach and simulator is proposed that lowers the distributed communication and time synchronisation overhead—to improve on the scalability of the discrete time simulator—while still economically reusing the existing models. The proposed hybrid architecture was implemented and its real-time performance analysed. The hybrid architecture is found to support a parallelisation speed-up that is not bounded, but *linearly related to the number of distributed processors* up to at least the 11 processing nodes available for experimentation.

Contents

Glossary: List of Abbreviations and Definitions

Below is a list of abbreviations and often used definitions. It is intended to serve as a quick reference list.

- AFAP (As-Fast-As-Possible)
- C4I (Command, Control, Communications, Computers, Intelligence)
- COTS (Commodity Off The Shelf)
- CSIR (Council for Scientific and Industrial Research)
- CSP (Communicating Sequential Processes)
- Dead-reckoning - Also referred to as *active quantisation*. Usually applied in the quantisation of the position, velocity, etc. of a body of mass under *forced* motion.
- DEDS (Discrete Event Dynamical System)
- DESS (Differential Equation System Specification)
- DEVS (Discrete Event System Specification)
- DIS (Distributed Interactive Simulation)
- DPSS (Defence, Peace, Safety and Security)
- DTSS (Discrete Time System Specification)
- Dynamical System - The dynamical system concept is a mathematical formalization for any fixed rule which describes the time dependence of a point's (could be a real number) position in its ambient space. Examples include the mathematical models that describes the swinging of a clock pendulum, the flow of water in a pipe, and the number of fish that may be found each spring in a lake.

- Experimental Frame - A specification of the conditions—due to the modelling objectives and outcome measures—under which the system is observed or experimented with. See Section 2.2.
- FOM (Federation Object Model)
- FU (Fire Unit)
- GBAD(S) - (Ground Based Air Defence (System))
- Hybrid simulator/modelling-approach - In this dissertation the meaning of hybrid with regards to a simulator or modelling approach implies that the simulator applies a mixture of discrete time and discrete event concepts. Hybrid *does not* refer to—as is sometimes the case—a simulator or modelling approach that integrates discrete and continuous models.
- HIL (Hardware In the Loop)
- HLA (High Level Architecture)
- IEEE (Institute for Electrical and Electronic Engineers)
- LOS (Line Of Sight)
- MCM (Mathematical and Computational Modelling Research Group of the CSIR)
- Model - An executable specification of a system at a certain specification or knowledge level, within a certain experimental reference frame and to a certain degree of validaty. Discussed in Chapter 2.
- M&S (Modelling and Simulation)
- Object - An instance of a model or component of a model. Sometimes referred to as the simulator if only one model is involved in the simulation.
- OEM (Original Equipment Manufacturer)
- OIL (Operator In the Loop)
- PADS (Parallel and Distributed Simulation)
- Parallelisation Speed-Up, S.
  $S(p) = \frac{Execution\ time\ using\ single\ processor\ system}{Execution\ time\ using\ a\ multiprocessor\ with\ p\ processors}$

- PDES (Parallel Discrete Event Simulation)
- Platform - Has two meanings: Firstly, a *computing platform* or a *processing platform* generally refers to a computing machine or PC. Secondly, platform refers to a stationary or moving object in the synthetic environment which might carry a sensor, weapon or other item of interest.
- Relation - connection between things: a meaningful connection or association between two or more things, e.g. one based on the similarity or relevance of one thing to another [Encarta World English Dictionary, North American Edition].
- SANDF (South African National Defence Force)
- Scenario - In the case of the GBADS simulation, a scenario is a collection of interacting ground based air defence systems—composed of sub-systems—deployed in such a way within the environment as to defend against an expected threat scenario. A threat scenario may be a collection of incoming airborne targets such as aircraft and missiles.
- SDK (Software Development Kit)
- Scalability - maximum attainable parallelisation speed-up, S, given an infinite supply of nodes.
- Simulation - The combined input traces, output traces and object state transitions of the simulator during an execution run.
- Simulation Frame - Similar to the image frame in a cartoon strip, a simulation frame is a snapshot of the simulation and the state of its simulator at a specific instance in time. Also sometimes referred to as just a *frame*
- Simulation Time - The synthetic environment has its own notion of time i.e. the simulation time according to which the virtual inhabitants play out the action. The simulation time need not progress at the same second-to-second pace as real-time and may for example be paused or restarted as required.
- Simulator - A computing system that executes the executable specification of each model instance and couples the instances' input ports to output ports

- SOM - (Simulation Object Model)
- TCP/IP (Transport Control Protocol/Internet Protocol)
- Time Frame - Same as *Simulation Frame*
- Time Complexity - The number of steps that it takes to solve an instance of a problem as a function of the size of the problem. Model time complexity refers to the number of computational steps to update by one unit the internal time of a model instance of size n—size possibly measured by the number of interacting sub-systems
- UML (Unified Modelling Language)
- VGD (Virtual GBADS Demonstrator)

Part I

Introduction, Background and Literature

Chapter 1

Introduction

The introduction to this dissertation provides an overview of the dissertation topic, the technology involved and the applicable literature for the development of a new distributed parallel battlefield simulator. The literature includes recent work on the need for modelling and simulation within the South African National Defence Force (SANDF). The discussion of the topic background identifies certain problems in the development of the simulator and the hypothesised solution is then formulated as a research question. The dissertation road map is finally presented.

1.1 Background

The Oxford English Dictionary (1989) describes simulation as "*The technique of imitating the behaviour of some situation or system (economic, mechanical, etc.) by means of an analogous model, situation, or apparatus, either to gain information more conveniently or to train personnel*".

The SANDF requires decision support and tactical doctrine development during the different phases of their Ground-Based Air Defence System (GBADS) acquisition program to gain information more conveniently. According to Pretorius[1], Baird and Nel[2] this need offered an opportunity to establish an indigenous and credible Modelling and Simulation Support capability within the South African Defence Acquisition environment. The broad requirement of the capability is to simulate a GBADS battery of existing and still to be acquired (possibly still under development) equipment and their related human operators at a system of systems level. A pilot systems simulation capability, provided by the Mathematical and Computational Modelling (MCM) research group of the CSIR's Defence Peace Safety and Security (DPSS) operating unit, was established during 1998 to 2003 in

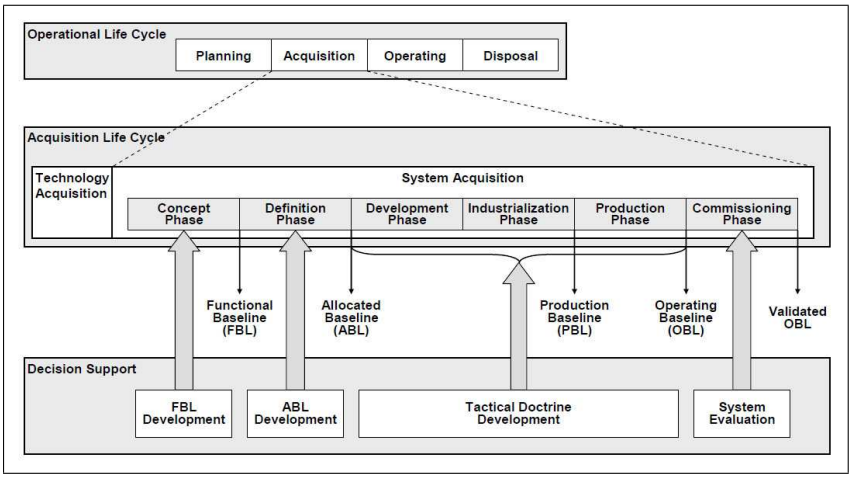

Figure 1.1: The System Life Cycle[3]

support of the concept and definition phases of the acquisition life cycle [3], which is part of the system life cycle shown in Figure 1.1. During this time the family of simulator architectures discussed by le Roux[4], also known as the Virtual GBADS Demonstrator (VGD), providing the systems simulation capability evolved as the value of modelling and simulation became better understood and as the project requirements also evolved. A GBADS deployment, such as shown in the map view scenario planning tool in Figure 1.2, consists of a layered air defence. The outer layer typically consists of eight very short range missile systems, each having a virtual operator and an accompanying buddy with a wide angle pair of binoculars. The second layer has four gun systems, each consisting of two guns, a tracking radar, a designation radar, a fire control system and at least three operators to operate the guns. The inner layer of defence usually has two short range missile systems, each consisting of a ground based launcher, a designation sensor, a fire control system and a few virtual operators. The deployment defends some Vulnerable Point (VP) against an airborne threat scenario—the GBAD system is discussed in more detail in Chapter 5. A selection of the models was derived from high fidelity engineering models, some developed by OEMs, that were based on direct numerical method solutions. The models were then developed further within a 100Hz logical Discrete Time System Specification (DTSS) that simplified the time and causality management.

The 2003 version of the architecture had its roots set in the High Level Architecture (HLA), an IEEE standard for large scale distributed simulation interoperability, but finally settled on a single process DTSS simulator architecture with a TCP message passing interface to the outside world. The

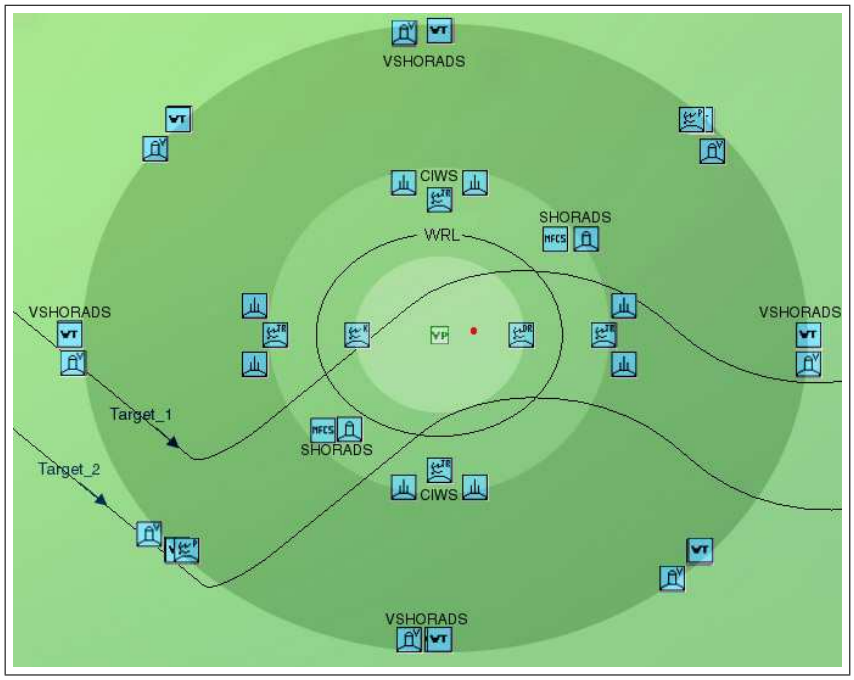

Figure 1.2: A Typical GBADS Deployment

tightly integrated single process architecture is, in a way, the HLA's archnemesis, but the history leading up to its adoption is explained below.

The HLA is introduced in the book by Kuhl et al.[5]. It is a software architecture and development process for creating distributed computer simulations out of component objects or other simulations, while ensuring component reuse and simulation interoperability of HLA compliant simulations as defined in the IEEE standard 1516[6][7]. In the late 1990s, the United States (US) Department of Defence (DoD) developed the HLA and mandated its use for all of its modelling and simulation activities. According to Page and Smith[8], the idea of interconnecting distributed simulation began to take shape in the mid 1970s and Straßburger[9] states that from a corporate perspective the demand for the HLA is a very clear business case in minimising duplication of effort and reducing expenditure. The design goals of the HLA standard are that resulting systems should have the following characteristics:

1. it should be possible to decompose a large simulation into smaller parts that are easier to define, build correctly and verify,

2. it should be possible to combine the resulting smaller simulations into a larger simulation system,

3. it should be possible to combine the smaller simulations with other, perhaps unanticipated simulations to form a new simulation system,

4. those components that are generic to component-based simulation systems should be separable from specific simulations and reusable from one simulation system to the next, and

5. the interfaces between the simulations and the generic infrastructure should insulate the simulations from the changes in the technology used to implement the infrastructure, and insulate the infrastructure from technology in the simulations.

The key characteristics of the HLA are:

1. the HLA is a layered architecture, each layer providing services to the layer above it and serving as a client to the layer below it,

2. the HLA is a data abstraction architecture, where the architecture's infrastructure is unaffected by the changes in the simulation and *vice versa*, and

3. the HLA is an event based architecture, where a simulation component broadcasts one or more events, which other interested simulation components may associate with a procedure or function to execute on receiving the message.

As mentioned, one of the earlier simulation architectures that was in use in 2002 did implement a logical time DTSS simulator within the HLA. This was done to help promote the HLA in South-Africa and get in line with international trends in the military modelling and simulation community in terms of simulation interoperability. This architecture, shown in Figure 1.3, supported distributed parallel small-scale real-time and as-fast-as-possible simulation using an in-house HLA SDK. As a capability demonstration, the architecture was integrated with a commercial flight simulator to provide realistic reactive human behaviour to the simulation. During the definition phase of the acquisition life cycle (around the beginning of 2003), accurate time-line analysis and Monte-Carlo experiments became increasingly important while supporting real-time execution, as the simulation capability evolved, lowered in priority. The unfortunate lack of a near-future integration requirement with other local simulators and the fact that pre-recorded repeatable flight profiles, requiring no run-time human interaction, had to be used for the Monte-Carlo type batch experiments, also made supporting the HLA for current and new models an unnecessary expense.

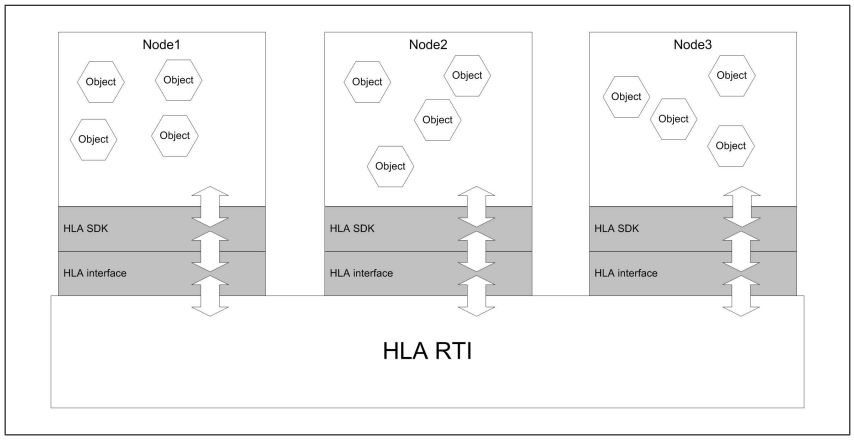

Figure 1.3: The High Level Architecture

To meet project time-line constraints, the statistical analysis results of the simulation runs had to be available for analysis in a timely fashion. This meant doing the batch runs of simulations in as an efficient manner as possible. It was proposed that the simulation be stripped of the HLA interface and HLA SDK code in an effort to remove from the architecture the overhead associated with the unused features and services of the HLA. This allows an undistributed simulator to, for example, execute one simulation job in a much shorter time than it would take an eight-machine-cluster to execute eight distributed (each job decomposed across all machines) jobs. Eight non-distributed simulators is therefore more efficient in executing many batch jobs—also referred to as *replicated trials*[10]—than an eight-machine-cluster spending all resources on one job at a time.

The resulting architecture, shown in Figure 1.4, employed a simple non-distributed *logical and conservative* (as opposed to *real-time and optimistic*, described further in Chapter 2) discrete time management scheme within a single process to interface the simulation objects directly with each other through their C++ interfaces. A TCP message passing interface served as the communication mechanism to an external 2D viewer and a human behavioural component.

In the beginning of 2004, entering the development phase of the acquisition life cycle, renewed interest in real-time simulation execution developed. This was due to a growing realisation of the positive impact of *realistic* human-simulation interaction when doing tactical doctrine development. Human interaction became a prioritised requirement and was to happen through an Operator In the Loop (OIL) console. The possibility of then recording the operator's actions to be re-used in statistical simulation runs when and as re-

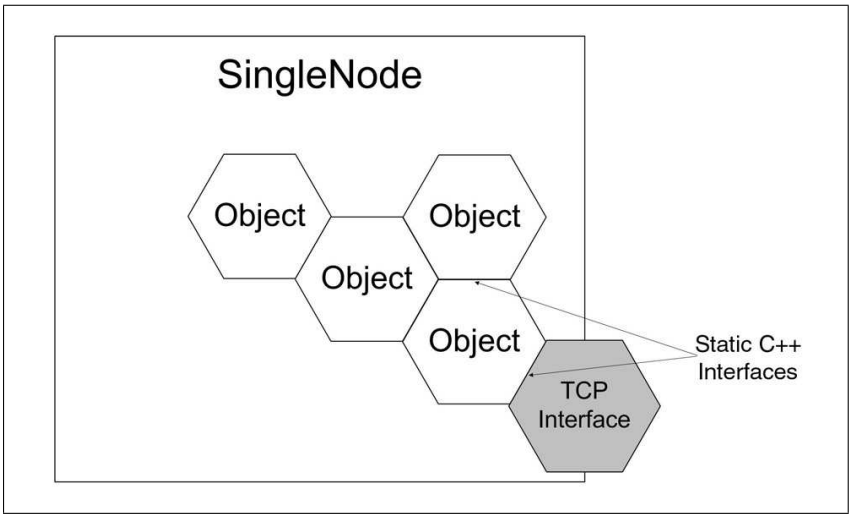

Figure 1.4: Tightly Integrated Non-Distributed Architecture

quired also existed. Within the MCM development team, the author had the responsibility of developing and implementing the distributed parallel simulator that is currently in use and capable of distributing the computational load across multiple PC's. The simulator is capable of reaching at least soft real-time performance through parallel execution. Reusing as many components as possible, including the C++ models and their 100Hz logical discrete time management, from the previous simulation was a necessity for cost effectivity. An important related requirement that impacted the hardware infrastructure supporting the simulator architecture was that the simulator must be transportable to field deployments. This requirement necessitated a dedicated infrastructure and the cost implications of new hardware limited the technologies that may be employed in the construction of the hardware infrastructure.

The design goals and required characteristics of the new discrete time simulator architecture were very similar to those mentioned above in relation to the HLA. Given this similarity and the international trends in distributed military simulator architectures, the HLA was identified as the ideal candidate for the distributed simulation capability. The HLA is, however, a discrete event architecture which promotes the use of Discrete Event System Specification (DEVS) modelling formalisms for efficient execution. From experience in applying the HLA and from documented case studies and other literature on embedding a DTSS within an interoperability standard, some technical and economic viability risks were identified in directly migrating to the HLA. The risks are discussed in Chapter 3. An alternative architecture

which would mitigate the migration risks had to be found. Even though the new architecture was not to be HLA, it nevertheless had to be a flexible, distributed, parallel simulator architecture.

1.2 Introduction to the Literature

Considerable work is currently being done on the need for modelling and simulation within the SANDF's GBADS and Joint Air Defence System (JADS) domain. Research feedback is being presented to, among others, the South African Joint Air Defence Symposium (SAJADS)[11], the European[4] and Fall[3][12] Simulation Interoperability Workshops(SIWs), and the European Air Defence Symposiums[13][2]. The article by Roodt et al.[14] looks forward—past GBAD and JADS—to what a system-of-systems simulation architecture for command & control at the joint operations level should look like.

However, to develop, instantiate and further research the simulation capability to address these needs and even just to form a shared vision of, and communicate about the required capability, necessitates the study of a set of core disciplines.

The theory of modelling and simulation is the first of the set of core disciplines and is discussed by Zeigler et al. [15] in an authoritive textbook style. Systems theory is introduced as the mathematical formalism for specifying and understanding dynamical systems. System specification formalisms, of which the DEVS modelling formalism and the DTSS modelling formalism are two, are discussed in great detail by Zeigler et al. A summary of the applicable literature, theory, and principles and practice of modelling and simulation may be found in Chapter 2. A framework for modelling and simulation is discussed and shown to contain a source system, an experimental frame, a system model and a simulator. Modelling formalisms are then used to specify the different classes of system models. The simulator's architecture is dependent on the above mentioned aspects, but also on the target computational infrastructure. Section 2.4 discusses distributed simulation and the basic hardware infrastructures. The modelling formalism and time management approach most suitable to efficient distributed simulation is identified to be the DEVS modelling formalism and optimistic time management. Chapter 16 of Zeigler et al. provides a very good and comprehensive finale on the *"DEVS Representation of Systems"*.

Kuhl et al.'s book [5] provides a simulation builders view of the HLA and Chapter 2 concludes by elaborating on the brief introduction to the HLA given earlier. The referenced suite of IEEE 1516 standards that further detail the HLA and some recommended practices in its use may be found in

[6].

Chapter 3 exposes the reader to:

- The MCM group's previous experience with the HLA,
- case studies on the use of the HLA and similar architectures in high resolution and logical discrete time management applications,
- a PhD (published as a book) by Straßburger on advances in simulation, and
- an article by Taylor et al.[16] on the potentials and pitfalls of distributed simulation.

Two categories of risks in migrating to a standardised discrete event interoperability architecture such as HLA, namely technical and economic viability, are then derived from these sources and the implications discussed.

The notations that are used to further describe and present the system specifications, their simulators and the proposed architectures are Communicating Sequential Processes (CSP), presented by Roscoe in [17], and the Unified Modelling Language (UML), presented in [18]. Chapter 4 discusses CSP and UML for use in the description and visual representation of the system specifications and simulator architectures.

Within the above mentioned literature chapters, in particular Chapter 3, a gap is shown to exists between a single process high resolution DTSS simulator and a large scale parallelisation of the high resolution logical time DTSS. A DEVS is desirable for its documented advantages in building an efficient distributed simulator. A DTSS simply embedded within a discrete event interoperability architecture is however shown to be possible, but not efficient because the high resolution logical discrete time system has a high communication overhead between distributed nodes. The HLA and DIS are shown from case studies to hit a discrete time simulation frame rate ceiling of 25Hz-30Hz. Part II discusses the discrete time simulator that is currently in use to *fill the gap* in distributively parallelising the 100Hz logical time simulation.

1.3 Research Question and Approach

This chapter has thus far provided the background to the required simulation capability and briefly made reference to the applicable literature and the motivations behind the current custom discrete time simulator. This simulator, discussed fully in Part II of the dissertation, is in use and working as expected.

However, the limits in the architecture's continued and future use needed to be determined, specifically in terms of scalability. From the HLA and DIS case studies mentioned in Chapter 3 it is expected that the high simulation frame rate is a major scalability limiting factor. This is thought to be mostly due to the sequential nature of the inter-node communication channels. In response to these research issues the scalability limits of the existing discrete time simulator is experimentally analysed and the results discussed in Part II of the dissertation.

A further research question is whether the simulator's scalability can be improved by somehow following a different modelling approach instead of the current discrete time approach? And, if so, can this be done with the same model reuse economies as was required from the discrete time simulator?

Mention has been made of the large scale and event-based HLA interoperability standard and of the success of the DEVS modelling approach. An investigation has been carried out into whether a DEVS modelling approach may somehow be married to the current discrete time simulator to improve its scalability. A hybrid discrete-event/discrete-time simulator is therefore proposed, discussed and its performance analysed in Part III of the dissertation.

To summarise, this dissertation will follow a research approach of:

- Doing the relevant background study and, in doing so, describing the current distributed parallel simulator within the newly explored—but well known within the international military simulation community—body of knowledge,
- experimenting with, and analysing the current discrete time simulator,
- proposing and implementing a hybrid discrete-event/discrete-time simulator, and
- experimenting with, and analysing the proposed architecture, leading to various conclusions and future work proposals.

1.4 Dissertation Roadmap

The starting point for this dissertation is a requirement for a simulation capability within an air-defence context. The dissertation discusses the appropriate literature and the analysis of the requirements, design and implementation of the simulation architecture to support the capability as described.

The dissertation is laid out in three parts and a conclusion as follows:

Part 1: Introduction, Background and Literature. Readers that have read the introduction and are familiar with the mentioned core disciplines and the SANDF GBADS domain may skip the remaining chapters of this part.

- The introduction to this dissertation has thus far provided an overview of the dissertation topic, the technology involved and the applicable literature. A gap is shown to exists between a single process high resolution DTSS simulator and a large scale parallelisation of the simlator. An allusion is made to the specialised discrete time simulator that is currently implemented and discussed later in Part II. The future scalability of the discrete time simulator is under question in this dissertation and the hypothesised scalability *solution* is formulated as a research question.

- Chapter 2 gives an introduction to the principles and practice of modelling and simulation. An understanding of the theory, relevant concepts and a common M&S framework is required to generate a shared vision of the simulation capability between the parties involved such as system engineers, users and developers. Setting the description of the current discrete time simulator within a commonly known and used framework also allows and is absolutely necessary for applying existing theorems and corollaries in further research. The chapter leads up to and then does a proper introduction of the DTSS, the DEVS and, of importance for this dissertation, the advantages, in terms of the simulator, of using the DEVS modelling formalism for the specification of the system to be simulated. Basic distributed hardware infrastructures and the HLA are also discussed in more detail.

- Chapter 3 researches the risks involved in migrating from a specialised discrete time simulator to a DEVS and discrete event simulator. Once the migration risks are understood, an educated decision may be made on the way forward in migrating to a discrete event simulator.

- Chapter 4 gives an introduction to the notations used to formally describe and visually present the simulator processes. This is given to the level required for the analysis, design and implementation of the respective simulator architectures.

- Chapter 5 returns the focus to the GBAD domain. The intention is to now build a clear picture of the GBAD system of systems (of subsystems) such that the simulator may be properly analysed in terms of the system model scale. Representative scenarios are identified for use in

analysing the performance of both the current discrete time simulator and the proposed hybrid DEVS/DTTS simulator.

Part 2: The Current Discrete Time Simulator

- Chapter 6 describes the layered simulator architecture. The publish-subscribe simulation model, the Synthetic Environment within which the model instances interact with each other and the lower level messaging are described. The GBADS simulation object model within which the GBADS simulation is implemented is then described.
- Chapter 7 analyses the simulation model and discrete time architecture performance results.

Part 3: Migrating to a Hybrid Discrete-Event/Discrete-Time Modelling Approach and Simulator

- Chapter 8 looks at applying a hybrid discrete-event/discrete-time modelling approach to increase the scalability of the simulator.
- Chapter 9 analyses the new hybrid modelling approach and simulator, and then presents the comparative performance results.

This dissertation thus, from existing research, builds up to a detailed description of the design and implementation of a new discrete event simulator and then subsequently finds resolution in the critique of the architecture as improving on the scalability of the current discrete time simulator. Chapters 10 and 11 respectively provide the conclusion to the dissertation and discuss the planned and potential future work. Chapter 12 contains a self examination of the dissertation.

Chapter 2

Modelling and Simulation, Principles and Practice

This chapter gives an introduction to modelling and simulation as an understanding of the theory, relevant concepts and technology involved is required to properly understand, specify and implement a simulation capability. As mentioned in the introduction, setting the description of the current simulator architecture within a commonly known and used framework also allows and is absolutely necessary for applying existing theorems and corollaries in doing further research.

The book by Zeigler et al. [15] introduces key concepts that underlie the theory, principles and practice of modelling and simulation. Although there are many references to most of these concepts in the literature, [19, 20, 21, 22, 23, 24, 10, 25] among others, the book by Zeigler, et al. is used as the authorative textbook reference and often referred to as only *Zeigler, et al.* in this dissertation. The most basic concept is that of mathematical systems theory. The theory provides a fundamental, rigorous mathematical formalism for representing dynamical systems. Only once it is understood how these dynamical systems may be represented can their simulators be built. The two main and orthogonal aspects to the theory are:

- Levels of system specification—These are differentiated levels of knowledge at which a system may be known and at which system behaviour can be described.
- System specification formalisms—These are the modelling formalisms that modellers can use to specify the models of dynamical systems.

Systems theory therefore distinguishes between the system's structure and its behaviour. An important structural and system specification concept

is that of decomposition and composition. These indicate how a system may be broken down into component systems and how component systems may be coupled together to form a larger system. The system specification formalisms allow the specification of the structure and behaviour of dynamical systems at various levels of knowledge and system specification—as discussed in Section 2.3.1. Subclasses of dynamical systems such as Discrete Event Dynamical Systems (DEDS) and Discrete Time Dynamical Systems (DTDS) may however be defined with each having their own unique system specification formalism. Essential to distributed simulation, is the fact that systems theory—and each subclass of dynamical systems—is closed under composition. This allows hierarchical composition of systems with well defined structure and behaviour. The theme of atomic versus composed systems is revisited throughout the chapter.

The rest of this chapter does, however, not discuss the mathematics of systems theory and system specification formalisms in depth. Readers who are interested in this aspect are referred to [26][27][19][15]. This chapter rather attempts to establish a basic M&S framework based on systems theory within which the current discrete time simulation capability may be described and researched, and within which the SANDF, modellers and simulator builders can effectively communicate and align their future modelling and simulation efforts.

The levels of system knowledge and specification are introduced, followed as promised by a M&S framework that is well known within the international military simulation community. The system specification formalisms are introduced to provide insight into the different types of model specifications and their simulators. The focus is then narrowed slightly to parallel and distributed simulation and the simulators' architectural intricacies. The links to the HLA, which is a discrete event interoperability architecture, are also covered here.

2.1 Levels of System Knowledge and Specification

The difference between decomposed and non-decomposed systems may be phrased, within the M&S context, in terms of levels of system knowledge as discussed by Klir[28] and levels of system specification as discussed by Zeigler et al.[15]. According to these levels the decomposed system is at a higher level of knowledge and specification than the undecomposed system since more information is provided about the structure of the system.

Table 2.1: Klir's Levels of System Knowledge[15]

Level	Name	What is known at this level
3	Structure	Components (at lower levels) coupled together to form a generative system
2	Generative	Means to generate data in a data system
1	Data	Data collected from a source system
0	Source	What variables to measure and how to observe them

The four basic levels of system knowledge recognised by Klir is shown in Table 2.1. At each level some important things that were not know at lower levels become known. The *source level* identifies a portion of the world that is going to be modelled and how to observe it. The *data level* is the set of observations made for the source system. The *generative level* contains the means to generate the same data again which is knowledge that did not exist in the data level, for example. The concepts identified at this level are usually referred to as *models* of the system. At the top level, *structure level*, a special case generative system exists in which it is known how to generate the data observed at Level 1 through the interactions of the identified system components. These components are chosen to reflect what is believed to be the system's real underlying structure that generates (or leads to) the observed behaviour. This is again knowledge that possibly did not exist at the lower knowledge levels.

Klir's framework is useful in the sense of providing perspective on what is usually considered to be distinct concepts. Klir also reasons that, when moving to a lower level no new knowledge is generated, but what is implicit in the description already available is only made explicit. The reverse applies when moving to higher levels and is referred to as inferring new knowledge. There are therefore only three basic kinds of problems dealing with systems: systems analysis, systems inference and systems design. In *systems analysis*, the goal is to understand the behaviour of an existing or hypothetical system based on its known structure, moving from higher to lower knowledge levels. *Systems inference* tries to guess a black-box system's structure from observations, moving from lower to higher knowledge levels. Thirdly, in *systems design*, alternative structures for instantiating a certain system specified at the data level is investigated, in effect moving from lower to higher knowledge

levels.

The levels (hierarchy) of systems specifications formulated by Zeigler include the concept of a dynamical system, and therefore time, which is more oriented towards the M&S context than Klir's knowledge levels. The systems are also viewed as having defined input ports and output ports through which they interact with other systems in a modular way. A system receives stimuli ordered in time—an input trajectory—on each of its input 'ports' and places a time-indexed response—an output trajectory—on each of its output ports. The system specification hierarchy, shown in Table 2.2, has an *observation frame* specification, *I/O behaviour* and *I/O function* specifications, a *state transition* specification and a *coupled component* specification. Table 2.2 also shows to which Klir level of system knowledge each of the system specifications correspond.

The *Observation Frame* specifies how to stimulate the system, which variables to measure and how to observe them against a time base. The decisions on what observation frame to use is dependent on the modelling choices made. These modelling choices are specified by the choice of an experimental reference frame which is explained in Section 2.2:*Framework for Modelling and Simulation*.

All the *observed* input trajectories along with their associated output trajectories are called the *IO Behaviour* of a system. If a method exists by which the IO behaviour of a system can be predicted, knowledge at the next level of the system, the *IO Function*, would have been attained. The IO function includes knowledge of the initial state and can determine a unique output for every input stimulus.

At the *State Transition* level of system specification, knowledge is gained of the system's (as a whole) state transitions as it responds to the input trajectories. The next level of system specification, *Coupled Components*, opens up the system to reveal more than just the system's aggregated state. At this specification level the system is composed of interacting components. The components may in turn be specified at levels 1 to 3 (or even at level 4) and are coupled using their input and output ports.

2.2 Framework for Modelling and Simulation

The framework presented in this subsection is done so that everyone involved in a simulation exercise—model developers, simulator developers and users alike—may use it to communicate effectively with each other and the wider military simulation community, and to align their research and development efforts. The system specification hierarchy (Section 2.1) provides the basis

Table 2.2: System Specification Hierarchy[15]

Level	Specification Name	Corrosponds to Klir's	What is known at this level
4	Coupled component	Structure system	Components and how they are coupled together. The components can be specified at lower levels or can even be structure systems themselves - leading to hierarchical structure.
3	State transition	Generative system	How states are affected by inputs; given a state and an input what is the state after the input stimulus is over; what output event is generated by a state.
2	I/O function		Knowledge of initial state; given an initial state, every input stimulus produces a unique output.
1	I/O behaviour	Data system	Time-indexed data collected from a source system; consists of input/output pairs.
0	Observation frame	Source system	How to stimulate the system with inputs; what variables to measure and how to observe them over a time base.

for the framework. The entities of the framework are introduced in the subsection below, followed by the relationships between them in a subsequent subsection.

The next subsection describing the framework then briefly introduces concepts related to model characterisation, validation, verification and qualification and links them to the framework. Finally, the different aspects of simulation time management and a classification for military simulation types are discussed.

2.2.1 Entities of framework

The basic entities of the framework, shown in Figure 2.1, are the *source system*, the *experimental frame*, the *model* and the *simulator*. The basic relationships between the entities, also shown, are the modelling and simulation

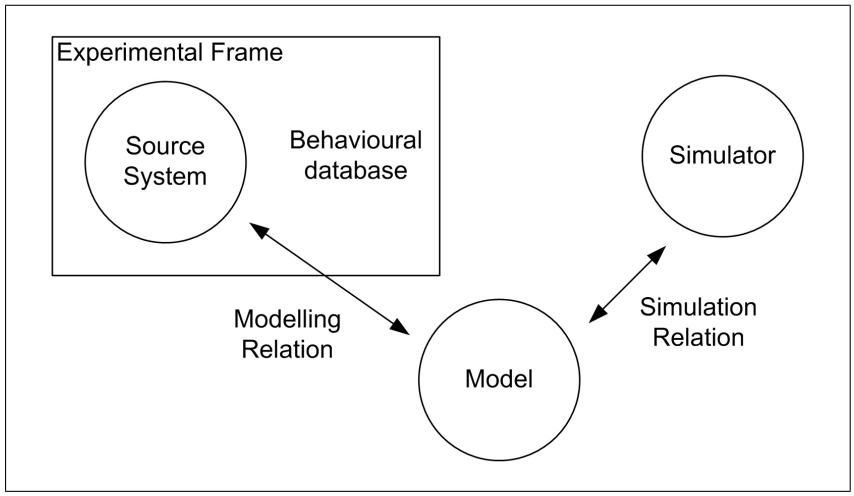

Figure 2.1: The basic entities of an M&S framework and their relationships[15]

relationships.

The *Source System* is the environment that is of interest to the modeller; the source of the observable data. The system *behavioural database* contains the data that has been gathered by experimenting and observing the source system. The source system corresponds to level 0 of Klir's hierarchy, and the system behavioural data base corresponds to level 1 of Klir's hierarchy (the data system).

The behavioural data base is acquired within an *Experimental Frame* of interest to the modeller. The experimental frame is a specification of the conditions under which the system is observed or experimented with. Many experimental frames may be defined for the same source system and a single experimental frame may apply to many systems just as the same system may be modelled with respect to different objectives and different systems may be modelled with respect to the same objective.

The statement of objectives serves to focus model construction on particular issues. Such a statement should be formulated as early as possible in the model development process so that parties involved may agree on their goals and align their efforts. These objectives may then be translated into more precise experimentation conditions for the source system or its models. The result of this translation, outlined in Figure 2.2, is the experimental frame.

The modelling objectives typically concern system design, in which case the *outcome measures* are measures of effectiveness in evaluating the system's design alternatives. In air defence system simulations, outcome measures are typically 'the number of threats successfully engaged before they could reach

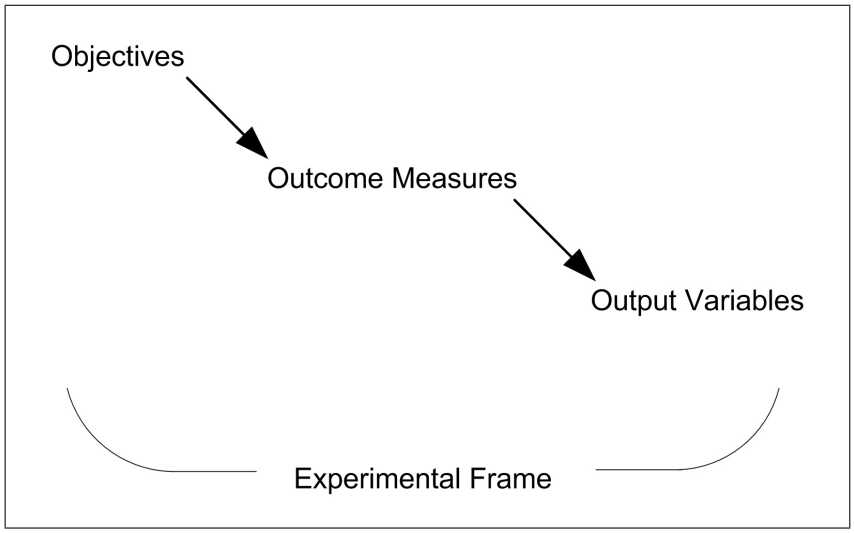

Figure 2.2: Transforming objectives into experimental frames[15]

their weapon release line' or possibly 'the percentage hits of a particular weapon system'. In order to compute the relevant outcome measures the appropriate output variables must be defined, which in turn drive the modelling effort within the specific experimental frame.

As suggested, a *Model* may be defined as a system specification at any of the levels in Table 2.2. In the traditional context of M&S, though, the system specification is usually done at levels 3 and 4, corresponding to Klir's generative and structure levels. These models are typically instructions or equations for generation of IO behaviour. A typical Air Defence System consists of a number of subsystem models specified at level 3 which are coupled together to specify more complex system structure models at level 4. The various system models may then be coupled together to specify a more complex system of systems structure model, also at level 4.

As sets of instructions or equations, the model instances require a *third party* to execute the instructions or to evaluate the equations and to manage the coupling of output ports to input ports. This third party 'computing system' is the *Simulator*. The simulator is the final entity of the M&S framework and the focus of this dissertation. According to Zeigler et al., separating the model and simulator concepts provides a number of benefits for the framework, namely:

- The same model, expressed in a formalism, may be executed by different simulators, thus opening the way for portability and interoperability at a high level of abstraction, and

- simulator algorithms for the various formalisms may be formulated and their correctness rigorously established.

2.2.2 Relationships among entities

According to Zeigler et al., the entities—system, experimental frame, model, simulator—become truly significant only when properly related to each other. As was seen in the previous section, a model of a system is built within a certain experimental frame according to certain objectives and it is critical for the success of the simulation that certain relationships hold. The two most fundamental are the *Modelling* and *Simulation* relations, also shown in Figure 2.1, as these ensure the validity of the model and the simulator's correctness respectively. A third relation that is important for understanding modelling and simulation work is *Modelling as Valid Simplification*. These three relations are now discussed.

Modelling Relation

As shown, in Figure 2.1, the *Modelling Relation* refers to a relation between a system, the experimental frame and the model. In other words, the modelling relation, in effect *model validity*, describes to what degree a model faithfully represents the system being modelled within the experimental frame chosen. The degrees of validity that are differentiated here are:

- Replicative validity
- Predictive validity
- Structural validity

Replicative validity holds if, for all the experiments possible within the chosen experimental frame, the IO behaviour (observed input/output trajectories) of the model and system agree to within an acceptable tolerance. Replicative validity, therefore, holds if the model and the system are equivalent at a system specification level of 1. *Predictive validity* is a stronger form of validity which requires not only replicative validity, but also the means to predict behaviour not observed yet. This requires that the model and system be equivalent at the next system specification level of 2 which is on an IO function level.

Structural validity is the strongest of the degrees of validity differentiated here. As the name implies it "looks inside" the system and requires that the model and the system be equivalent at a system specification level of 3 (or

4). This allows the model to not only replicate or/and predict the system's behaviour, but to mimic on a system level (or sub-system/component level) the state transitions of the system.

The term fidelity is often used and it refers to combination of both model validity and model detail. Detail refers to the depth in terms of the number of output variables that the experimental frame requires the system to be probed and is therefore highly dependent on the modelling objectives. It is important to note, though, that high detail alone does not imply high fidelity. High fidelity also requires the modelling relation to hold to the appropriate degree.

Simulation Relation

As shown, in Figure 2.1, the *Simulation Relation* is a relation between the simulator entity and the model entity of the M&S framework being described. According to Zeigler, a simulator correctly simulates an instance of a model, if, given the object's initial state and input trajectory, the simulator generates the expected output trajectory. Simulator correctness thus requires agreement at, at least, the IO function specification level. If, however, the model was specified at a state transition or coupled component system specification level then, depending on the modellers interest in the detail, the simulator correctness might require agreement at the system state transition or the coupled component levels respectively.

Modelling As Valid Simplification

The modeller should constantly keep in mind the limited resources of the currently existing simulators. A model might have to be simplified to be simulated within a reasonable time period. The simplified model must still represent the modelled system to the required degree of validity within the chosen experimental frame, though. The detailed (valid for a large set of experimental frames) model is referred to as a *Base Model* and the simplified model as a *Lumped Model*. The lumped model is typically valid for a small set of experimental frames or a single experimental frame, but it is important to note that the lumped model is just as valid within these specific experimental frames as the base model. In Chapter 13 of [15], Zeigler et al. discuss the concept of a morphism for judging the equivalence of base and lumped models with respect to a specific experimental frame, and how to construct such morphisms.

2.2.3 Model Characterisation, Validation, Verification and Qualification

This subsection briefly deals with the *error* in the relations between the entities in the M&S framework. Verification and validation concepts from Zeigler et al.[15] are discussed to address the error in the simulator and model respectively. These concepts are then integrated with the characterisation, validation, verification and qualification cyclical modelling process used by Sargent[29][30] which has already been studied within the GBADS modelling domain.

Zeigler et al. discuss verification as the attempt to test that the simulation relation (discussed earlier) holds between a simulator and a model. The simulation relation in effect establishes error free simulator correctness while verification is the process to evaluate the errors introduced by the process of building the simulator.

Validation, in turn, attempts to test whether the modelling relation (model validity) holds. The modelling relation establishes that an error free model faithfully represents reality to a certain degree of validity within the chosen experimental frame while the validation effort evaluates the faithfulness of the representation taking into account the errors introduced during the modelling process.

Model verification and validation, along with model characterisation and qualification, is a significant part of the modelling process from beginning to end. The modelling process, used by Sargent[29][30], shown in Figure 2.3 is built around these aspects. The process has already been investigated within DPSS's M&S team by Roodt[31] and is reiterated specifically within the GBADS and M&S support domain by Pretorius[1].

The qualification aspect of the modelling process makes the cyclic link between Figure 2.1 and Figure 2.2 taken from Zeigler et al. During the modelling process qualification in effect evaluates whether the chosen experimental frame represents reality in such a way that the outcome measures, and therefore the objectives, are satisfied.

The specific verification and validation techniques and processes are not discussed in this dissertation. The interested reader is referred to [29], in which Sargent continues with validation techniques for each of the above mentioned aspects, and also [31], in which Roodt continues to propose a verification, validation and accreditation (VV&A) process. The viewpoint in the above two references seems to be that model validity may be inferred when there is *failure to reject the model*. Zeigler at al.[15] however follow a more formal approach and attempt to prove validity and simulator correctness in Chapters 12, 13 and 14 of their book.

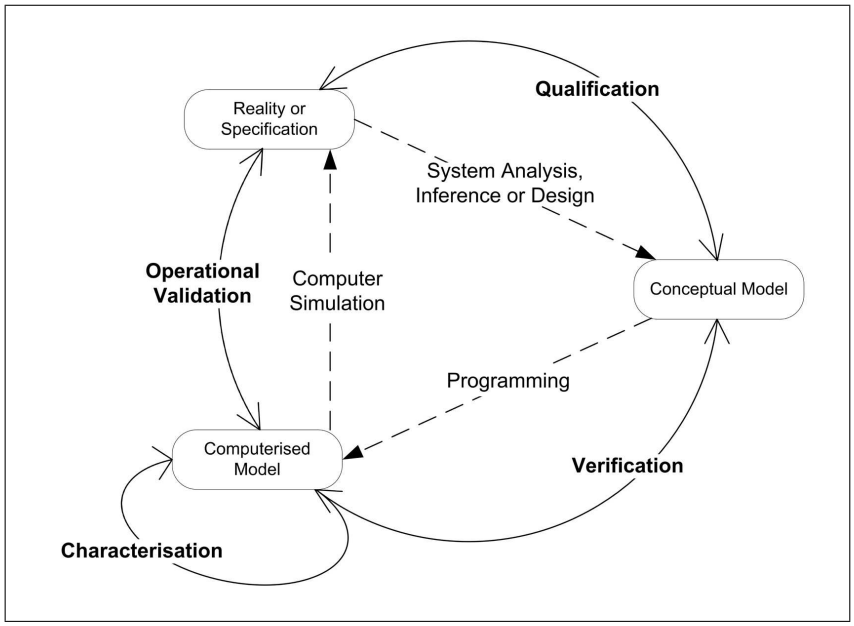

Figure 2.3: Model characterisation, validation, verification and qualification within the simplified modelling process (Adapted from [31])

Accreditation is also referred to as certification of a model. This is usually done by a third party (accreditation agent), possibly making use of subject matter experts. According to Roodt[31], the accreditation agent's final assessment may result in:

- Full accreditation—-simulation produces results that are sufficiently credible to support the application,
- limited or conditional accreditation—constraints are placed on how the simulation can be used to support the application,
- modification of the simulation is needed—simulation capabilities are insufficient to support an accreditation decision which requires modifications and subsequent V&V to correct,
- additional V&V information is needed—requires supplemental V&V, and
- no accreditation - the simulation is not fit to support the application.

Sargent [29] is of the opinion that accreditation is most viable (due to cost) when done in parallel with the model development process such that it facilitates the appropriate quality and quantity of V&V efforts during development.

2.2.4 A Taxonomy for Classifying Military Simulation Types

The taxonomy most often used for classifying the different military simulation types is described by Page and Smith[8]. This live, virtual and constructive classification scheme originated from the US DOD during their early simulation efforts. The three classes are officially described as:

- *live simulation* refers to a real simulation involving real people operating real systems.
- *virtual simulation* refers to a simulation involving real people operating simulated systems. Virtual simulations inject human-in-the-loop in a central role by exercising motor control skills (e.g. flying an aeroplane), decision skills (e.g. committing fire control resources to action), or communication skills (e.g. as members of a C4I team).
- *constructive simulation* refers to a simulation that involves simulated people operating in simulated systems. Real people stimulate (make inputs to) such simulations, but are not involved in determining the outcomes.

The simulation type taxonomy aids the people involved in creating a common understanding of the typical use cases of the simulation which guides the people involved in aligning their efforts in regard to the required experimental frames, potential simulator architectures, simulation logistics, etc.

2.2.5 A Simulation Time Management Taxonomy

The synthetic environment has its own notion of time i.e. the *simulation time* according to which the virtual inhabitants play out the action. Simulations and simulators may be distinguished based on their time management strategies. *Logical time management* requires the simulation to execute every simulated clock tick, stepping the entire system from one state to the next in a deterministic manner. Logical time simulation is also known as, *As-Fast-As-Possible (AFAP)* simulation as there is no implicit synchronisation of the simulation time with a real world clock. Such a simulation takes as long as is required to simulate any given scenario.

Real-time simulation, on the other hand, actively tries to keep its simulation time synchronised with an external real world (wall) clock. A real-time simulation continually *jumps* forward in time as far as is needed to keep executing at real-time. All partaking models and external constructive and

Table 2.3: A Simulation Time Taxonomy

	Logical Time	Real-Time
Locally Managed	Type 0 (Not possible)	Type 1
Globally Managed	Type 2	Type 3

virtual components must support this *time jump* behaviour for real-time simulation to be successful. The bigger the time jumps, however, the harder it becomes to ensure causality of simulation messages and events within a certain time resolution, until at some point when either real-time execution must be sacrificed or causality is lost along with the credibility of the simulation. A logical time simulation may of course ensure, within performance limits, that its execution is throttled so that it does not exceed real-time. Even in such cases, the differences between logical time and real-time simulation remains in that the former simulation still executes a state transition on each and every simulated clock tick. A logical time simulation that cannot be simulated fast enough to keep up with real-time will fall behind the wall clock.

Time can also be managed either locally or globally across the coupled component models. Global time management implies that there is some explicit time synchronisation mechanism between the coupled components. This is not required for local time management since each component runs at a predefined execution rate. Combining the local/global time dimension with the real-time/logical time dimension results in a *Time Taxonomy* shown in Table 2.3.

A simulation of Type 0 can not be built as there is by definition no mechanism to synchronise the time between the logical time models. Logical time models execute AFAP or apply a best effort to keep to a predefined ratio of real-time. A global mechanism is required to block the execution of models that get ahead of others. Within this time taxonomy, the discrete time simulator discussed in this dissertation, and currently in use, is of type 2. It is a distributed simulator, whose processing nodes each execute AFAP (logical time) and the global synchronisation mechanism tracks node progress and pauses the execution of nodes that get ahead of the others It is worth noting that the previously mentioned simulation type taxonomy is a use case view, while the time taxonomy is an implementation view further down the development cycle of the simulator.

Table 2.3 does however not indicate the *mechanisms* of global time synchronisation. Global time management is further differentiated into conservative and optimistic global time management. Conservative time management is how we intuitively expect the coupled components to behave in that causality violations are strictly avoided. Avoiding causality violations is done by processing the input messages in strict increasing time order which is called the *local causality constraint*.

Optimistic time management, on the other hand, lets a component assume that it is safe to process the events at its inputs although future events may violate the local causality constraint. The object may, however, only operate under this optimistic assumption if it is capable of backtracking its own execution should a message that is older than its current internal time arrive at its input so that causality may be repaired when detected. Optimistic time management ensures that components do not wait unnecessarily for each other or for time advance requests from the simulator. In scenarios where this does not lead to excessive backtracking, such optimistic time management leads to improved simulation performance. Fujimoto experimentally demonstrated the improved performance of optimistic time management in [10] and Zeigler et al. further argues that under typical conditions excessive backtracking does indeed not happen.

2.3 Modelling Formalisms and System Specifications

A system specification formalism (or modelling formalism) is a notation with which to articulate the specification of a system to be modelled. Every subclass or type of system has its own modelling fomalism with which to articluate the system specification. Each system subclass implies constraints in terms of description of dynamic behaviour and time management. Zeigler et al. does also distinguish between *basic* and *coupled* system specification formalisms for the spesification of atomic and coupled component models respectively. As mentioned at the onset of this chapter, the system specification formalisms are not discussed in a formal mathematical manner and the interested reader is referred to the work by Zeigler et al. The basic system specification formalisms (basic system subclasses) are introduced to provide insight into the different types of models and their simulators that may be used to instantiate and breathe life into a simulation capability. The coupled extensions to the basic formalisms will be differentiated in the next section.

The basic DTSS is discussed followed by the basic DEVS. The Differen-

tial Equation System Specification (DESS) modelling formalism is a third basic modelling formalism discussed by Zeigler et al. The DESS modelling formalism is not explicitly required for this dissertation and therefore not discussed. Many of the existing GBADS sub-system models do however include sets of differential equations which is currently numerically analysed in discrete time models, but could be articulated using the DESS formalism.

Interconnection between a DTSS and a DEVS is then discussed and the DEVS modelling formalism is finally presented as being universal in the sense that it includes the DTSS modelling formalism, and any other, modelling formalisms. This knowledge further facilitates a common language between M&S groups and allows sensible reuse of models and interconnection of different simulation types. The fact that DTSS specified models may be interfaced to and embedded within a DEVS is also critical to this dissertation. The hybrid DEVS/DTSS simulator proposed to address the research question and discussed in Part III is built upon this knowledge.

2.3.1 General Dynamical Systems

A dynamical system has a *time base* that orders all dynamical changes. Dynamical systems may also be specified or described at the various knowledge levels or system specification levels discussed in Section 2.1. At the IO observation frame system specification level—source system knowledge level—the dynamical system my be specified as the structure $IO = (T, X, Y)$, where T is the time base, X is the input values set and Y is the output values set. At the IO behaviour system specification level—data system knowledge level—the I/O relation observation of the dynamical system is specified as the structure $IORO = (T, X, \Omega, Y, R)$, where Ω is the set of allowable input segments, R is the I/O relation and the rest are as before. The I/O relation, R, is the set of all input and corresponding output segment pairs, for every and all allowable input segments in Ω.

At the I/O function system specification level the I/O relation, R, is replaced with a set of functions $F = \{f_1, f_2, ...f_i...\}$ and the I/O function observation (IOFO) of the system is specified as the structure $IOFO = (T, X, \Omega, Y, F)$. This allows the correct output segment to be chosen from the set of output segments for a specific input segment given that the system's initial state and in effect which function f_i to use is known.

At the state transition system specification level the set of functions, F, is replaced with a set of states, Q, a global state transition function, Δ, and an output function, Λ. At this level the dynamical system may be specified as $S = (T, X, \Omega, Y, Q, \Delta, \Lambda)$. The global transition function, Δ, describes the state-to-state transition caused by the input segments. The output function,

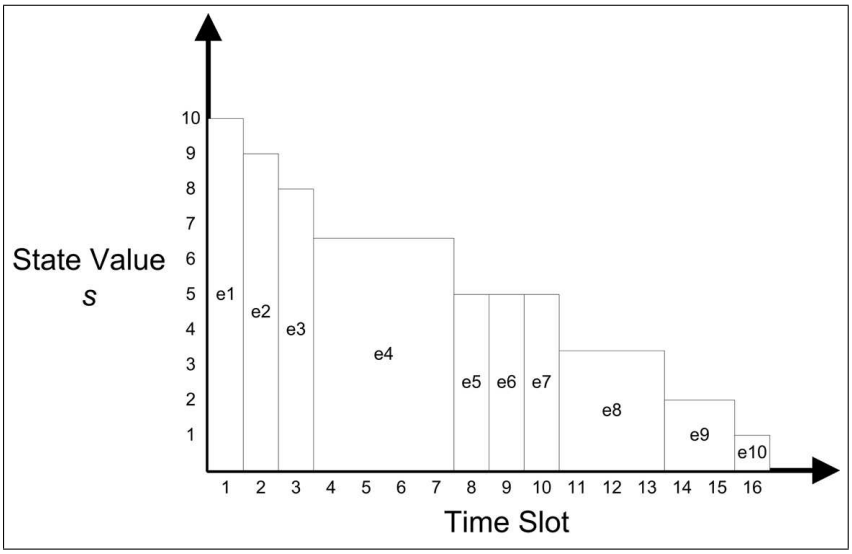

Figure 2.4: Discrete Time System Specification

Λ, describes the state-to-observable-output mapping. The output function may be a function of state and input (Mealy-type system) or of only state (Moore-type system).

The subclasses of dynamical systems represented in the next two subsections are of similar structure. Certain constraints have however been imposed on the systems. These constraints define the subclass boundaries.

2.3.2 The Discrete Time System Specification (DTSS)

The class of discrete time dynamical systems exist as snapshots at discrete instances in time. Figure 2.4 shows an output trace s represented within discrete time slots. Each e represents which state the system is in and the value is sampled at each time slot, t. With each new time instance the system's internal state may be updated based on its state history and input history up to and including the previous time instance. This offers a simple and easily understood way to enforce causal simulation progression. The time steps are generally of a fixed time length and there is an inverse correlation between the time step frequency and the *lumped* model's simplification. The Discrete Time System Specification (DTSS) formalism—known as an atomic or basic formalism—is proposed as a formalism for specifying discrete time dynamical systems.

The mathematical structure for expressing a DTSS is formally described in Zeigler et al., but the differences from a general dynamical system is min-

imal. The important difference is that the time base, T, is constrained to the set of multiples of the *time step value*. The time step value is also known as the *discrete time frame length*.

Systems specified at the coupled component level require mechanisms to synchronise the discrete time increments across the components. The stepwise execution required by discrete time systems is akin to conservative global time management previously discussed. The coupled component extensions to the basic modelling formalisms, and the different time and causality management algorithms are discussed further in Section 2.4 on distributed simulation.

2.3.3 The Discrete Event System Specification (DEVS)

In contrast to a discrete time dynamical system of which the input, state and output traces exist at globally announced—and often equally spaced—instances in time, a discrete event dynamical system exists as a time stamped history of important events and actions. It involves a mind shift from time focussed to event—a distinct value level of a trace—focus. According to Zeigler et al.[26], DEVS—also known as an atomic or basic formalism—may be viewed as a shorthand to specify dynamical systems whose input, state and output trajectories are—or may with acceptable fidelity be modelled as—piecewise constant. The step-like transitions in the trajectories are then identified as discrete events and the system as belonging to the class of discrete event dynamical system. Zeigler et al.[15] refer to the process of finding *distinct* level segments as quantisation. Figure 2.5 shows the event dimension view of the same output trace s as before. Each vertical bar represents an event and the transition of the system to a new state. The time dimension of the output variable is indicated by a t and the event dimension by an e in Figure 2.4 and Figure 2.5.

This modelling formalism therefore only specifies the intercommunication of events that are important to the simulation. This method, according to Zeigler et al., provides a good fit to the current bandwidth-to-computation ratio of modern digital computers and interconnect infrastructures. Zeigler et al. also presents a hierarchical simulation algorithm/protocol for DEVS models. Further, theoretical and experimental evidence suggests that DEVS is not only efficient in terms of communication overhead, but also model time complexity. This will be discussed further in the next section when introducing the coupled component extensions of DTSS and DEVS.

The mathematical structure for expressing a DEVS is formally described in Zeigler et al., but the differences from a general dynamical system is minimal. The important difference is, as mentioned, that the input, output and

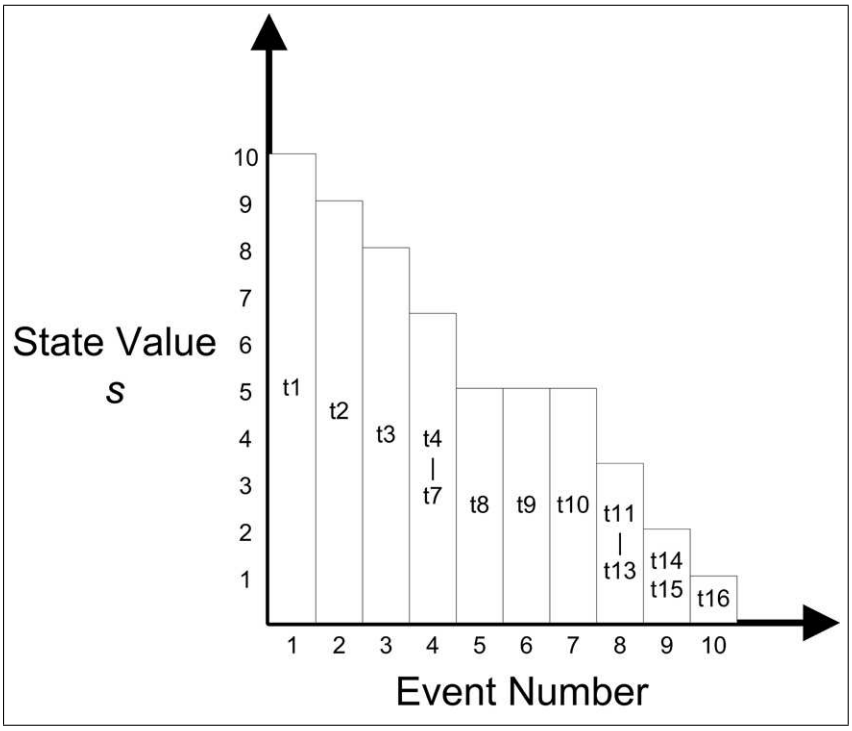

Figure 2.5: Discrete Event System Specification

state trajectories are constrained to be piecewise constant.

Systems specified at the coupled component level again require mechanisms to synchronise their internal time and execution across the components. This may be done conservatively as was the case for a DTSS, but a DEVS also more naturally supports optimistic time management. The coupled component extensions to the basic modelling formalisms, and the different time and causality management algorithms are discussed further in Section 2.4 on distributed simulation.

2.3.4 Interconnection of a DEVS and a DTSS

As already shown in Figure 2.4 and Figure 2.5, a DTSS specified model may be viewed from a discrete event dimension. Figure 2.6 completes the description of the DTSS versus the DEVS by showing the *top view* of the output trace which results from combining Figure 2.4 and Figure 2.5. A DTSS and a DEVS are therefore argued to be two different projections (viewpoints) of the same system trace.

Interconnecting a DEVS with a DTSS involves either projecting the discrete time view onto a discrete event view or vice versa. If the output of a

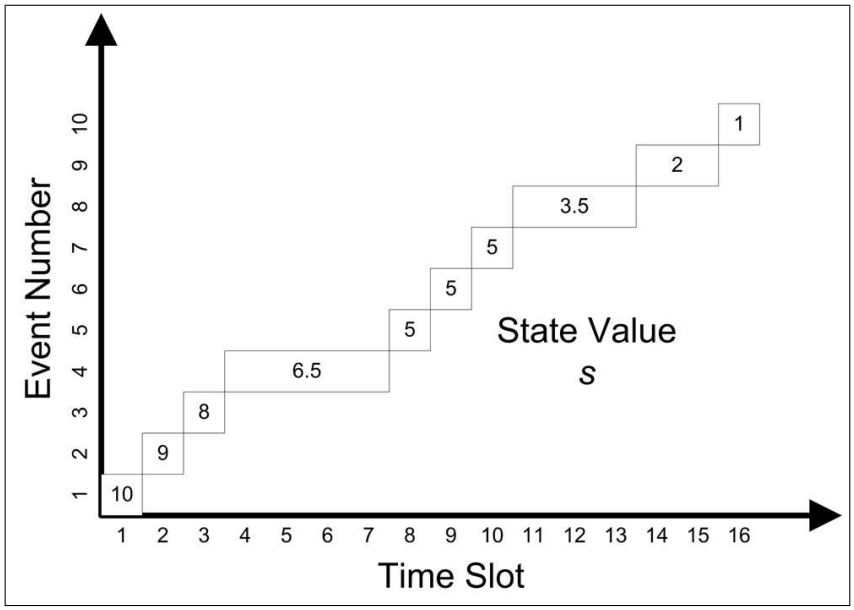

Figure 2.6: DTSS vs. DEVS

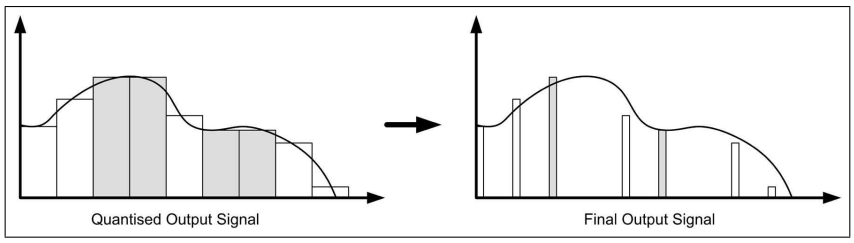

Figure 2.7: Output Signal Quantisation

DTSS system is connected to the input of a DEVS system, the DTSS system is projected to a discrete event view. The conventional DEVS approach involves generating a time stamped discrete event for each time slot trace value. This may be optimised by grouping consecutive time steps that are of similar value into a single event—a process known as quantisation or the quantised DEVS approach. Figure 2.7 shows this more clearly. The continuous output signal that have been quantised is shown on the left and the final output after combining similar output values into events are shown on the right.

If the output of a DEVS specified system is connected to the input of a DTSS specified system, the DEVS system is projected to a discrete time view. This, in turn, involves *sampling* the most recent discrete event at the time slot granularity of the DTSS system to generate time slot trace values.

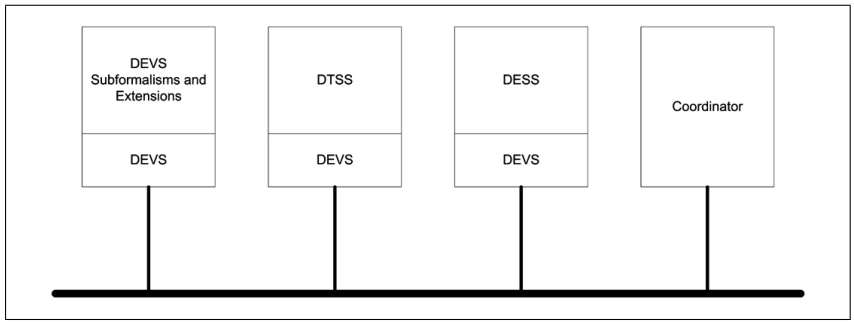

Figure 2.8: The DEVS BUS concept[15]

Care must be taken to not sample at a too large time slot granularity as short duration events may be missed.

2.3.5 Universality of the DEVS

Kim et al.[24] and Zeigler et al. introduce the *DEVS BUS* concept primarily for facilitating interoperation between diverse discrete event modelling formalisms. DEVS BUS is an encapsulation of models within DEVS *wrappers*. When modelling formalisms are mixed—for example by connecting a DTSS model to a DEVS model as mentioned in the previous subsection—it is known as multi-formalism modelling, heterogeneous simulation or embedding one modelling formalism into another. Zeigler et al. have shown that the DTSS formalism may be successfully embedded within the DEVS formalism and further that the DEVS BUS concept may be used to interoperate discrete time and discrete event system model instances as shown in Figure 2.8.

Discrete Event System Specification (DEVS) is referred to as a *universal formalism for a DEDS*[26]. The *universality* of the DEVS modelling formalism in being able to include/encapsulate other modelling formalisms provides the basis for distributed simulator architectures and reusable model implementations.

In other words, the DEVS formalism is the underlying protocol of the DEVS BUS[24]. The centralised coordinator largely corresponds to the simulator, being responsible for running the overall simulation loop—i.e. for doing the time management and for acting as the communication switch for connecting object inputs to outputs. The DEVS BUS and interoperability concepts are also extended to the DESS, but as mentioned before, the DESS modelling formalism and class of systems is discussed in this dissertation.

Kim et al. [24] goes further to define *DEVS compliance* of a model as *the model obeying the DEVS BUS protocol*—which is in effect the DEVS

hierarchical simulation protocol. The disadvantage of DEVS BUS is however 20-25% slower execution[24] *compared to DEVS*—at least when not distributed such as in DEVSim++. This is due to DEVS BUS' centralised time management- and message-controller.

A distributed simulation built using the HLA, mentioned before and discussed further at the end of the next section, may apply the DEVS BUS concepts to achieve the same type of model interoperability.

2.4 Parallel and Distributed Simulation

The previous section distinguished between basic and coupled modelling formalisms. Coupled modelling formalisms are associated with the system structure concepts of composition and decomposition. These concepts have been mentioned at the beginning of and throughout this chapter. This linkage comes to the fore when a system model needs to be simulated in a parallel and/or distributed fashion: an atomic model of the system is typically decomposed into smaller parts which are then coupled together using the appropriate modelling formalism operators. Note that distributed simulation refers to the coupling of model instances executed jointly by different processing nodes in order to capitalise on the benefits of sharing computational or model resources. According to Fujimoto[10], parallel simulation may be distinguished from distributed simulation by referring to the balance between two basic objectives. These are:

- increased simulation throughput; and
- sharing in geographically distributed simulator and model resources—which typically increases the coupling's communication latency.

The former objective is generally more strongly associated with parallel simulation, and the latter, with distributed simulation. Achieving the combined objectives simultaneously is typically technically difficult due to latency and bandwidth limitations (resulting in a time overhead) when communicating between distributed physical locations.

For the purposes of this dissertation, *parallel* simulation is however regarded as being a special case of distributed simulation—referred to as distributed parallel—geared more towards increased simulation throughput than the resource sharing that distributed simulation provides, but limited by the distributed hardware infrastructure that will be described in this section and in Chapter 7. Discussions around distributed simulation therefore include distributed parallel simulation, unless otherwise stated.

A brief overview of the coupled extensions to the DEVS and DTSS modelling formalisms that are relevant to distributed and also, more specifically, distributed parallel simulators is given below. This is followed by an introduction to the different distributed time management strategies. Distributed and distributed parallel simulation architectures and infrastructures found in the literature are also highlighted. The section ends with a more in depth discussion of the HLA.

2.4.1 DEVS and DTSS Coupled Extensions

The DEVS BUS concept has been introduced to facilitate interoperability between different modelling formalisms. However, as already mentioned, within the DTSS and DEVS formalisms there are sub-formalisms (extensions of the basic modelling formalisms) to facilitate model coupling. The coupled extensions mentioned below expand the classes of system models that can be represented in the DTSS and the DEVS formalisms to distributed and parallel systems.

The Discrete Event Specified Network (DEVN) formalism facilitates modular construction of discrete event models that are more naturally suited to distributed simulation. Using DEVN (also known as DEVS coupled model specification) components are DEVS specified systems. In DEVN, these are couple by means of their input/output ports alone. The components may be classic DEVS systems or Parallel DEVS (PDEVS) systems. PDEVS is an extension of DEVS. The major difference being that in classic DEVS, only one component is activated at any time, while in the PDEVS class of dynamical systems any number of components may be activated at the same time, allowing parallelisation of execution.

The Discrete Time Specified Network (DTSN) formalism similarly facilitates modular construction, but of discrete time models. Multicomponent DTSS (multiDTSS) is the DTSS extension that allows DTSS system decomposition and coupling. Zeigler et al. shows that multiDTSS is well suited to parallel execution. That is, the system state at time $t + 1$ may be arrived at by each component independently, and in any order, processing only its own input trace and state at time t.

As seen previously, the basic(atomic) modelling formalisms allow systems to be specified at level 3 and lower of Zeigler's system specification hierarchy. The coupled modelling formalisms informally described above, in turn, allow systems to be specified at the highest system specification level—level 4 in Table 2.2—which is the coupled component level. The modelling formalisms therefore mirror Zeigler's system specification hierarchy which allows models to be implemented at every level of system knowledge and specification. The

mathematical structure for specifying a modular coupled component network is formally described in Zeigler et al.

It is important to note here that even though multiDTSS is well suited to parallelised execution, Zeigler et al. offer a further conjecture at the end of Chapter 16 of [15]. They argue from the developed theory and case studies that, in the coupled and distributed simulation context, the efficiency of quantised DEVS is never less than that of DTSS to achieve the same accuracy. In this conjecture, efficiency is related to communication overhead (recall Figure 2.7), but model time complexity is also included in a stronger version of the conjecture. This is the main motivation within this dissertation for using the DEVS and the DEVS BUS concept in developing the new hybrid distributed simulator.

The main reason for the efficiency of quantised DEVS lies in its application of a process called *quantisation* which reduces state update transmission. A sound theoretical foundation for the quantised system approach [27][19][15] to discrete event representation of discrete time systems has been developed. Theoretical and experimental evidence, discussed in [15][23] among other literature sources, again suggests that DEVS is not only efficient in terms of communication overhead, but also computational time complexity. Nutaro et al.[23] note that potential speed advantages with DEVS are to be expected for partial differential equation systems—and in general dynamical systems—that are characterised by heterogeneity in their time and value space behaviour. Zeigler, et al.[22] also presents some nice examples of various complex adaptive systems that have been successfully and efficiently modelled by making use of this quantised discrete event approach.

The quantised DEVS concepts will be discussed further in Chapter 8 when applying them in the proposed hybrid simulator. It will be seen that DEVS uses output quantiser and quantised integrator pairs to respectively quantise and reconstruct the signal as required. The second order extrapolators used in dead-reckoning are more advanced versions of signal quantisation.

2.4.2 Distributed Time Management

Distributed time management is important for maintaining causality among all the coupled components. Such time management may be done either conservatively or optimistically, as mentioned in the Section 2.2.5. The conservative and optimistic time management approaches are discussed further below. Only conservative time management is considered in relation to *discrete-time* simulation. Both conservative and optimistic management is considered in relation to *discrete-event* simulation.

Distributed Conservative Discrete-Time Time Management

Zeigler et al., discuss a hierarchical simulator for a coupled DTSS in [15]. The peer-to-peer structure of the discrete time simulator under study in this dissertation will be discussed against this reference hierarchical simulator in Part II.

Section 2.4.1 pointed out that multiDTSS is well suited to distributed and parallel execution. The hierarchical simulator discussed by Zeigler et al. defines a simulator protocol to accomplish the time management and coupling of components required for multiDTSS. This simulator protocol is discussed briefly here. A coordinator controls the protocol and establishes the hierarchical nature of the simulator. In every simulation cycle, at time t, a *-message is sent to all objects. Each object then computes its output based on its current state and sends the output as a y-message back to the coordinator. The coordinator will act on the y-messages by sending each object an x-message containing the object's input at time t. The objects respond by updating their internal state to the next discrete time step.

Simply put, there are two types of models. A Mealy-type model has outputs that depend on the object's current state and its most recent input. A Moore-type model, on the other hand, has outputs that depend only on the object's current internal state. The internal state of an instance of a Moore-type model may of course be updated from inputs, but doing a state update and generating the object's outputs are two independent processing steps.

From such a perspective, the simulation cycle may be split into updates to firstly all Moore-type model instances followed by a second phase of recursive updates to the remaining Mealy-type model instances until all the Mealy-type model instances have been activated and have sent their y-messages back to the coordinator. This allows zero delay couplings between components given that the coupled component network is acyclic and therefore of type Moore.

In Chapter 6 it will be seen that the current discrete time simulator implements a variation of the above. The current simulator requires that all coupled components be Moore-like. This results in a simpler simulator which does not require the recursive updates to Mealy-type model instances. The reader may at this stage be wondering how information flows across Moore-type only model instances. The brief answer is that the Mealy-type models in the network are converted to Moore-like models by not allowing delay-less connections between coupled components. The models are aware of the link delays and the simulator has an order independent execute phase followed by an input update—*gather*—phase within each discrete time frame.

Distributed Conservative and Optimistic Discrete Event Time Management

Conservative and optimistic time management techniques exist for the discrete modelling formalism. Basic conservative time management will be discussed followed by an optimistic time management technique. As mentioned in section 2.2.5, optimistic time management improves simulation performance under most circumstances.

Chandy and Misra propose in [32] the use of null-messages to *conservatively manage* the time advance synchronisation of a discrete event simulator. The basic discrete event simulation algorithm works within an important assumption that a model instance will generate output events that are time stamped and ordered chronologically. A model instance is assigned a unique input port for every output to which it is connected, and each input port has a message FIFO queue. The simulation algorithm continually processes the earliest time stamp message—obeying the *local causality constraint*—for as long as all input ports have at least one message queued. However, if any input port has zero messages queued, then the algorithm waits. The reason for this is that the empty input port might still receive a message that is earlier than the earliest messages at the other input ports.

However, such an unqualified wait condition can lead to deadlock. This is solved by a requirement that each model instance should send null messages, indicating time advance, to all its output ports. A null-message effectively announces a time into the future that is the lower bound on the message time stamp arriving at an input port and guarantees that the earliest message may always be chosen among the input ports of a model instance. Even if the earliest message is a null-message it does at least update the object's simulation time. The null-messages therefore has the effect of continually breaking any deadlock that might have existed, but breaking deadlock fundamentally relies on *lookahead* to timestamp the null-messages into the future.

Lookahead is the capability of a model instance to know that it will not generate any output for a certain time into the future. As mentioned, this allows null-messages to be time stamped a lookahead time into the future. Lookahead includes at least the output to input link delays of the coupled component network. The component network must be of type Moore—acyclic or no zero link delay loops—to be simulatable and such a set of link delays is also a solution of the minimum link lookaheads for deadlock avoidance. Zeigler et al. provides the proofs that this type of null-message with lookahead conservative time management *does not violate causality* and *is deadlock free*. More realistic and accurate lookahead beyond the link delay decreases the number of times a null-message is the earliest message amongst

the input ports. Therefore realistic lookahead minimises the time synchronisation network overhead.

Due to the difficulty of calculating accurate lookahead times, the null-message overhead can however be large compared to event execution when event-message traffic is low and lookahead small. In chapter 11 of Zeigler et al., the performance characteristics of conservative time management is described. Zeigler et al. argue that the null-message overhead makes conservative time management unlikely to become of general use for simulation systems.

One technique, proposed by Jefferson and Sowizral[33], to do distributed *optimistic discrete event time management* is Time Warp. It is more complex to implement than conservative time management, but has less overhead than strict causality preservation by null-messages. Optimistic time management, as described earlier, executes in a less controlled way than conservative time management. It does not strictly enforce causality, but rather tries to detect causality violations—on message arrival—and then correct them. To do the correction Time Warp requires the capability to roll-back the execution of a model instance to a time $t_{rollback}$. This in turn requires the state information history, output trace and input trace to be stored to respectively:

- roll back the object state,
- undo the outputs that happened since $t_{rollback}$, and
- redo the inputs after $t_{rollback}$ to finally create the updated output trace.

The information history must at least be kept to as far back as what is called the roll-back time horizon. This horizon is the time beyond which the simulator will certainly not be rolled-back. The time horizon is also called the global virtual time. The potential irregular time jumps of a discrete event simulation and especially the roll-back nature of optimistic time management unfortunately makes the use of hardware interfaces and OIL interfaces generally difficult. The difficulty is due to most hardware systems and humans—to a lesser extent though—expecting a *realistic*—strictly increasing and linear—flow of time. Some research is being done by McLean and Fujimoto[34] to bridge this gap between conservative and optimistic time management—or analytical and real-time simulation as McLean and Fujimoto refers to it.

2.4.3 Basic Hardware Infrastructures

The simulator architectures are, for the purpose of this dissertation, divided into two paradigms. The first is called the manager-worker (server-client)

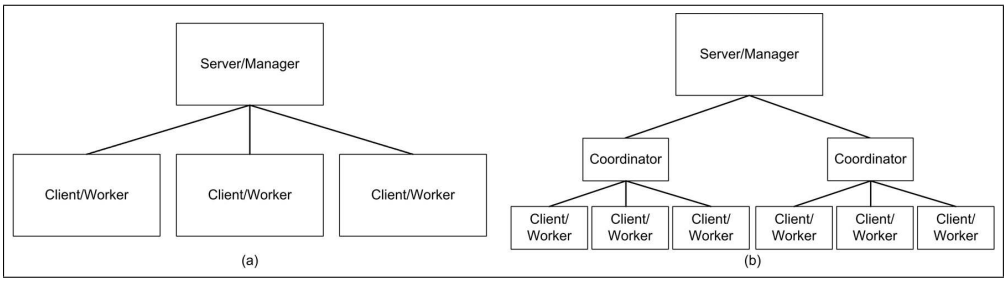

Figure 2.9: The Manager-Worker Structure

paradigm and the second, the peer-to-peer paradigm. A second dimension to the simulator architectures is the communication channel between the processing nodes. The channels may be as diverse as, for example, low latency shared memory to high latency long distance ISDN lines.

The manager-worker architecture allocates one processing node to manage the task distribution to the other processing nodes—the workers. The workers are then responsible for the actual task execution. The manager-worker structure is indicated in Figure 2.9a. Any communication between nodes happens via the server. The manager-worker may also be structured more hierarchically such that a manager becomes the coordinator of its clients to a higher level manager as shown in Figure 2.9b. Magee and Kramer[35] calls this the supervisor-worker architecture. According to them, one of its characteristic features is that the workers may operate completely independently of each other and this architectural pattern is typically suitable in domains where workers with this characteristic is present. If a high level of worker interaction is required, then Magee and Kramer do not advocate this pattern.

A Peer-to-Peer architecture is shown in Figure 2.10a. Each processing node is responsible for managing and executing a subset of the tasks to be executed. The communication between processing nodes happens directly with each processing node typically having a communications connection to every other node. The current discrete time architecture, discussed in Part II, employes a peer-to-peer architecture. The peer-to-peer nodes may also be structured hierarchically, so that one processing node serves as a coordinator between two peer-to-peer networks, as shown in Figure 2.10b.

The communication channels between the processing nodes may be characterised by bandwidth and latency. Bandwidth is defined as the number of pending bytes per second that a receiver can continually remove from a channel, and latency (also referred to as lag) is the delay from the time that data enters into the channel until it becomes pending at the receiver

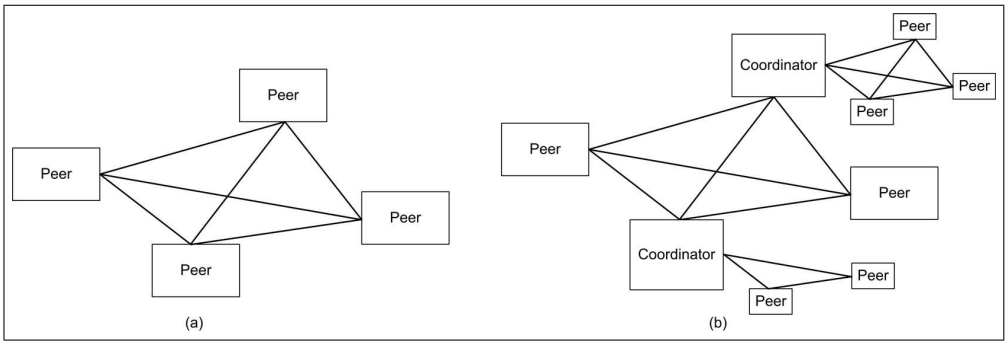

Figure 2.10: The Peer-to-Peer Structure

Table 2.4: Bandwidth versus Latency of Example Communication Channels.

	Lower Latency	Higher Latency
Lower Bandwidth	IEEE1394 (Firewire)	Gigabit Ethernet
Higher Bandwidth	Shared Memory	Posting a stack of DVDs to someone

side. Table 2.4 gives some example communication channels with different bandwidth and latency characteristics. Note that the latency of a communication channel is typically related to the actual physical separation between processing nodes, but also by design to the potential physical separation.

In general, lower latency communication between processing nodes enhances the responsiveness of a system, while higher bandwidth communication aids in alleviating congestion in a busy system. In terms of distributed parallel simulation architectures low latency communication is therefore absolutely required for a high discrete time simulation frame rate[10], while higher bandwidth promotes simulation scalability. Furthermore, the physical separation of processing nodes can be expected to influence the frame rate ceiling of the distributed simulation. There is however a bandwidth to message size trade off characteristic, shown in Figure 2.11, of each communication technology. This affects the simulation scalability ceiling by imposing a messages per second ceiling that results in an actual bandwidth that is lower than the potential maximum bandwidth. This characteristic curve was measured on the Gigabit Ethernet network infrastructure of the current discrete time simulator. Similar graphs will be shown and used in Chapter 7—in

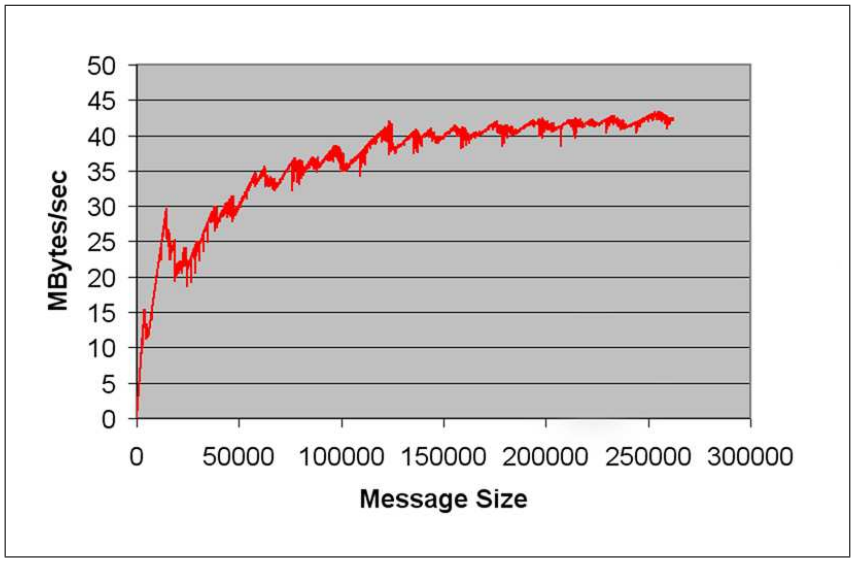

Figure 2.11: Bandwidth to Message Size Trade-Off of a TCP Communication Channel

Part II—when analysing the performance of the current discrete time architecture.

The bandwidth and short message latency characteristics of a communication technology is related to its cost. In the current context it might therefore be argued that it would have been economically sensible to make use of existing shared computing resources, such as the Centre for High Performance Computing (CHPC) located in Cape Town. However, the *field deployability* requirement—in support of field exercises—of the current capability necessitated a *dedicated* infrastructure which can be disassembled, transported and assembled quite easily. The cost implications of such a dedicated infrastructure unfortunately limits the technologies that may be employed in support of the architecture. For this reason the current discrete time simulator, discussed in Part II, uses standard Pentium 4 desktop PCs as processing nodes. The communication backbone is a commercial Gigabit Ethernet infrastructure.

2.4.4 The High Level Architecture

Chapter 1 introduced the HLA as an IEEE standard for simulation interoperability and presented its design goals and key characteristics. This subsection continues with the introduction to the HLA. The history behind the HLA, the incentive for introducing it and its simulation reuse and interoperabil-

ity design goals are presented below. This is followed by indicating how the use of a common object model supports these interoperability and reuse characteristics.

History Behind HLA

According to Page and Smith[8], the most popular and thoroughly analysed, simulation technologies within the military domain seem to be Distributed Interactive Simulation (DIS) and the High Level Architecture (HLA). These simulation technologies support the DEVS BUS concept and DEVS modelling formalism as described in [19][20][15][24][36]. The HLA is a generalisation and extension of DIS and the Aggregate Level Simulation Protocol (ALSP), both of which evolved from the Simulator Networking (SIMNET) project.

SIMNET was developed by the Defence Advanced Research Projects Agency (DARPA) of the US in 1983. It is, according to Page and Smith [8], the first meaningful attempt to interoperate and reuse military simulators within the United States' Department of Defence. From the US DoD's corporate perspective, according to Straßburger[9], the demand for an interoperability standard has a very clear business case. SIMNET successfully demonstrated the viability of reusing and interoperating distributed training simulators, where these simulators were of a type that created a synthetic environment for training military personnel.

SIMNET was followed by DIS which is a set of standards (IEEE Std 1278.1-1995) for general simulation interoperability[37]targeted at *entity level, real-time* simulations. According to Page and Smith[8], ALSP in turn built on the concepts of reuse, interoperability and a general protocol for application to *aggregate level and logical time* simulations. *Dead reckoning* is used to reduce the number of entity state messages sent over the network.

The HLA is the latest interoperability standard, IEEE Std 1516-2000. It is defined for wide application domain and consists of the following three components:

- a common set of rules governing compliance with the architecture.
- common services describing the HLA runtime environment and
- an object model template—a specification of the common format and structure for documenting HLA object models,

The HLA is based on the concept of a *federation* which is a collection of *federate* objects that together create the unified simulation environment.

The *rules governing compliance* of a federate to the HLA are discussed in IEEE Std 1516-2000[6] along with details about the other components. An HLA Runtime Infrastructure (RTI) manages federation execution and information exchange by providing an implementation of the *runtime services*. IEEE Std 1516.1-2000[38] details these standard services and the interfaces to them.The IEEE Std 1516.2-2000[39] documents the HLA Object Model Template (OMT). The primary objective for proposing this template is to facilitate simulation interoperability and component reuse. IEEE Std 1516.3-2003 [40] presents a recommended practice for the processes and procedures that should be followed to develop and execute federations. This recommended practice is given in the form of a high level Federation Development and Execution Process (FEDEP) to meet the requirements of a federation user or sponsor. The FEDEP promotes the development of models that interact with a DEVS view of their surrounding environment. Such a model compliments the DEVS BUS' wrapper concepts often underlying HLA-based simulations. The process consists of seven steps and is also recommended for simulations that are not HLA based.

DEVS/HLA[41] is an *HLA-compliant* M&S environment formed by mapping of the DEVS-C++ system to the C++ version of the DMSO RTI. This allows the development and execution of DEVS simulations under the HLA. DEVS/HLA supports quantisation and model composability over TCP/IP and other networks

Facilitating Interoperability and Reuse through Common Object Models

Although DEVS and DEVS BUS concepts may be applied in the design of an HLA-based *universal* simulator, the semantic side of the interoperability must still be standardised among interacting models. The HLA OMT is the standardised documentation structure of HLA object models. This provides a commonly understood mechanism for specifying the types of objects and interactions that can be exchanged and what each attribute represents for the exchange of data and general coordination among federates.

An HLA object model called the Federation Object Model (FOM) is constructed in accordance with the HLA OMT. The constructed FOM provides the federate specification as well as the structure of the runtime interaction between federates within a specific federation[5].

A second HLA object model, the Simulation Object Model (SOM), specifies the encompassing simulation capability of a specific federate. The federation's FOM may be constructed from the union or set difference, as appropriate, of the SOMs of the participating federates.

Several *reference* FOMs have been developed in an attempt to further facilitate interoperability. This allows independent development of models as federates in a federation. These federates have the shared semantics and they operate within the boundaries of the reference FOM. Using a reference FOM (as opposed to not using one, therefore has the added advantage that no preparation time and effort is required to liaise on and construct a common FOM. According to Straßburger[9], the most known and most discussed reference FOM is the Real-time Platform Reference FOM (RPR-FOM) which was mainly developed for users of the former DIS community. A second reference FOM is the FOM for Synthetic Natural Environment (SNE) which includes dynamic terrain, ocean, atmosphere and space capabilities.

2.5 In Summary

This chapter has given an introduction to the principles and practice of modelling and simulation. The main reason for including this chapter is to establish the required background to be able to describe and interpret the current discrete time simulator within a known framework. Only once this has been accomplished, can existing theorems and corollaries be applied in doing further research. To this end the five levels of Zeigler's system specification hierarchy have firstly been differentiated based on the depth of knowledge required of the system being modelled.

The system specification levels and the common framework for doing modelling and simulation then serve as a primer when thinking and talking about the existing models, the discrete time and proposed hybrid simulator and the entire process of modelling. The system specification modelling formalisms adds further depth to the discussions of models and their simulators, and provides the basic theoretical tools for further research.

The distributed and parallel aspects of models and their simulators have also been presented which provides the last cog in the machine, so to speak, for describing, analysing, understanding and designing the discrete time and proposed new distributed parallel simulators in Part II and Part III of this dissertation.

Beyond retaining the above summary, the post-chapter echo that should remain in the reader's ear is: Using a quantised system approach to discrete event representation of a discrete time system is potentially more efficient than a DTSS—or in the very least of similar efficiency.

Chapter 3

Risks in Migrating to a Discrete Event System Specification

This chapter investigates the risks involved in migrating from a specialised discrete time simulator to a discrete event simulator. The existing models can not be respecified and rebuilt within the DEVS modelling formalism for economic reasons. The migration therefore involves embedding the existing discrete time models in a discrete event system specification such as the DEVS BUS and possible using a discrete event interoperability standard such as the HLA. These risks that are discussed are incurred, whether the migration is from a non-distributed (single process) discrete time simulator or from a distributed discrete time simulator. Only once the migration risks have been identified, can an educated estimate be made of the viability of migrating to a discrete event modelling approach.

The technical viability of simply embedding the discrete time simulator within a discrete event interoperability architecture is studied and identified as the first category of migration risks. The second category of risks relates to the economic viability of using distribution and interoperability standards. The question of how to mitigate the risk within the proposed hybrid discrete-event/discrete-time simulator is discussed. Finally, the question of whether or not to use interoperability standards is considered.

3.1 Simply Embedding a DTSS within a discrete event architecture

A survey of the literature suggests that the two most popular and thoroughly analysed, distributed simulation technologies within the military domain [8] are DIS and the HLA, both of which may implement DEVS or similar modelling formalisms. Ogata et al. [42] tested the real-time performance of DIS and different versions of the so-called HLA RTI-NG provided by the US Defense Modeling and Simulation Office (DMSO), where NG stands for "Next Generation". In the case of both the DIS and the HLA implementations, their real-time vehicle model simulation within a 3D graphical environment reached a frame rate ceiling of around 30Hz.

The HLA's real-time performance, for the RTI-NG implementation, is also studied by Jolibois et al. [43] in the context of a beyond visual range air to air combat simulation. The performance was shown to be less than ideal for 10Hz and higher simulation frame rates. This was due to message latency, object time advance latency and message deliveries leaking into adjacent simulation time steps.

Fujimoto and Hoare [44] investigated an alternative for the current versions of the HLA RTIs that can achieve latencies that are suitable for high simulation frame rates, but these are based on a low latency Gigabit Myrinet [http://www.myricom.com/myrinet/overview/] hardware layer and specialised RTIs. When Fujimoto and Hoare analysed the latency for DMSO RTI over an Ethernet TCP and UDP implementation it was found to be in the order of 10ms, which is too large to sustain a 100Hz simulation frame rate. They also found that the DMSO RTI supports a time advance frequency of more than 2000Hz between two nodes, but for three and more nodes the time advance frequency unfortunately dropped sharply to values as low as 10Hz with even only a few objects per node.

Watrous et al. [45] explain that in the HLA, and in fact in any distributed algorithm, a time management scheme (such as the logical time DTSS modelling formalism) which requires contributions from all other nodes, is relatively expensive. For this reason the HLA allows federates (simulation components) to employ their own unconstrained time management to avoid the time synchronisation overhead. In such an unconstrained case each model instance synchronises itself against its simulator's wall clock without explicit synchronisation with other model instances, or between simulators, but at the risk of loosing message causality.

The MCM research group also has experience in using the HLA, as mentioned in the introduction and as discussed by le Roux[4]. The HLA-based

simulator developed by the group could achieve a maximum real-time frame rate of 10Hz for small GBADS scenarios during the earlier phases of the life cycle. The simulator included an HLA gateway to a human behavioural component—that ran on proprietary process control software—and executed on computing technology which included a 10/100Mbit ethernet infrastructure.

Further, according to Fujimoto[10] the *network latency requirement for the DIS standard* for even tightly coupled interactions is a large 100 milliseconds. This would result in a maximum discrete time frame rate of 5 - 10Hz since a discrete time synchronisation would require at least one round-trip message resulting in a maximum rate of 5 - 10 round-trip messages per second.

3.2 Viability of Interoperability Standards

It has been mentioned that from the US DoD's corporate perspective the demand for an interoperability standard such as the HLA has a very clear business case. However, in pointing this out, Straßburger[9] also states that use of a standard such as the HLA, at least in its early phase, is often seen as a risk to individual program managers who have tight budgets and schedules.

According to Taylor et al. [16], the HLA and other interoperability standards are only economically viable if they are supported as part of a nation-wide simulation reusability and interoperability drive. A nation-wide simulation interoperability drive would ensure that simulations have a common SOM or at least use a shared FOM on their external interfaces making the possible reuse of simulators and models an attractive incentive for using standards. Simulation interconnection within South Africa has not been a present priority though, and therefore a national interoperability drive has not been established. Schulze et al. [46], however, did some research in the advantages of applying HLA within the civilian simulation domain. The civilian domain is similar to the South African military domain in that the use of a common interoperability standard has not been a national mandate.

3.3 Mitigating the Risks

The case studies have shown that it is possible, as Zeigler et al. theorised, to simply express a logical and discrete time system specification using a discrete event formalism while still ensuring message causality. This is known as embedding the DTSS within a DEVS. The case studies have also shown

though, that it is technically difficult to reach and maintain a scalable performance of a 100Hz discrete time simulation frame rate, when adhering to interoperability standards (as embodied, for example, in DIS or the HLA) .

However, the interoperability standards provide *generalised* services—services which might not be required in a *specialised* case. Furthermore, interoperability standards that are written into code may introduce execution overheads which may well be shed when a specialised custom architecture is implemented.

Nevertheless the existing interoperability architectures are used in this study as the DTSS-embedded-within-DEVS benchmark because case studies are readily available and these studies illustrate that nothing significant is gained—compared to the current specialised discrete time architecture—by simply embedding the DTSS within a discrete event architecture.

3.4 In Summary

Following Zeigler et al.'s argument on the efficiency of a DEVS versus that of a DTSS, as discussed at the end of Section 2.3, it seems that a smarter way of migrating to a DEVS modelling formalism is critical in order to fully exploit the higher efficiency of a DEVS—smarter, that is, than simply embedding a logical and discrete time system specification within a discrete event formalism. *The object of the present study is precisely to explore this assertion.*

The situation surrounding the economic viability of interoperability standards does not change when migrating to a DEVS modelling formalism and it still does make the use of such a standard a financial burden as discussed. This is not studied further or taken into account in the comparative study within this dissertation, but the future simulation builder wanting to use a DEVS should weigh the cost of using a standard against the added complexity of building a custom DEVS simulator. The work done by Schulze et al.[46] in applying HLA in the civilian domain may possibly be used as a starting point.

The proposed *smarter way* to embed a DTSS within a discrete event modelling formalism—without respecifying and rebuilding the existing discrete time models—is captured in the hybrid discrete-event/discrete-time modelling approach and simulator discussed in Part III of this dissertation.

Chapter 4

Using UML and CSP

This chapter gives an introduction to two notations used to represent and understand the composed GBADS models and their distributed deployment as well as the simulator processes to the level required to analyse the design and implementation of the respective simulator architectures. The two notations are the Unified Modelling Language (UML) and Communicating Sequential Processes (CSP). The rationale behind this selection is that UML and CSP—slightly less so than UML—is already known within the GBADS simulation development team.

The author proposes that applying UML and CSP to visualise and understand the *existing* composed nature of the GBAD system models, their application domain, distributed deployment and simulators can be accomplished by differentiating a system representation into the following views:

- The system use cases,
- the static coupled component structure,
- the dynamic component interactions, and
- the distributed deployment and operation of the simulator.

The last two views include the model and simulator time management aspects. Subsets of UML and CSP are used to complement each other in representing the different views of the system and simulator.

4.1 Unified Modelling Language

The UML is a visual language for modelling systems and for communicating information about systems. It relies on diagrams and supporting text[18].

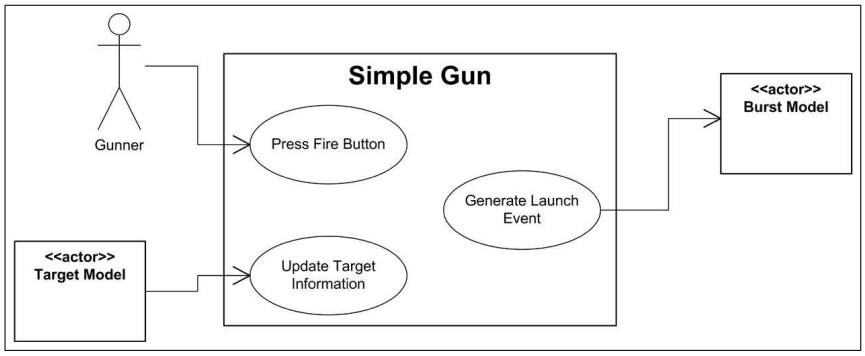

Figure 4.1: UML Use Case Diagram of Simple Gun Model

This dissertation applies UML 1.0, as the modelling tool being used does not yet support UML version 2.0. The UML 2.0 diagram that could in future add value is the *Composite Structure Diagram*.

The system *use case diagrams* capture the functionality of a system. In the context of this dissertation it captures the use of the system model within the chosen experimental frame. Figure 4.1 shows an example UML use case diagram of a simple gun model. The actors—stick men and $\langle\langle actor \rangle\rangle$ boxes—are users with which the system model interacts. A user may be a human or virtual operator represented by a stick man, or it may be another external system represented by an $\langle\langle actor \rangle\rangle$ box. Each ellipsoid shape is a disjunct *use case*. It describes a functional requirement from the perspective of a user of the system. The rectangle surrounding the use cases represents the boundary of the system. The communication associations between the actors and use cases are indicated by the line connectors. The navigation arrow on an association indicates from which direction an interaction is initiated. *No* arrow on a line connector would indicate that either the actor or the system may initiate the communication.

The system's static coupled component structure view is represented using UML component and deployment diagrams. The UML component diagram shows the implemented elements of the system and the dependencies between them. The UML deployment diagram in turn shows the implementation environment of the system, how the different components are deployed and therefore the resources required by the system. An example component diagram nested within its deployment diagram of the same simple gun model mentioned above is shown in Figure 4.2. The nested way of displaying these two diagrams is used throughout.

The 3D box shapes represent the *nodes* of the deployment diagram and the solid lines between the boxes represent the communication associations between the deployed nodes. In this context of a distributable simulator,

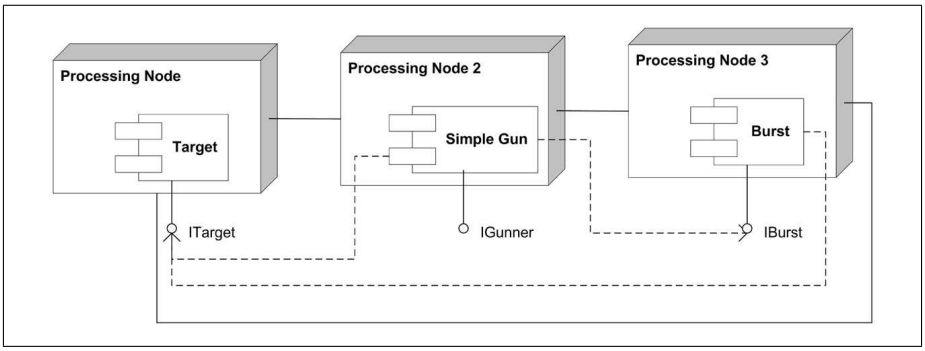

Figure 4.2: UML Component Diagram (nested within a deployment diagram) of the Simple Gun System Model

a deployment diagram node represents a *virtual* processing node. It is virtual in the sense that it is assumed that each virtual processing node has the potential of being distributed to a physical processing node of its own or that is shared by other virtual nodes. A deployment diagram will indicate deployment of components on virtual processing nodes unless otherwise stated. The deployment diagram is also used to indicate the communication requirements between nodes.

The components—elements of the system—are represented by the boxes with bi-protrusions that are drawn within the deployment diagram nodes. The dashed lines between the components indicate their inter dependencies. The only type of dependency that is shown, is the *use* dependency. A *use* dependency from a client component to a supplier component indicates that the client component uses or depends on the supplier component. For example, in Figure 4.2 the simple gun component uses both the target interface *ITarget* and the burst model interface *IBurst* to accomplish its use cases. The *deploy* dependency type is implicit in the node location of a component. In other words, if a component is drawn within a node, then it is deployed on that node.

The dynamic component interaction view of the system is represented by UML sequence diagrams. For the purposes of this dissertation, the dynamic component interaction is represented at the level of the simulator architecture. A UML sequence diagram shows the components of the system as they interact over time. It is important to note that, in general, such a sequence diagram represents a snapshot, depicting one of possibly many permissible sequences of interactions between the relevant components. Figure 4.3 contains an example UML sequence diagram of the simple gun system. The system components—Target, Simple Gun and Burst—are arranged horizontally with each component having a vertical dashed *lifeline*. Time flows along the

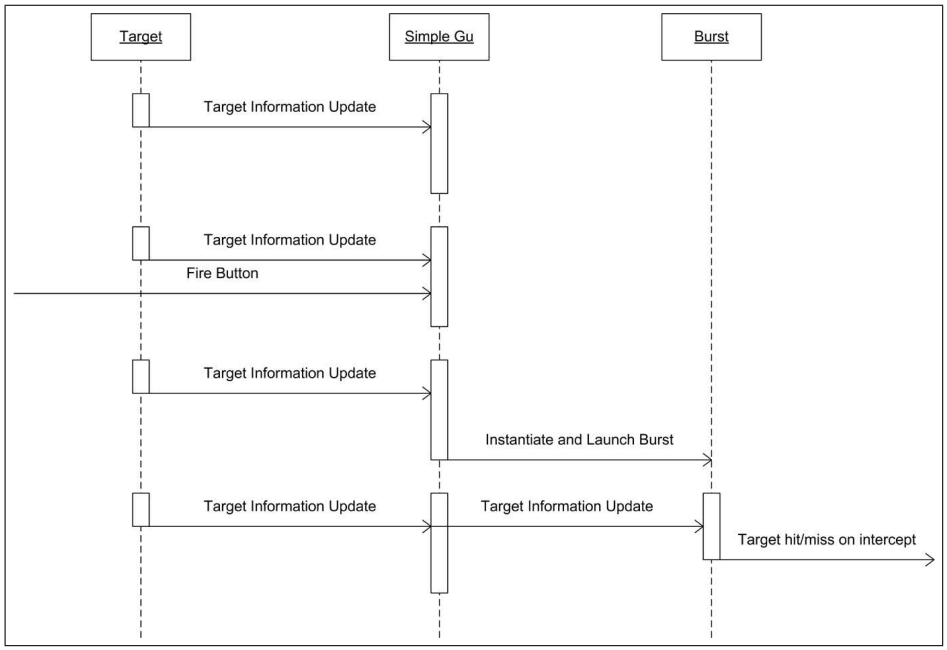

Figure 4.3: UML Sequence Diagram of the Simple Gun System Model

lifelines of the components in a downward direction. The activation periods of a component during its lifetime is indicated by the thin rectangles along its lifeline. To fully understand Figure 4.3 it is important to note that within the example the system components may receive input at any time, but the input messages are cached until the start of a fresh activation period. In addition, within this example, the disrete time components only produce output at the end of each activation period.

As seen, in addition to representing the different model views, the UML diagrams are used to represent the distributed deployment of the discrete time and proposed hybrid simulators. The distributed operation and time management aspects of the simulators may also be represented by making use of CSP as discussed in the next section.

4.2 Communicating Sequential Processes

CSP is, as mentioned, somewhat familiar to individuals within the development team. Its use in describing the operation of the distributed simulators is preferred because of CSPs powerfull set of operators and concise textual notation—which might in future be used with positive effect—and as a personal preference. Its usefullness for the description of distributed simulators

is however also evaluated within this dissertation.

Roscoe[17] refers to CSP as a notation and calculus to help us *understand* interactions between concurrent processes. It is also referred to as a process algebra. The more recent CSP version by Roscoe is used instead of the original version by Hoare[47]. According to Roscoe, the second version is focussed a lot more on *concurrent* systems. Roscoe's book covers the theory, but also practical issues surrounding the use of CSP.

CSP can complement UML in depicting the behavioural and, to a certain extent, the structural views of a system. The value of CSP for the task at hand is however that it is by design a notation for interacting concurrent processes. Such a notation may, to aid understanding, effectively present the dynamic interactions between the distributed processing platforms of the simulators. To this end CSP is used alongside UML sequence diagrams.

As mentioned CSP is a notation for concurrent process interaction, but a set of fundamental language constructs is firstly presented. These allow the creation or description of simple sequential processes within the constraints of various communication patterns. Parallel operators are then introduced to deliver on the promise that CSP creates or describes *concurrent* processes.

4.2.1 Fundamental Language Constructs

The communication events are assumed to be drawn from a set Σ—an alphabet of events—which contains all possible communication events for the processes in the application domain under consideration. Roscoe refers to a communication event as a transaction or synchronisation between two or more processes rather than a transmission of data. An event is seen as being instantaneous and takes place with hand-shaken communication—ie. both sides agree to act. Given an event a in Σ and a process P, $a \rightarrow P$ is the *process* that is initially willing to communicate a with the environment and will wait indefinitely for this a to happen. After a, $a \rightarrow P$ behaves like P. This is known as *prefixing*.

The recursion language construct allows a process to go on performing a sequence of events indefinitely. For example $P_1 = a \rightarrow P_1$ and $P_2 = a \rightarrow b \rightarrow P_2$ define recursive processes which indefinitely engage in the events a in the case of P_1, and a then b in the case of P_2 respectively. Mutual recursion is also possible between two or more processes, as in $P_a = a \rightarrow P_b$ and $P_b = b \rightarrow P_a$. Here, the process P_a behaves like the process P_2 in the previous sentence. A diagram depicting recursion and mutual recursion is shown in Figure 4.4.

The *guarded alternative* language construct allows the environment to choose any one of n events $a_1...a_n$ of a process P. Depending on the choice of event, P may then behave like any one of a number of processes $P_1...P_n$.

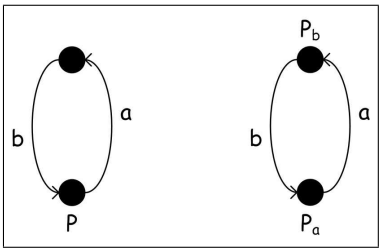

Figure 4.4: A Diagram of Recursion and Mutual Recursion

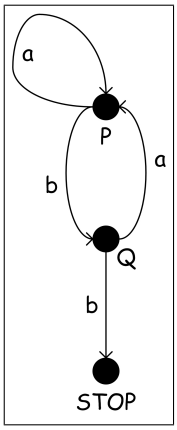

Figure 4.5: A Diagram of a Guarded Alternative combined with Recursion[17]

This is written as $P = (a_1 \rightarrow P_1|...|a_n \rightarrow P_n)$. Another example, $P = (a \rightarrow P|b \rightarrow Q)$ and $Q = (a \rightarrow P|b \rightarrow STOP)$, which combines the guarded alternative construct with recursion and mutual recursion is drawn in Figure 4.5. $STOP$ is a process that does nothing and may be used as a representation for *deadlock*.

Roscoe also uses a system of counter processes as a more complex example of combining the guarded alternative construct with recursion. The system, shown in Figure 4.6 may be defined as $COUNT_0 = up \rightarrow COUNT_1$ and $COUNT_n = (up \rightarrow COUNT_{n+1}|down \rightarrow COUNT_{n-1})\forall(n > 0)$.

If $A \subseteq \Sigma$ is a set of events and, for each $x \in A$, we have defined a process $P(x)$, then $?x : A \rightarrow P(x)$ defines the process which accepts any event a in A and then behaves like the appropriate $P(a)$. This construct is known as prefix choice. $Initialise = ?n : \mathbb{N}_0 \rightarrow COUNT_n$ for example initialises the system of counter processes by accepting a non-negative integer n and then behaves like $COUNT_n$.

A generalisation of the guarded alternative construct is the external choice operator, $\Box$. $P \Box Q$ is the process which offers the environment the choice *of*

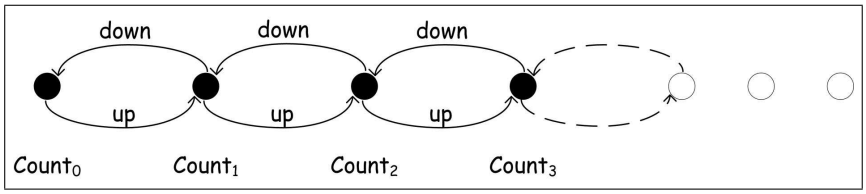

Figure 4.6: A Diagram of a System of Counter Processes[17]

the first events of P and of Q. Guarded alternative and external choice agree in that if the first event chosen is from P then $P \Box Q$ behaves like P would have, after engaging in that event. Similarly if the first event is chosen from Q then $P \Box Q$ behaves like Q would have, after engaging in that event. However, external choice allows P and Q to have initial events in common. After the first *in common* event has been chosen, $P \Box Q$ *non-deterministically* behaves like either P or Q would have, after engaging in that event.

4.2.2 Parallel Operators

Parallel operators allow the description of concurrent processes. Such concurrency occurs inherently in parallel and distributed simulators. The main goal is to be able to describe the behaviour of processes which are distributed across multiple computing platforms, which may operate independently and which may interact when necessary.

The synchronous parallel operator is the simplest in that it insists that processes agree on *all* events that occur. In effect, a synchronous parallel process $?x : A \rightarrow P(x) \;||\; ?x : B \rightarrow Q(x)$, composed of the two processes $?x : A \rightarrow P(x)$ and $?x : A \rightarrow Q(x)$, is equivalent to the process $?x : A \cap B \rightarrow (P(x)||Q(x))$.

$||$ is symmetric in that $(P||Q) = (Q||P)$, associative in that $(P||Q)||R = P||(Q||R)$ and distributive over external choice, so that $P||(Q \Box R) = (P||Q) \Box (P||R)$.

A more general version of the parallel operator is the alphabetised parallel operator. If X and Y are subsets of Σ, $P \;{}_X||_Y\; Q$ is the combination where P is allowed to communicate in the alphabet X, Q is allowed to communicate in the alphabet Y and the two processes must synchronise on the events in $X \cap Y$. It easily follows that $P \;{}_\Sigma||_\Sigma\; Q = P||Q$.

Yet another parallel operator is the *generalised parallel* or *interface parallel* operator. Written as $P \overset{||}{_X} Q$, where all the events in X must be synchronised and events outside X may proceed independently.

Parallel composition by interleaving behaves almost opposite to the parallel operators discussed so far. P interleaved with Q, written as $P|||Q$, is

equal to $P \underset{\{\}}{||} Q$. This states that P and Q executes strictly independently of each other. It touches on another aspect about parallel operators which has not been mentioned yet. That is, $P \underset{X}{||} Q$ implies that an event outside of X that are within both P and Q's alphabets can never be communicated simultaneously and $P|||Q$ implies that *no* event within P and Q's shared alphabet may be communicated simultaneously by both P and Q.

This survey of selected CSP operators is necessarily very brief. A full treatment of CSP, its various operators, its complete semantics under so-called traces, failures and divergences, respectively, the various algebraic laws that hold for the different operators, etc, may found in Rosco [17].

4.2.3 Using CSP to Describe Discrete Time and Discrete Event Simulators

Normal untimed CSP, where exact times of interaction events are unimportant, is differentiated from *Timed CSP*. According to Roscoe, *Timed CSP* uses a continuous model of time and has a mathematical theory quite distinct from untimed CSP. However, for practical reasons Roscoe rather places a *timed* interpretation on the untimed language. The passage of time may be signalled by the regular occurrence of a specific generalised synchronous event *tock*.

This is ideally suited to this dissertation and Roscoe even gives an example of how a discrete time system may be thought about and depicted in CSP. The discrete time CSP includes the event *tock* on which all processes synchronise. This is similar in nature to the behaviour of a simulator for the multiDTSS formalism. (This formalism was described in Section 2.4.1 which dealt with the coupled extensions to the DTSS and DEVS formalisms.) Figure 4.7 shows an example of modelling discrete time across three processing nodes A, B and C. The functioning of node J (where $J = A, B, C$) may be described in CSP as $J = executeDone_J \rightarrow tock \rightarrow J$. The overall system in the figure can be expressed as $A \underset{\{tock\}}{||} B \underset{\{tock\}}{||} C$. This means that the events $executeDone_A$, $executeDone_B$ and $executeDone_C$ can occur in any order, and that only after all have occurred can the three subsystems A, B and C engage in their common *tock* event. Thereafter, A, B and C recommence with their respective execute events.

In a similar manner, Chapter 6 applies the discrete time CSP to depict the current discrete time simulator. The use of *Timed CSP* for description of discrete event simulators may possibly prove useful and should in future be investigated, but for this dissertation only untimed CSP is considered.

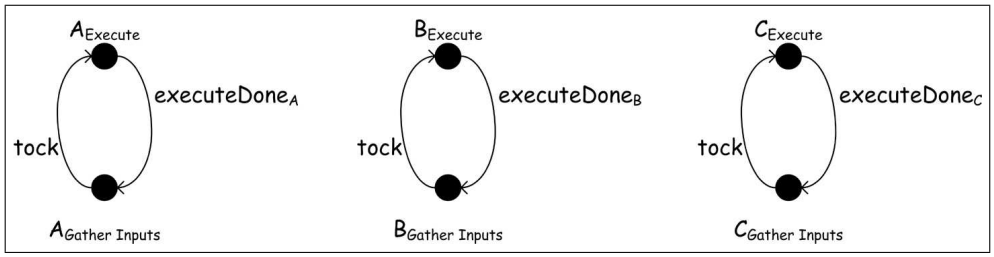

Figure 4.7: A CSP Diagram of Modelling Discrete Time

4.3 In Summary

The chapter has given an introduction to UML and CSP to represent—and in this way aid understanding of—the composed GBADS models and the current and proposed simulators. UML has been presented for the representation of:

- The system model use cases (the experimental frame to some extent) through UML Use Case Diagrams,
- the static coupled component/model structure through UML Component and Deployment Diagrams,
- the dynamic component interactions, and
- the distributed deployment and operation of the simulator.

CSP has also been presented for the representation of distributed simulators as CSP is a powerful notation and calculus to aid understanding of interactions between concurrent processes. CSP has therefore been presented specifically for representation of the last two system views in the above list. CSP is also supported by a large variety of tools that support the modelling and checking validity aspects such as proneness to deadlock. A CSP tool has been used in exactly this way in Chapter 6 to show that the current discrete time simulator architecture is deadlock free.

Chapter 5

Introduction to the Ground Based Air Defence (GBAD) System Model

In this chapter, the focus is returned to the GBAD domain. The intention is to build a clear picture of the GBAD system of systems (of subsystems). The GBADS model is characterised in terms of system scale. This is done to facilitate the identification of the model parameters that impact on performance when the system is scaled up.

Once such a sensitivity analysis is done, the system model and simulator may be appropriately instrumented to analyse its behaviour in terms of the system scale and simulator distribution. Representative scenarios are created for use in analysing the performances of the current discrete time simulator for small to large scale systems and distributions. These scenarios are then used to analyse the performance of the proposed hybrid discrete-event/discrete-time simulator and as the benchmark scenarios for comparing the performance of the proposed simulator to that of the current discrete time simulator.

5.1 The GBAD System of Systems (of Sub-Systems)

The GBAD system was introduced briefly in Section 1.1. This section discusses in greater detail the air defence system's various components—system of systems and subsystems as decomposed for the purpose of modelling—and the interactions between them.

The air defence system is deployed to defend some vulnerable point or

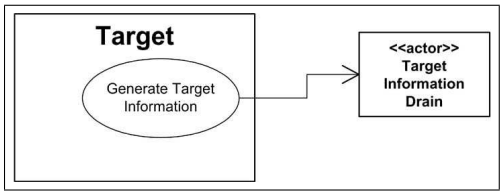

Figure 5.1: UML Use Case Diagram of the Target

valuable asset—the *defended asset*—against one of several possible threat scenarios, whose description is included in an expected set of airborne threat scenarios. A threat scenario may be composed of a number of incoming fixed wing aircraft, rotary wing aircraft, UAV and cruise missile targets. The munitions released by incoming aircraft are considered as targets in their own right. Each target may be simulated by either an airborne entity model or controlled via a pilot flight seat integrated with the simulation. In the first case, the targets' approach paths and weapon release times are pre-planned and fixed. In the second case a real pilot sits in the flight seat and may perform evasive manoeuvres against the ground based weapons. With tactical doctrine governing the operational behaviour of the ground crew weapon and other equipment operators, the target positions and velocity vectors may be seen as the action triggers for the rest of the simulation. For this dissertation only pre-planned targets are considered. The target model's use case is shown in Figure 5.1 and as discussed the preplanned targets have no inputs to stimulate reactive behaviour.

The ground based deployment relies on sensor information to gauge and engage the incoming threat scenario. The sensors that are valuable in this regard are search radars, radar- and optical trackers, and the human operators' eyes. Each sensor is simulated by a sensor model which requires as input up to date target state information. Sensors may be differentiated into two types namely, *search or scanning sensors* and *tracking sensors*. A search sensor is responsible for detecting all targets within its detection range and normally has a very wide Designated Region Of Interest (DROI). A tracking sensor is responsible for detecting and accurately tracking one to a small number of targets within a much smaller DROI.

The use case of a generic sensor is shown in Figure 5.2. Multiple target information sources—multiple target objects— may be connected to the sensor. The Fire Control System (FCS) provides the DROI.

The output of a tracking radar is a sequence of state updates of the tracked targets (position, velocity, etc). This sequence is characterised as having a high time resolution. The target detections returned by a search radar, on the other hand, are typically updated once every four seconds as

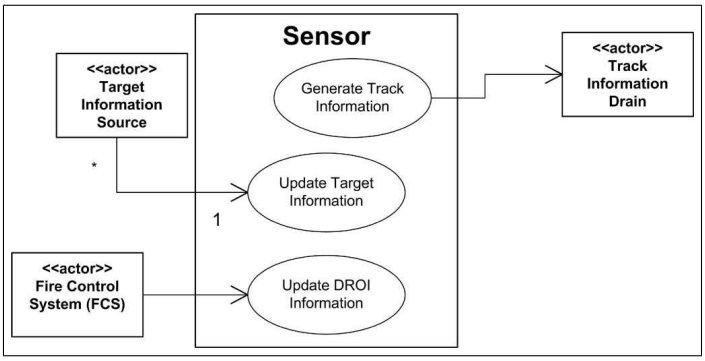

Figure 5.2: UML Use Case Diagram of the Sensor

the radar repeatedly scans its horizon. The *Air Picture Sensor* is a special sensor that uses data fusion to combine the outputs of a number of different sensors into a consolidated view of the surrounding airspace.

To be able to locate targets and accurately guide munitions to them as required, the weapon system may be reliant on the search radar's detections, or on the tracking radar's, or on both. The weapon system as referred to here consists of two parts:

- the weapon's Fire Control System (FCS); and
- the launcher and its munition.

The weapon FCS is responsible for receiving *designations*—which air picture tracks to prioritise—and *engagements*—which priority tracks to launch at. These orders are generated by the Fire Control Officer (FCO), which is a single operator that is responsible for the GBADS deployment's fire control. The FCS may also have associated with it, additional operators, such as the Fire Unit Commander (FuCmdr) and gunner. Such operators are specialised for the specific type of weapon. These details are however not discussed and the FCS may be seen as containing these *helper* operators. The FCS model requires the sensor data, the Fire Control Officer (FCO) commands and launcher/munition status as inputs, and provides launcher/munition commands, sensor DROI commands and operator feedback as output.

The FCS's use case diagram is shown in Figure 5.3. It is clear that the FCS intelligently interfaces the launcher and munitions to the air picture, the sensor and the FCO. The combination of weapon system's *sub-systems* and the designation sensor *sub-system* are referred to as a Fire Unit (FU) which is a GBAD *system level* component.

Each type of weapon has a unique physical configuration of its launcher and munitions, one being the gun example in the previous chapter. The

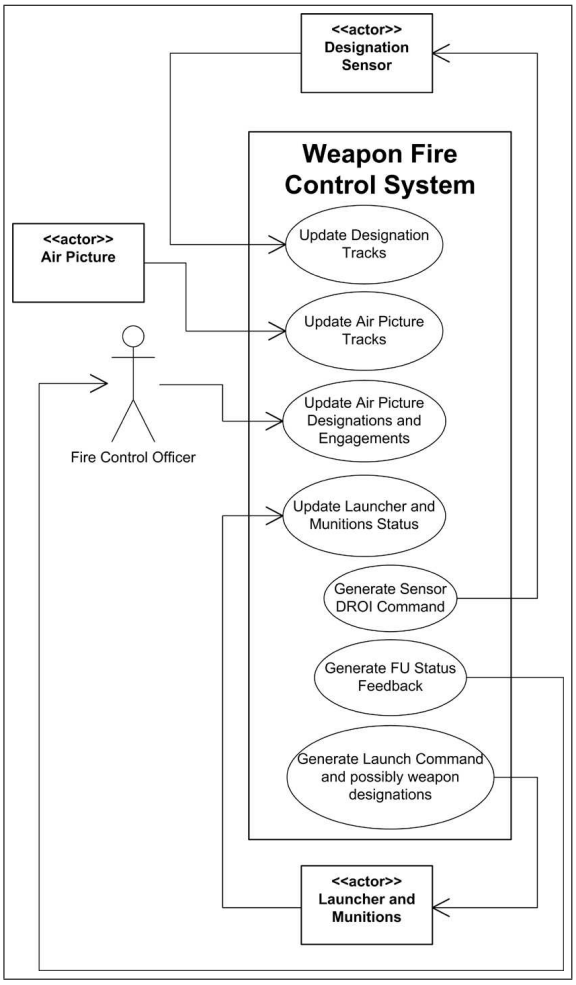

Figure 5.3: UML Use Case Diagram of the FCS

munitions may be ballistic or guided, and the launcher may be be able to engage one or multiple priority targets. What is common is that the launcher and munition models require the sensor data—possibly provided via the FCS model—to engage the target. Secondly the actual target state information is required for the model to test for a hit or miss of the munition. The FCS and operators within the FCS, such as the FuCmdr and gunner, provide the required translations of the FCS's designate and engage commands to the launcher and munitions as applicable to the specific weapon's designation and launch requirements. The use case diagram of the launcher and munitions is shown in Figure 5.4.

The weapon and FCS operators may be played by real operators interacting with mock-ups of the equipment terminals interfaced to the simula-

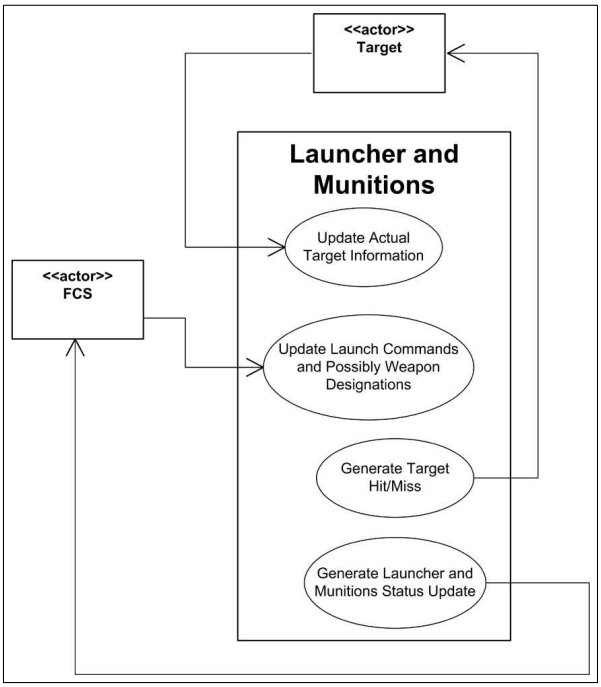

Figure 5.4: UML Use Case Diagram of the Launcher and Munitions

tion. However, the behaviour of most operators is based on known tactical doctrine. As a result, their standard behaviour can be modelled and their presence virtualised as required. For the benchmark scenario however, only virtual operators will be used.

Tactical data and voice communication networks are responsible for connecting radars and operators to the weapon systems. These communication networks are simulated by a network model. The objective of adding a communication network model is to simulate transmission delays and possibly transmission failures due to aspects such as line of sight restrictions, range restrictions on radio communication and network congestion. A simple distributed *per-receiver* network model is currently being used. This model shifts the responsibility of simulating the communication delays and failures to the receiver of a message. Such a simple model cannot include effects such as network congestion, but it has the advantage of being encapsulated within the GBAD system and sub-system models discussed.

Figure 5.5 depicts a minimal GBADS deployment with one target (c) and one FU (b). Figure 5.5a depicts the components shared among all FUs. The shared components are:

- The FCO in that it controls the fire of all the FUs,

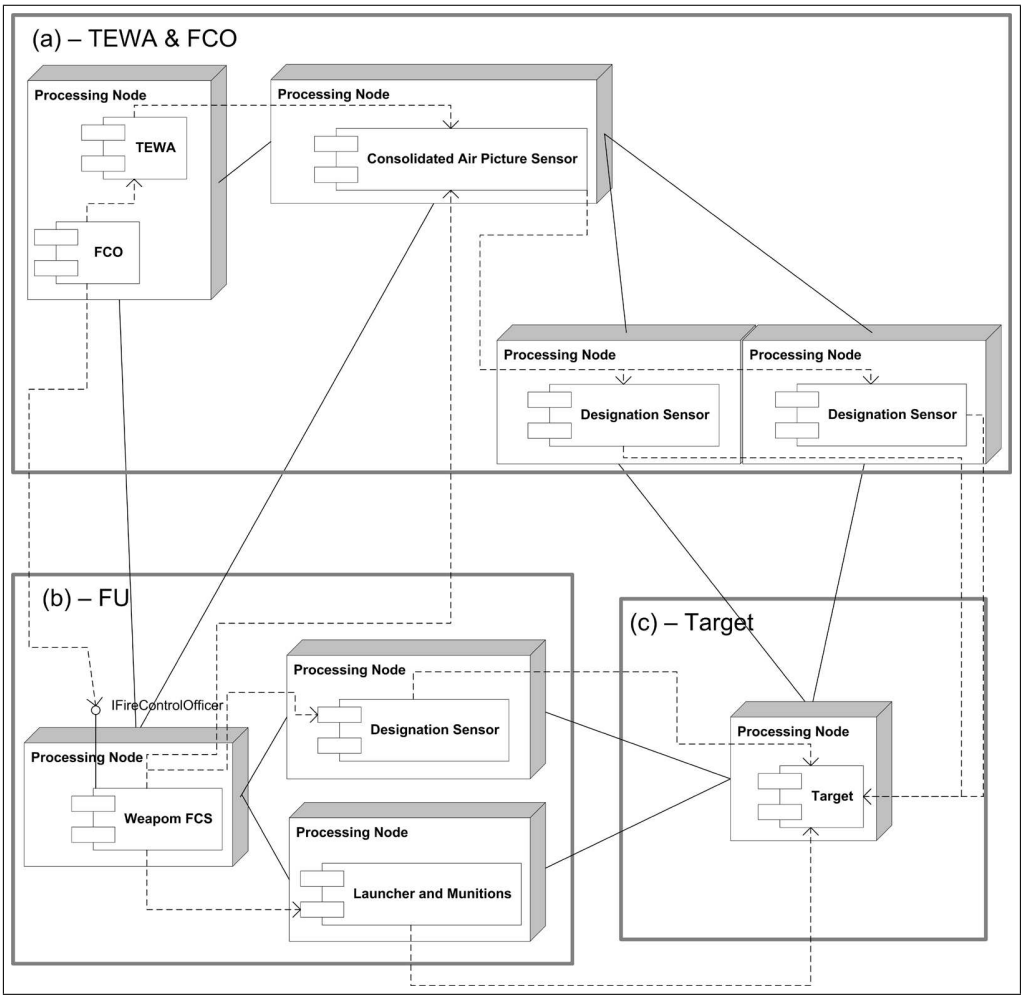

Figure 5.5: UML Component Diagrams (nested within their deployment diagrams) of the building blocks of a GBAD system of systems deployment

- the Threat Evaluation and Weapon Assignment (TEWA) system which provides fire control suggestions to the FCO, and
- a small number of designation sensors that are deployed for use by all the FUs.

A threat scenario would typically consist of multiple targets and the GBAD system of multiple FUs. The components of the deployment shown in Figure 5.5 are therefore the basic building blocks of a GBADS deployment.

Note that, although the figure represents a specific instance of a deployment, some of the components are still generalised to component type rather than specified in terms of exact components. An example of an actual

GBADS deployment is given in Chapter 7 when the simulator performance is analysed.

The number of targets and the number of FUs are two system model parameters. These two parameters define the scale of the deployment. As the number of FUs increases, the systems's execution performance is impacted by the cost of distributing the target information to more and more FUs. Alternatively, as the number of targets increases the performance is impacted by the cost of distributing more and more targets to each FU and then processing the targets. Adding targets therefore increases the global communication overhead *and* the computational requirements of the FUs.

The shared Air Picture, TEWA and Virtual FCO components impact the execution performance further as each of these are internally sequential and, according to Amdahl's law[48], decrease the potential parallelism of the system model. Amdahl's law states that: The speed-up of a program using multiple processors in parallel is limited by the sequential fraction of the program. It should be noted here though, that in the current discrete time simulator, the sub-systems within each FU are also distributable across different physical nodes. No preferred way of distributing the model instances across a number of processing nodes currently exists and the default approach has been to distribute the model instances randomly between the processing nodes. Duvenhage and Nel [49] have, however, done some preliminary analysis—on small scale distributions—which shows that a random distribution typically, for the GBADS scenarios, comes to within 85% of the best case distribution. This behaviour is expected to change for larger distributions as the communication overhead becomes a more significant part of the simulation execution time.

5.2 GBADS Benchmark Scenarios

The goal of constructing the benchmark scenarios is to analyse the distributed scalability behaviour of the simulators populated with instances of the *actual* existing models and not simplified or generic versions of them. Using the actual models gives credibility to the results. Furthermore it is easy to get representative behaviour and performance. The disadvantage is, of coarse, that the analysis of the reasons behind the execution performance can become more difficult. The analysis of the execution performance is done in Chapter 7 and Chapter 9 for the respective simulators.

The GBADS model's number of targets and number of FUs were chosen to define the system scale. The GBADS benchmark scenarios are further very much based on the deployment in Figure 5.5. What remains to be defined is

the number of targets, the number of FUs and the exact weapon and sensors to use for each scenario.

A single weapon type is used and for this purpose a 35mm gun Close In Weapons System (CIWS) FU type is chosen. It is considered to be permissable to use a single weapon type, as the FUs are independent of each other. An example of an actual GBADS deployment is given in Chapter 7.

For the purpose of this dissertation the distributed execution performance of the GBADS model should be analysed over one to many processing nodes. This should be done for a number of scenarios with varying number of FUs and targets. However, to decrease the number of free variables in such an analysis the number of targets is fixed to two per FU. The two free variables in the performance analysis, that is done in Chapter 7 and Chapter 9, are therefore the number of FUs in each scenario and the number of processing nodes that the scenario is distributed over.

5.3 In Summary

The aim of this chapter was to build a clear picture of the composed nature of the GBAD system of systems (of subsystems) models. This was done by making use of UML diagrams as presented in Chapter 4. The potential distributed deployments of the models were also shown.

The basic—aggregated—building blocks of a GBAD system model was then identified and the *scale* of a GBADS defined. Scenarios, representative of small to large scale deployments, can now be created using the identified building blocks. These scenarios are used in Chapter 7 and Chapter 9 for the scalability analysis of the current discrete time simulator and an implementation of the proposed hybrid simulator.

Part II

The Discrete Time Simulator

Chapter 6

The Discrete Time Simulator

The discussion on the current publish-subscribe simulator architecture is structured around the layered architecture of the simulator (shown in Figure 6.1), which includes a publish subscribe simulation layer, a message passing implementation of the simulation model and at the bottom layer a low latency TCP messaging protocol for Gigabit Ethernet.

The attraction of the layered architecture was the separation of concerns, in terms of design, between the simulation model as designed into the top layer and the distributed execution thereof, a concern delegated to the bottom two layers. An additional advantage is of course the ability to change the implementation of the bottom layers without affecting the top layer simulation application.

In Chapter 2 it was mentioned that the current simulator does not support zero delay links between coupled components. All output to input links between coupled components have, by design, one simulation time frame delay and this is taken into account in the modelling. More formally, each feedback cycle in the GBADS model has at least one Moore-type system (usually an aircraft object) or a Mealy-type system that is modified to become Moore during the modelling process. One example of a modified Mealy-type system would be a missile control system's target state predictor model that predicts to time $t_{Future} + dt$ instead of t_{Future} as its internal *deduced* target state is limited to only having a non-delay-less link to the sensor input data. Its internalised sensor track data is therefore one time step in the *past*. That is, the predictor has received sensor track data up to $t - dt$, where t is the current simulator time.

The resulting coupled component network is guaranteed to be solvable or simulatable by numerical methods. Just as important however, the coupling still allows the much required output to input feedback cycles within the GBADS model which is described in Chapter 5. The gun tracking system

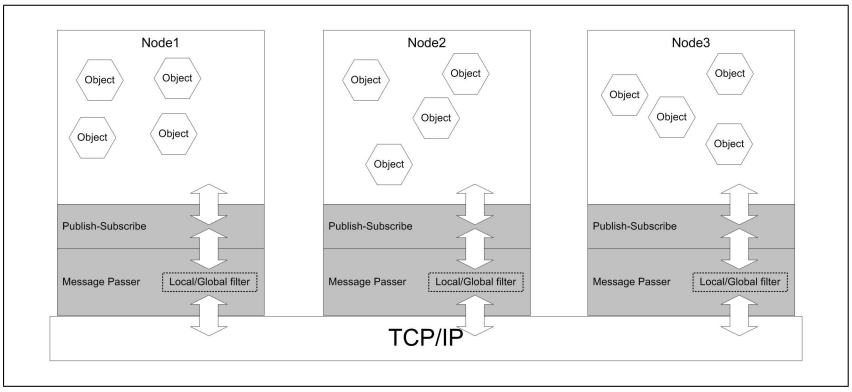

Figure 6.1: Layered Peer-to-Peer Simulator Architecture

and a missile control system are two examples that require feedback cycles to be modelled.

The current simulator however also forbids pure (un-Moore'ed) Mealy-type components altogether. This has additional advantages: firstly, it ensures that the execution of the components become order independent; and secondly, it then allows the distribution of the coupled components across the computing platforms to be independent of their coupling to one another. The order independence allows for an elegant increment-publish-gather simulation update cycle in the top layer of the architecture. This is somewhat similar to the MultiDTSS simulator described in Chapter 2.

6.1 Publish-Subscribe Simulation Model

The top layer simulation model encompasses a number of aspects, which include the simulation time management, the system specification modelling formalism, the object communication framework and the synthetic environment services.

As mentioned, the pre-existing models—before the current discrete time simulator—had been implemented within a conservative logical time management scheme and a discrete time modelling formalism. It was decided to keep these aspects unchanged within the now current simulator to simplify the reuse of the existing models. The object communication framework that is under investigation for the simulation model is a specialised publish-subscribe framework to be discussed next. Discussions on the synthetic environment services then follow.

6.1.1 The Publish-Subscribe Object Communication Framework

The publish-subscribe paradigm is well known as a means to regularly acquiring information. For example, someone subscribes to a magazine on his or her topic of interest in order to receive the information on a regular basis. Each magazine within one's topic (category) of interest has a title and a regular interval at which the categorical information is made available (published). The subscriber may request that the information be delivered to one's doorstep in the form of, say, a weekly or a monthly magazine issue.

The publish-subscribe simulation framework is a direct analogy to the magazine example. An instance of a simulation model (an object) may express its desire to receive information within a certain category of interest, e.g. aircraft positions, by adding the category (and title name, if known) to its *Subscription Wish List*. An object may also express its willingness to share information within a certain category, such as its own position, by adding a title (name and category) to its *Owned Title List*. A subscribing object has no guarantee that any object will share information under the title category or name in which it is interested. Similarly a publisher object has no guarantee that any other objects will be interested in the information that it is willing to share. At simulation start-up each object's owned title list is made known to the rest of the simulation. The titles are then processed against the objects' wish list subscriptions. Each title matching a wish list subscription generates a subscription which is sent back to the title owner to be added to the title's subscriber list.

At simulation run-time each processing node goes through regular *increment, publish and gather cycles*. The *increment phase* updates each object's internal state from its *cached* inputs. The *publish phase* generates each object's output issues. Each issue is sent to the messaging layer addressed to the appropriate subscriber or subscribers if there are multiple. The *gather phase* follows, during which each object (now in role of subscriber) *receives* the published issues from all other objects via the messaging layer, thereby refreshing the subscriber's input cache.

Consider three discrete time processing nodes A, B and C. For process A (visually shown in Figure 6.2):

$$\begin{aligned} A_{Increment} &= incrementDone_A \rightarrow A_{Publish} \\ A_{Publish} &= publishDone_A \rightarrow A_{Gather} \\ A_{Gather} &= frameDone_A \rightarrow A_{Increment} \end{aligned}$$

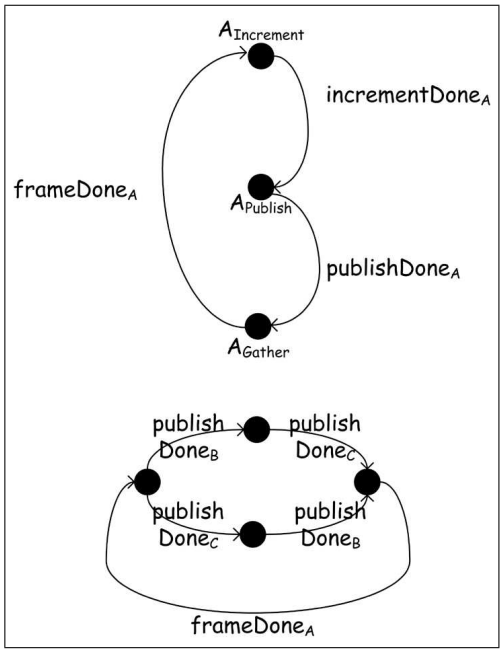

Figure 6.2: A CSP Diagram of the Increment Publish Gather Cycle composed of A and A_{sync}

$$A_{sync} = (\quad publishDone_B \rightarrow publishDone_C \rightarrow frameDone_A \rightarrow A_{sync} \quad | \\ publishDone_C \rightarrow publishDone_B \rightarrow frameDone_A \rightarrow A_{sync} \quad)$$

Similarly for process B:

$$\begin{aligned} B_{Increment} &= incrementDone_B \rightarrow B_{Publish} \\ B_{Publish} &= publishDone_B \rightarrow B_{Gather} \\ B_{Gather} &= frameDone_B \rightarrow B_{Increment} \end{aligned}$$

$$B_{sync} = (\quad publishDone_A \rightarrow publishDone_C \rightarrow frameDone_B \rightarrow B_{sync} \quad | \\ publishDone_C \rightarrow publishDone_A \rightarrow frameDone_B \rightarrow B_{sync} \quad)$$

and for process C:

$$\begin{aligned} C_{Increment} &= incrementDone_C \rightarrow C_{Publish} \\ C_{Publish} &= publishDone_C \rightarrow C_{Gather} \\ C_{Gather} &= frameDone_C \rightarrow C_{Increment} \end{aligned}$$

$$C_{sync} = (\quad publishDone_A \rightarrow publishDone_B \rightarrow frameDone_C \rightarrow C_{sync} \quad | \\ publishDone_B \rightarrow publishDone_A \rightarrow frameDone_C \rightarrow C_{sync} \quad)$$

$A = initialise \rightarrow A_{Increment}$ is the mutual recursive increment, publish and gather cycle of processing node A. For B and C defined similarly to A, the parallel composition $(A \mid\mid A_{sync}) \mid\mid (B \mid\mid B_{sync}) \mid\mid (C \mid\mid C_{sync})$ describes the way in which the three processes jointly interact with one another. The composition is the description of the discrete time architecture and it can be shown that the overall interaction between the processes is deadlock-free. This has in fact been done by re-expressing the above CSP in its FSP equivalent and applying the LTSA analyser provided along with the book by Magee and Cramer[35].

An object is incremented every n'th discrete time simulation frame where n is the object's *trigger frame period*. Each wish list subscription, and therefore each subscriber in a title's subscriber list, is also associated with its own trigger frame period. During publish, each subscriber entry of each owned title is visited and an issue sent to the subscriber object if it is the subscription's trigger frame. An important publish rule that is required to ensure consistent issues is that the contents of a title issue may only be updated during the publisher's increment cycle. Objects may during the simulation run express their wish to share a new category of information or a new title within an existing category. This is done by submitting a *run-time title* to the communication framework. Similarly objects may express interest in categories (or titles within categories) of information during the simulation run by submitting a *late subscription*.

An object has an issue pigeon hole for each of its wish list subscriptions. When an issue is received from the messaging layer, during the *gather phase*, it is placed in the appropriate pigeon hole. A pigeon hole may have subscription history turned off or on. If history is off then a newer version of an issue replaces all old issues that may remain in the pigeon hole. If history is turned on then issues are added to the pigeon hole in chronological order. The object may then read issues and manually delete them as required during its increment phase. Turning history on for a specific wish list subscription is typically required when a subscriber doesn't want to miss any important updates (events) for that subscription. Having history off allows the subscriber to always have access to the current issue without the overhead of always caching and processing a subscription's recent history. The populated issue pigeon holes serve as the input cache, mentioned at the beginning of this chapter, which stores the inputs until the next increment cycle.

6.1.2 The Synthetic Environment Services

The two types of simulation services supported are, firstly, low level services that are built into the simulation model and, secondly, high level services that run on top of the simulation model as simulation objects. The only low level service currently implemented is that of delayed issues. An issue may be given a future delivery time by the publisher or the subscriber may delay the processing of an issue. Such an issue is, however, delivered to the subscriber immediately, but once there it resides in a *delayed issue list* until the time of delay has passed at which point the issue is put into the appropriate pigeon hole of the subscriber. Delayed issues are handy if transmission delays of messages within the synthetic environment are to be modelled. In the current simulator the message delays of the tactical communication network model are implemented by making use of delayed issues.

High level synthetic environment services subscribe to titles of which the issues contain information such as object position. Environmental information services such as Line Of Sight (LOS) and terrain engines to then give each object individual feedback on its height, which objects it can see. To accomplish the personalised feedback a service advertises what is called a stem or differentiatable title. Each time a subscription is made to a stem title the simulator automatically differentiates the stem title to a personalised title and subscription for the subscriber. The service may then use the created titles to publish to individual objects.

A service need not always publish data back to the simulation, though. Logging, for example, is a high level service that accumulates object states and other information. The logging service may then apply user configured data analysers to the accumulated data and log the results to disk.

6.2 Peer-to-Peer Message Passing and Node Synchronisation

The publish-subscribe communication framework and the simulator synchronisation is implemented by means of a peer-to-peer message passing architecture. A peer-to-peer architecture is specifically preferred above a client-server architecture to avoid the double latency that exists when a sender machine communicates via a server to a receiver machine. The double latency may be avoided if the sender communicates directly with the receiver. The messaging implementation of the publish-subscribe communication framework is presented, followed by the implementation of the simulation synchronisation.

6.2.1 Messaging Implementation of Publish-Subscribe

The publish-subscribe framework naturally translates to a messaging architecture containing only three message types.

- A title may be advertised as a title message containing all the title and publisher details.
- A wish list subscription may similarly be a message containing the details of the wish list subscription and the subscriber.
- The third message type is an issue message that contains the subscriber's node-number delivery address, the targeted wish list subscription pigeon hole and the actual issue payload.

The messaging implementation has a local/global filter (see Figure 6.1) that loops a node's self addressed messages back to be cached for the next simulation frame without passing anything down to the TCP layer.

6.2.2 Peer-to-Peer Node Synchronisation

The peer-to-peer synchronisation scheme is shown in Figure 6.3. As mentioned in the previous section, each simulation frame has three phases. Within the publish phase, the published issues are not messaged directly, but are grouped per destination node and sent in message bundles to optimise bandwidth usage. The publish phase must be followed by a time-stamped publishDone message to each peer node to signify that all the issues for the current simulation frame have been sent. The publishDone messages perform a similar function as Chandy-Misra null messages [32] do—in conservative distributed discrete event simulation—in terms of dead-lock avoidance. A simulator node waits in the gather phase until it has received and processed a publishDone message from each of the other simulator nodes. Having received the publishDone messages from all the other nodes guarantees that all messages for the current simulation frame have been received. The node may now start with the increment phase of the next simulation frame in effect having generated a frameDone message.

6.3 TCP Message Passing Implementation

The communication setup requires a process to process connection on top of Ethernet. The IP protocol, machine to machine, is used with the possibility of either UDP or TCP for providing the process to process communication.

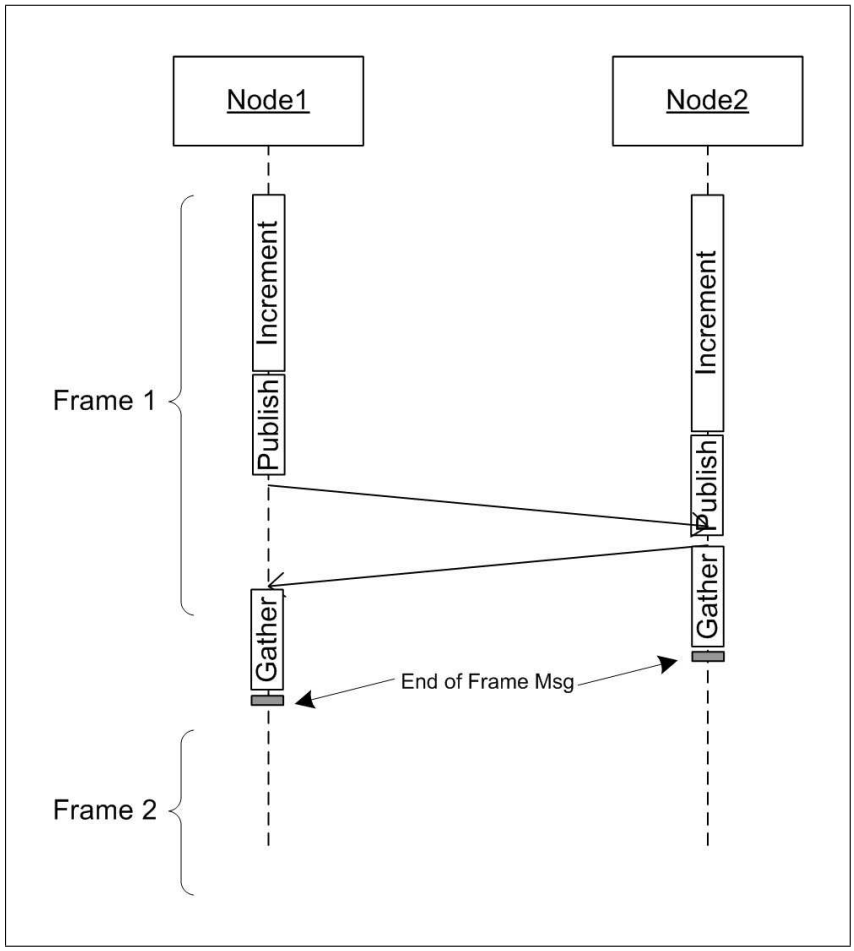

Figure 6.3: UML Sequence Diagram of Peer-to-Peer Message Passing and Simulation Synchronisation

TCP was chosen over UDP for TCP's already built-in reliable and stream based nature. TCP therefore guarantees message delivery and further that messages are received in the same order they were sent in. The TCP messaging implementation consists of two components. The first is an address translation from destination node number to destination IP and port before any message can be sent via TCP. This translation is pre-configured and fixed for each distribution configuration.

A disadvantage of using TCP instead of UDP is of coarse its higher communication overhead in terms of both bandwidth and latency. The second component of the TCP messaging implementation is therefore a two-tiered approach to lowering TCP message bandwidth and latency overhead. The first tier is to ensure that as much as possible of the TCP send and receive

overhead happens in parallel to the node execution. This is accomplished by increasing TCP's send and receive buffers to an adequate size such that the buffers have enough space for two simulation frames worth of data. This ensures that all TCP sends are non-blocking. It also facilitates the use of CPU time, spare CPU cycles or from a second CPU, to transport the data across the network to the appropriate TCP receive buffers for quick retrieval when needed.

The second tier takes control of the TCP message send times. TCP's Nagle algorithm tries to optimise bandwidth usage by accumulating sent messages in the send buffer until it is large enough to fill a TCP packet or until a certain time-out is reached. The unfortunate side effect of the Nagle algorithm is that control over message latencies is lost. To give control over the message latency back to the simulator the Nagle algorithm can be optionally and currently is disabled.

6.4 The GBADS Simulation Object Model

This section describes the common GBADS simulation object model within which the GBADS simulation is implemented. The *base object class* of the GBADS simulation is *Object*. The following paragraphs will discuss how Object differentiates into the *child objects* shown in Figure 6.4. The child objects then represent and implement the different aspects of the GBADS model as discussed in Section 5.1. The time synchronisation, the object increment, and the publish- and subscribe-logic is implemented in and inherited from the base Object. The child objects then add the properties and behaviour of the actual models.

Model is the base for all equipment and target platforms. Model generally defines every type of object that has a position, orientation and a GBADS type description. Model therefore adds some properties, but no behaviour yet and is in fact, like Object, a C++ pure virtual class. The child classes of Model implements, as shown in Figure 6.4, the additional properties and concrete behaviour required, for example, by the Target class of objects.

Service implements the high level simulation services such as *hight above terrain*, *line of sight* and *data logging* as mentioned earlier in this chapter. *Console* is the base for all OIL interfaces and also the *gateway* interfaces to other systems and simulators such as a flight seat.

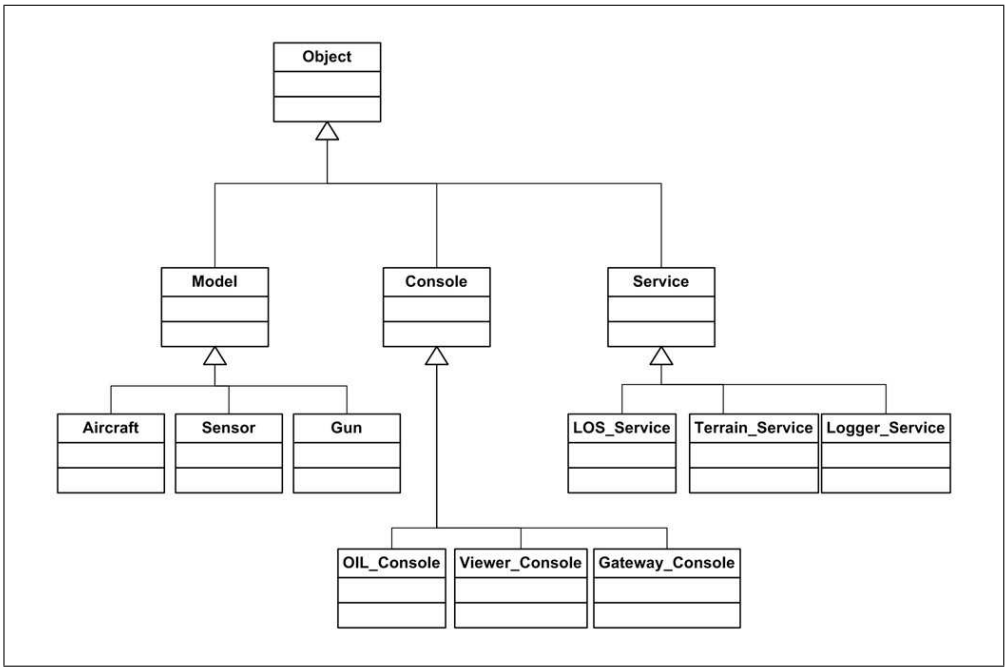

Figure 6.4: Simple UML Class Diagram of the GBADS Simulation Object Model

6.5 In Summary

The current discrete time architecture was presented. This includes:

- An overview of the layered architecture,
- the modelling approach in terms of Mealy and Moore type models,
- the publish-subscribe simulation model—including the communication framework and a CSP expression of it—layer,
- the peer-to-peer message passing and node synchronisation layer along with its TCP implementation, and
- the existing GBADS simulation object model.

The main goal of presenting the above was to familiarise the reader with the current discrete time architecture. The architecture—and simulation model—supports only Moore type models to allow a time synchronisation strategy which is independent of the execution order and node distribution of the models. The approach to modelling the GBAD system therefore takes this into account. The communication framework of the publish-subscribe

simulation model was discussed as a general distributed simulation communication framework. The distributed framework was also expressed using CSP and shown to be deadlock free with the aid of a CSP tool.

The peer-to-peer message passing implementation of the communication framework and the discrete time node synchronisation was then discussed with the aid of a UML Sequence Diagram. The details of the TCP messaging implementation and how strict control over message latency may be achieved was also given.

The discussion of the current architecture leads into the experimental analysis of the architecture's potential parallelisation speed-up—scalability—found in the next chapter.

Chapter 7

Performance Results and Analysis

This chapter analyses the performance and scalability of the current discrete time simulator. Initial TCP messaging and simulator architecture experiments were carried out during the design of the current discrete time simulator and new experiments were done for this dissertation after the simulator was implemented. The experimental setup and results of the performance analysis against the benchmark scenarios—discussed in Section 5.2—are then given. Some conclusions on the scalability of the current simulator are finally drawn.

7.1 Initial Messaging Experiments and Results

A performance test of a client-server distributed discrete time simulator was carried out during the early design of the current architecture. This had been done before the current hardware infrastructure was acquired. For the purpose of this dissertation, after acquisition of the current infrastructure, the inter-node communication was also characterised and is reported on. A synthetic peer-to-peer scalability *stress test* was also done and is reported on. The hardware infrastructure itself is also described.

7.1.1 Initial Client-Server Experiments

Initial TCP messaging tests that were done have been reported in Duvenhage and le Roux[50]. The tests included up to 6 computing nodes. These tests were done before the implementation of the current architecture and also

before the acquisition of the hardware infra-structure currently in use. A client-server discrete time simulator architecture was used.

At the start of each discrete time frame the server sends a work package of a certain size down to each of the six client nodes. The processing of the work package has a fixed execution time of 0.005 seconds which is 50% of the 100Hz discrete time frame, effectively leaving room for a 50% communication overhead. Once a node has finished its work it sends a result package, of the same size as the work package, back to the server. Communication overhead is defined here as all time spent before and after processing a work package. If the communication overhead stays below 50% of the frame—in other words, takes less than 0.005 seconds—then the discrete time frame can be executed within 0.01 seconds and the simulation may run at 100Hz or higher.

It is worth noting that the server and one node were co-located in a lab while the other five nodes were located in offices outside the lab. The server and first node were connected via the lab's local switch to the building's LAN while the other five nodes were connected directly to the building's LAN. The experiments were executed on a 100Mbit Ethernet infrastructure. The actual communication overhead, as a fraction of the measured execution time for a single discrete time frame, is shown in Figure 7.1. As long as this fraction is below 0.5 the communication overhead takes less than 0.005 seconds and the 100Hz frame rate can be achieved as explained in the previous paragraph. The graphed results show that this is indeed the case for work package messages smaller than 3kBytes.

For six nodes and a 3kByte work package the bandwidth from the server amounts to $6 * 3kBytes/0.01s = 18kBytes/0.01s = 1800kBytes/s$ *on average*. Similarly, the bandwidth back to the server is also an average of $1800kBytes/s$. The required server bandwidth of $1800kBytes/s$ in each direction is well within the bandwidth performance of 100Mbit Ethernet (i.e. 100Mbit/s > 10MB/s > 3600kB/s(half-duplex) > 1800kB/s(full-duplex)) and it was thought that the $3kBytes$ per simulation frame would have been more than enough for the GBADS scenarios simulated at the time.

In the GBADS simulation, however, the result package of a node would contribute to the contents of a future work package of not simply one node, but potentially of many nodes. The decision to rather implement a peer-to-peer architecture was, therefore, made on the idea that a peer-to-peer architecture would prevent a future bandwidth bottleneck at the server and, more importantly, avoid the double latency of communicating a result via a server to a destination node.

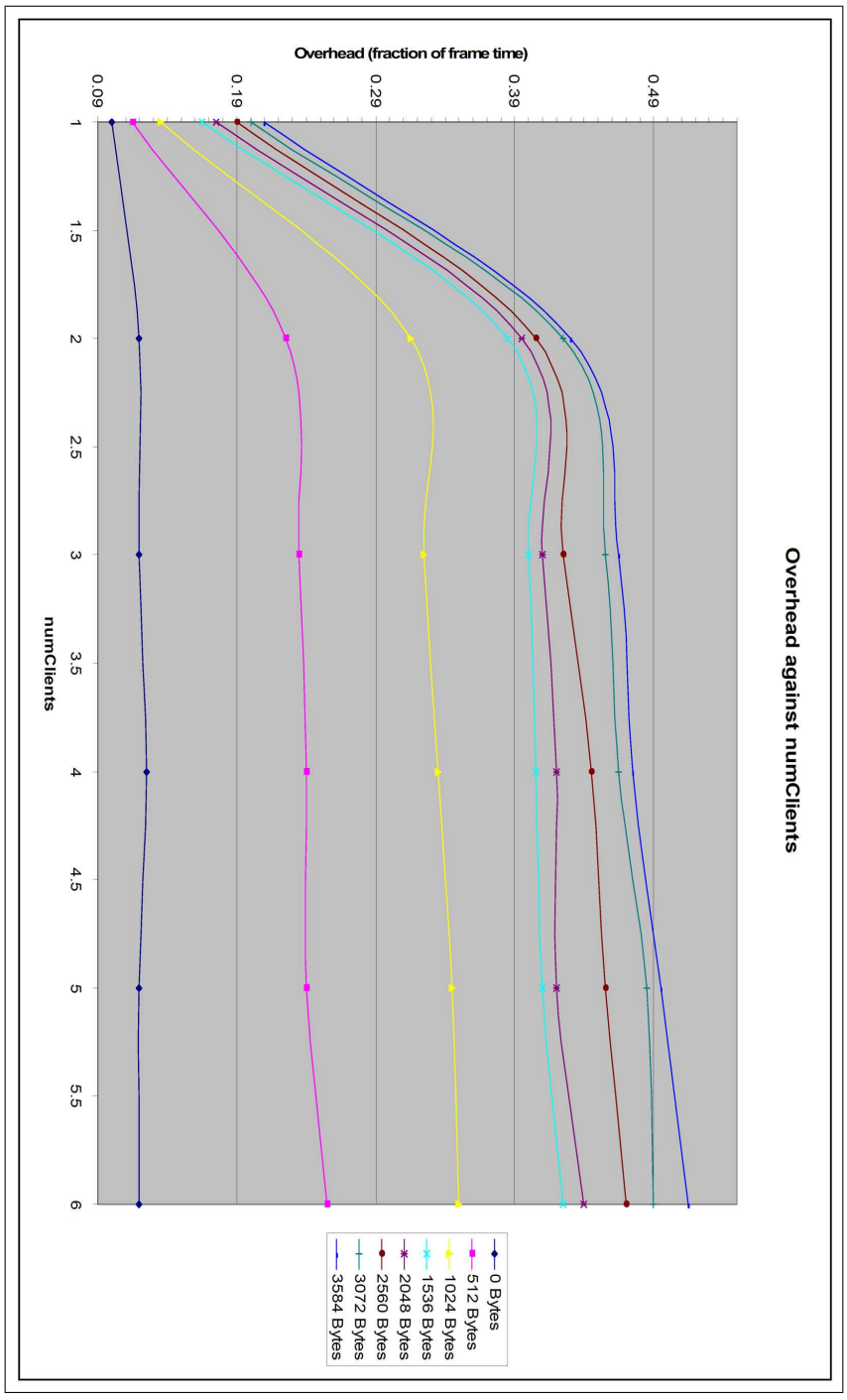

Figure 7.1: TCP Communication Overhead as Fraction of Time Frame against the Number of Clients and the Message Size[50]

Figure 7.2: The Hardware Infrastructure for the Current Distributed Discrete Time Simulator

The relatively constant positive gradients of each of the *overhead* against *number of nodes* graphs—ignoring the first node that had a direct connection to the server—indicate the communication overhead will rise to 0.005 seconds at points which are dependent on the work package size and the number of nodes. The locus—set of all (number of nodes, overhead) pairs/points—at which this happens is known as the real-time scalability ceiling of the simulator.

It is important to note that a trade-off exists between the work package size and the number of distributed processing nodes that may be used. Determining the maximum work throughput of such a simulator is therefore an optimisation problem of the number of nodes versus the work done per node.

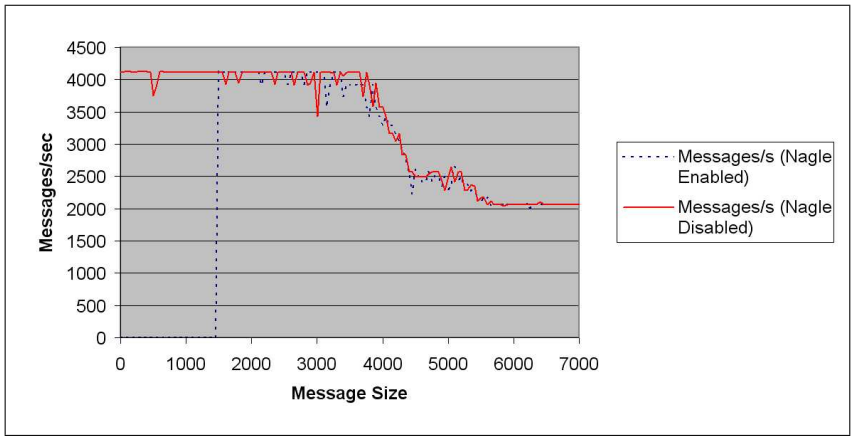

Figure 7.3: Messages/second to Message Size Trade-Off of a TCP Communication Channel

7.1.2 The Current Hardware Infrastructure

The hardware infrastructure that was finally acquired for the distributed discrete time simulator consists out of a transportable cabinet, shown in Figure 7.2, with seven rack mounted processing nodes. Each processing node is a standard PC with a 3.0 GHz Pentium 4 CPU and 2GB of memory. The processing nodes are clustered with a commercial off the shelf Gigabit Ethernet infrastructure. The Gigabit switch with the lowest latency of the set of commercial switches analysed was chosen. A 24-port network switch was chosen to support the connection of OIL and HIL systems and for potential future expansion of the cluster.

With the custom simulation communication layer and simulator architecture, but commodity hardware, the cluster may be classified as a cost-effective class 1 Beowulf cluster running Windows XP sp2 as the operating system.

7.1.3 Characterisation of the TCP Gigabit Ethernet Infrastructure

The performance of the current Gigabit Ethernet network infrastructure is characterised. The characterisation is done between two of the processing nodes—A and B—connected with TCP and using the same message structure that was used in the initial client-server tests. The messaging architecture is also similar to what is used in the current simulator architecture. Each message has a 4 byte header which contains the message size and is followed by the message payload. This message structure necessitates that the receiver reads the message header first to get the payload size. The message's actual

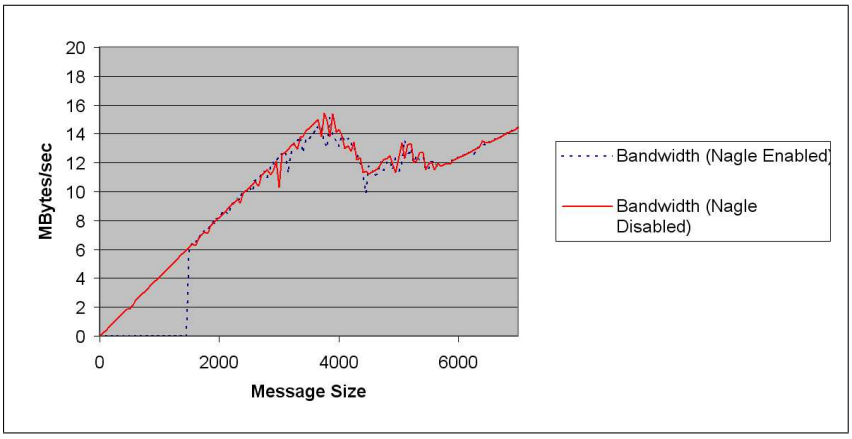

Figure 7.4: Bandwidth to Message Size Trade-Off of a TCP Communication Channel

payload is then read as soon as the entire payload has arrived in the TCP read buffer.

The characterisation results were arrived at by sending a message from node A to B. Once B has received the message it is bounced back to A. Before sending another message, A waits until it has received the previous message back. For this experiment A starts with a small message size and incrementally increases the size of the message by 100 bytes until a maximum of 7 kBytes at which point the experiment ends.

To get good statistical results, for each message size, the messages per second value (i.e. performance) is derived by measuring the time taken to transmit 1000 messages. Figure 7.3 shows A's sent messages per second throughput against message size. Figure 7.4 shows the related bandwidth—also only measuring the sent messages from A—against message size.

The performance of the differently sized messages show good local similarity in that the performance results of one message size to the next is always very similar. Due to this local similarity the results are expected to be a fairly good statistical sample—containing little noise. The trend of the performance graphs is therefore expected to be due to the operation and implementation of the TCP stack—possibly the flow and congestion control—and the Ethernet hardware.

It should be noted that the round trip test was specifically set up to characterise *single* message times. Sending bursts of messages would have similar bandwidth performance to sending larger messages since, for example, a second and third message in a burst may potentially be concatenated to the first and be sent before the response on the first message is received back. However, communication patterns such as request-reply and those within

coupled component networks can in general not take advantage of this for all message types.

The goal of the TCP messaging layer of the discrete time simulator now comes down to transporting a certain number of messages per simulation frame between the processing nodes. Figure 7.3 shows that to reach a certain number of messages per second the message size cannot exceed a certain value. The figure also shows that the messages per second throughput of a node has an upper limit irrespective of the message size. This upper limit is about 4100 messages per second—or just 41 messages equally spaced over the 100Hz simulation frame. This presents a very hard scalability ceiling and is imposed by the communication hardware in use.

In both Figure 7.3 and Figure 7.4 the dotted line graph shows the results with the TCP Nagle algorithm enabled, while the solid line graph shows the results with the Nagle algorithm disabled. As mentioned in the previous chapter, the Nagle algorithm caches short messages to be sent once a TCP packet is filled or once a certain time out is reached. The TCP packet size is 1.5kByte. It is only once packet sizes exceed this threshold that the TCP connection with the Nagle algorithm enabled starts having significant throughput. This is the reason why the Nagle algorithm is disabled in the current simulator. The drawback of disabling the Nagle algorithm is that poorer bandwidth efficiency results for bursts of individual small messages due to the TCP, IP and Ethernet package header overhead. The simulator may however take control of message grouping—and in fact does so—to overcome this drawback.

7.1.4 Initial Peer-to-Peer Scalability Test

A second set of scalability performance tests was also done on the current discrete time simulator. Instances of a *test* model were distributed over 1 to 7 processing nodes. The instances were fully connected with one another—i.e. the output of each test model was connected to the input of every other model. The number of objects per node were limited so that system execution performance is always just within real-time. The details are described by Duvenhage and Kourie[51], but the main result is shown in Figure 7.5.

Two graphs appear in the top part of Figure 7.5. For each of a given number of processing nodes ranging from 1 to 7, the bottom graph indicates the *maximum number of objects per node* that still achieve real-time execution and the top graph indicates the resulting *total number of objects in the cluster*. The solid lines show actual measured data and the dashed lines show an extrapolation of the measured data beyond 7 nodes.

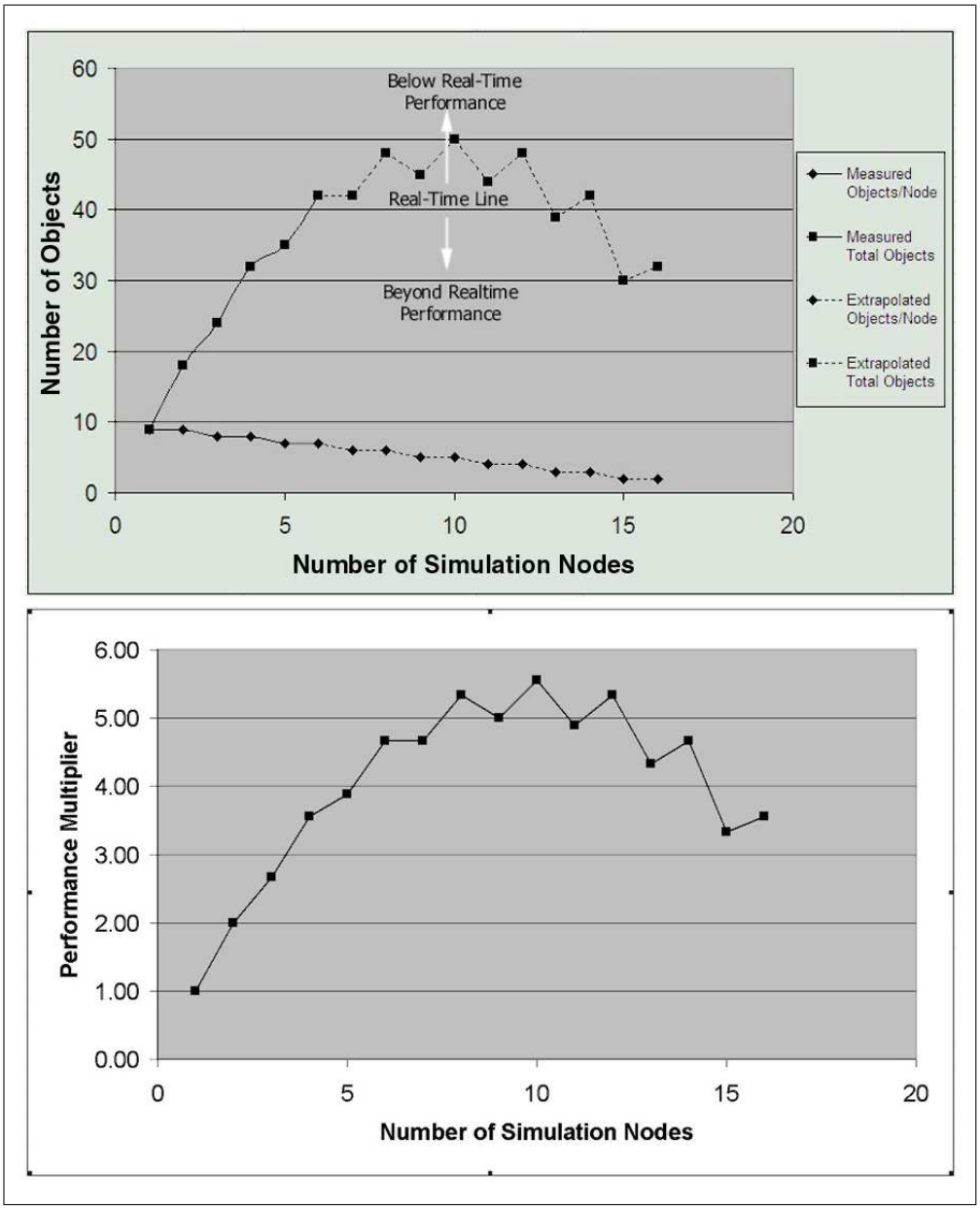

Figure 7.5: Number of objects and resulting performance multiplier against the number of processing nodes[51]

The measured *maximum objects per node* showed that, to add a node, the number of processes per node had to be, on average, decreased by 0.5 to still stay within real-time performance. This trend is stepwise linearly extrapolated—i.e. a decrease of 1 object per node for every two nodes added—to sixteen nodes under a simple assumption based on Amdahl's law[48] and the sequential nature of each node's communication channel[1]. The extrapolation of the *objects per node* results in the extrapolated *total objects* trend shown. The step-like nature on the *total objects* trend is due to the *objects per node* being equal across all nodes and also the stepwise extrapolation of the *objects per node* trend which is believed to be realistic.

The graph in the bottom part of Figure 7.5 shows the resulting so-called *performance multiplier* (also called the parallelisation speed-up) of the simulator for each distribution size. The parallelisation speed-up is defined by Wilkenson and Allen[52] to be

$$S(p) = \frac{Execution\ time\ using\ single\ processor\ system}{Execution\ time\ using\ a\ multiprocessor\ with\ p\ processors}.$$

For this arrangement of test objects (i.e. where model instances were fully connected with one another), a maximum parallelisation speed-up of 5.5 to 6 times is attained. This maximum parallelisation speed-up is reached on 9 to 11 nodes—adding more processing nodes would decrease performance, due to the added communication and time management overhead.

7.2 Benchmark Scenarios, Experiments and Results

The benchmark scenarios' *building blocks* were identified in Chapter 5. It was suggested there that the scale of each benchmark scenario finally be measured on the number of FUs in the scenario. Table 7.1 shows the list of objects in two of the smaller benchmark scenarios. The graph resulting from plotting the parallelisation speed-up against the number of nodes is expected to follow the same trend as the result of the initial synthetic test shown in Figure 7.5. The version of the discrete time architecture used was version 194 (8 June 2006; According to the root History.txt).

As already mentioned within Chapter 5 the model instances are currently distributed across the processing nodes in a random fashion, but assigning an equal number of objects to each node. The static load balancing experiments

[1] Amdahl's law states that: The speed-up of a program using multiple processors in parallel is limited by the sequential fraction of the program.

Table 7.1: List of Objects in Two of the Smaller Benchmark Scenarios

Scenario A	Scenario B
AIR_PICTURE_BOX	AIR_PICTURE_BOX
TEWA_BOX	TEWA_BOX
OIL	OIL
PERFECT_DR1	PERFECT_DR1
Gun1_Barrel1	Gun1_Barrel1
Gun1_Barrel2	Gun1_Barrel2
GFCS1	GFCS1
PERFECT_TR1	PERFECT_TR1
-	Gun2_Barrel1
-	Gun2_Barrel2
-	GFCS2
-	PERFECT_TR2
Target_Waypath1	Target_Waypath1
Target_Waypath2	Target_Waypath2
-	Target_Waypath3
-	Target_Waypath4

done by Duvenhage and Nel [49] indicate that a random distribution of a typical GBADS scenario comes, on average, within 85% of the best performing distribution.

However, in the analysis of the proposed hybrid modelling approach, to be discussed in Part III, the value of intelligently grouping the model instances into aggregated model types according to subsystems of systems will be demonstrated. (Model instances will be grouped into Fire Units (FUs), for example.) For this reason, the analysis of the current architecture is also done with objects being logically grouped into FUs before being distributed. The objects in the first 8 rows of Table 7.1 are typically instances of very computationally intensive models. These objects were placed on a processing node of their own to ensure that enough processing power is available to prevent them from causing a processing bottleneck. The FUs within each scenario are then distributed randomly to the remaining processing nodes.

The analysis of the benchmark scenarios was carried out with the contained objects distributed over 1 to 11 similar processing nodes. The first 7 processing nodes were located in the cabinet shown in Figure 7.2 and an additional 4 processing nodes were set up in the laboratory—directly connected to the cabinets 24-port network switch—for the purpose of these tests.

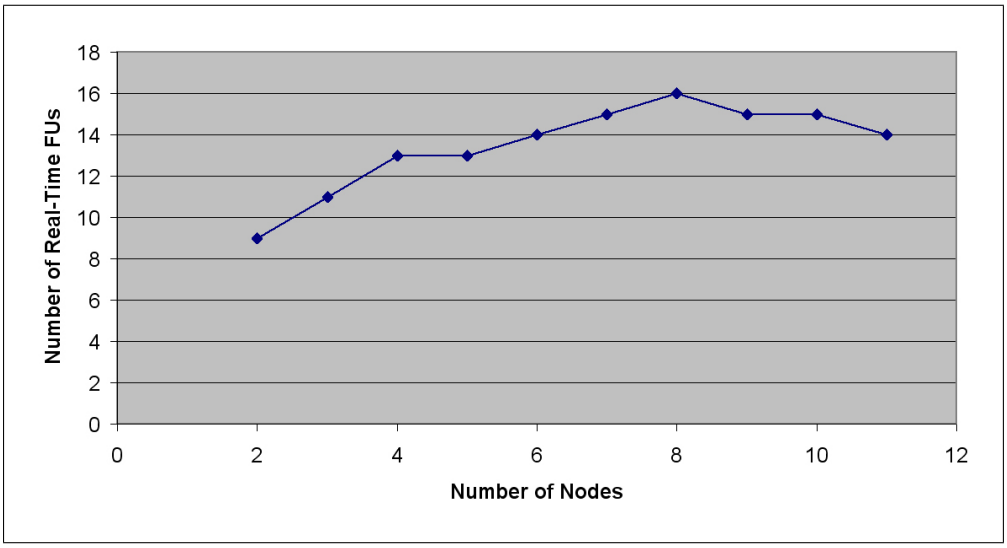

Figure 7.6: Measured Maximum Scenario Sizes for Real-Time Execution

The number of FUs that were allocated per node was set up to keep the distributed execution performance to just within real-time. Note, however, that in all experiments, whenever a scenario was amended by adding an FU, two additional targets were also inserted into the scenario as discussed in Section 5.2.

This resulted in the *number of FUs* against *number of nodes* real-time performance graph shown in Figure 7.6. The number of FUs per node across all the nodes were kept as similar as possible when adding FUs to each distribution. A maximum number of 16 FUs could be executed in real-time. This was achieved over 8 processing nodes. Using an FU count above the real-time performance line would result in slower than real-time performance and using an FU count below the real-time performance line would result in a faster than real-time performance.

The execution time of a scenario is however not linearly related to the number of FUs. The execution times for scenario sizes of 0 to 27 FUs were measured on a single processing node. These measured times were scaled by the duration of the scenario in simulation time to determine the so-called computational load of each scenario—i.e.

$$ComputationalLoad = \frac{Execution\ time\ of\ the\ scenario\ on\ a\ single\ node}{Duration\ of\ scenario\ in\ simulation\ time}.$$

Figure 7.7 shows the results obtained. See Section 2.2.5 for a discussion of simulation time.

Curve fitting on the data obtained suggests the following approximate relationship between computational load and number of FUs in a scenario::

$$ComputationalLoad = \left(\frac{FuCount}{8.11}\right)^2.$$

This means that a node loaded with a scenario of 8 FUs runs the given scenario approximately in real-time; a node loaded with a scenario of 16 FUs runs approximately 4 times slower than real-time; a node loaded with a scenario of 4 FUs runs approximately 4 times faster than the real-time requirement; etc. The quadratic nature of the relation is plausibly explained by the following: increasing a scenario by one unit means adding an FU *and* two targets—see Section 5.2; furthermore each FU in a scenario is required to process *all* the targets in the scenario. In effect, in a scenario of n FUs, these n FUs process $2n$ targets resulting in a performance relation proportional to $2n^2$.

The computational load value is exactly the parallisation speed-up, S, that real-time execution of the scenario would require. To see that this is indeed the case, observe that:

- if a scenario is executing in real-time on p processors, then the *duration of scenario in simulation time* exactly corresponds to the scenario's *execution time using p processors*; and
- under these circumstances, the values for parallelisation speed-up and computational load as defined above, are identical.

It should be noted that the current GBADS simulation capability need never support more than around 15 FUs during real-time execution. This requirement translates to a required parallelisation speed-up of $\left(\frac{15}{8.11}\right)^2 = 3.42$.

Using the quadratic computational load relation in Figure 7.7 allows the computational load, shown in Figure 7.8, of each real-time distribution to be estimated from the FU load of each distribution in Figure 7.6. Figure 7.8 is similar to Figure 7.5 in that both depict a performance trend that reaches a ceiling at a certain number of nodes (about 8 nodes in each case) before exhibiting decreased performance with increased number of nodes. This similarity validates to a certain degree the linear *objects per node* extrapolation used to derive Figure 7.5 as discussed in Section 7.1.4. Figure 7.8 shows that the maximum parallelisation speed-up—the scalability—achievable in the benchmark scenarios is about 4. Distributions over more than eight nodes do indeed, as expected from the initial scalability results shown in Figure 7.5, become less efficient.

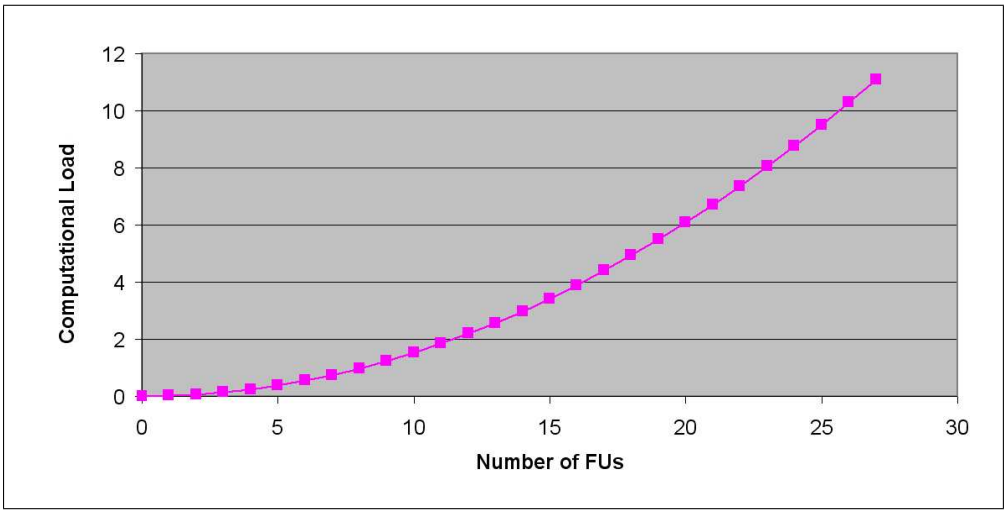

Figure 7.7: The Computational Load of Increasing Scenario Sizes

It is possibly important to emphasise here that the maximum parallelisation speed-up is only applicable to the real-time simulation of a scenario. 11 nodes can indeed handle larger scenarios than 8 nodes—albeit at below real-time—due to the 11 machines collectively having more memory and hard disk space for example. The current hardware infrastructure has however been configured with more than adequate memory and hard disk resources on each processing node to make the sharing of processor power for real-time execution the primary concern.

7.3 Analysis and Preliminary Conclusions

Figure 7.8 shows not only the measured scalability performance of the current discrete time simulator, but also the theoretical scalability performance of the ideal distributed simulator. An ideal distributed simulator has a parallelisation speed-up of $S(p) = p$ and an unbounded scalability given an infinite number of processing nodes. The difference between the current architecture and the ideal distributed architecture is due to the communication and time management overhead. The initial synthetic scalability test and the final benchmark results indicate that the current discrete time architecture does have a scalability ceiling. For the GBADS benchmark scenarios the scalability ceiling for real-time execution is a parallelisation speed-up of around 4. This ceiling is reached on a distribution of eight processing nodes.

Duvenhage and Kourie[51] argue that the scalability ceiling is due to the sequential communication channel of each processing node. One limit of

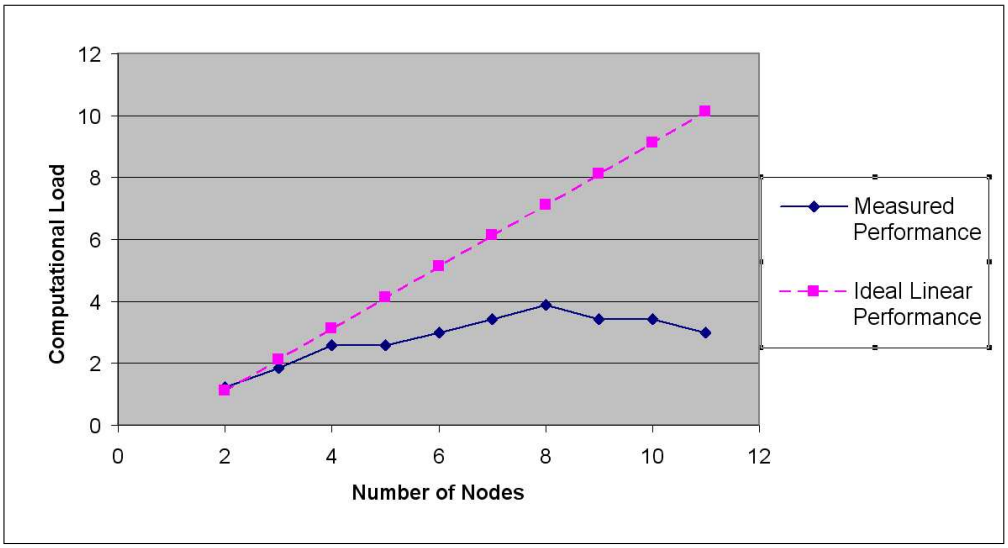

Figure 7.8: The Computational Load of Each Real-Time Distribution vs. The Computational Load of An Ideal Distributed Architecture

the sequential nature of the communication channel is the 41 message per simulation frame upper limit identified earlier in this chapter. A second limit is the bandwidth upper limit of the underlying network technology which is in the order of 100 Mbytes/sec for Gigabit Ethernet. A question the reader might ask is whether installing multiple network cards into a PC will improve the scalability of the simulator.

The answer is: Yes, multiple network cards may be installed into each PC, effectively connecting each node to its peers with two, three, four or more disjoint networks. This solution will improve the available bandwidth between peers by two, three, four or more times, but the number of network cards that may be installed into a standard PC—especially with a technology like Gigabit Ethernet that requires a lot of system resources—is often limited to three or less. The proposed hybrid discrete-event/discrete-time architecture discussed in the next chapter aims instead, drastically to improve the pattern of use of the sequential communication resources in order to steer the scalability behaviour towards the ideal distributed simulator case.

Part III

Migrating to a Hybrid Discrete-Event/Discrete-Time Modelling Approach and Simulator

Chapter 8

A Hybrid Modelling Approach

This chapter looks at applying a hybrid discrete-event/discrete-time modelling approach to increase the scalability of the current GBADS real-time simulation capability. In other words this chapter will—given an infinite supply of processing nodes—attempt to increase the size of scenarios that will run real-time to beyond the parallelisation speed-up limit of 4 that the current discrete time architecture was shown to have in Chapter 7.

Based on the technical risk identified in Chapter 3, the hybrid modelling approach does more than just embed the discrete time models within a discrete event architecture or DEVS BUS-like concept. For the implementation of the hybrid approach, this chapter revisits:

- the aggregation of,
- inter-object connections and
- time synchronisation between objects

to improve on the scalability of the discrete time simulator. The theoretical concepts have already been discussed in Chapter 2. In particular, a quantised system approach to discrete event representation of a dynamical system was mentioned. It was shown that such an approach is potentially more efficient than a DTSS (or at the very least, of similar efficiency) both in terms of the communication overhead *and* in terms of the model time complexity. It has also been shown that, in typical synthetic environment scenarios, optimistic time management has lower overhead than conservative time management.

A restriction of the proposed hybrid architecture is however, as mentioned, that the current conservative and discrete time models have to be reused in an economical way. For this reason, the following restrictions will be adhered to in the design of the proposed hybrid architecture:

- The internal system specification of each existing model will continue to be a DTSS, since re-articulating some or all of the models as DEVSs (or similar) would be prohibitively expensive.

- Conservative time management is kept, since the addition of roll-back to the current models in support of optimistic time management, implies changes to these models. This would complicate the migration process and increase its cost.

Beyond the aggregation and enveloping of the sub-system models into system level models, the migration of the modelling approach will therefore concentrate on—as discussed in the rest of this chapter—the following two discrete event aspects for *potential* application in the proposed hybrid architecture:

- The discrete event like quantisation of *selected* discrete time outputs including the application of advanced extrapolation such as dead-reckoning, and

- conservative *discrete event* time management instead of the current conservative *discrete time* time management.

8.1 Aggregation of Sub-System Models

The typical layout of a GBAD system of systems (of sub-systems) deployment was described in Chapter 5. Most of the current GBADS models are at the level of GBAD sub-systems of systems, such as the gun or FCS sub-systems of the FU. These models are typically modelled at a state transition system specification level or higher. At the GBAD system level the sub-system models are brought together to create system level models—FUs—at a coupled system specification level as shown in Figure 8.1. The GBAD system models are then coupled again to create a GBAD system of systems level model—the GBADS deployment—also at the coupled component system specification level.

The nature of the system of systems simulation experiments typically requires objectives and outcome measures at the GBAD system level—FU level. In such an *experimental frame* the output variables of sub-systems within the GBAD system level coupled component models (the system structural knowledge) is hidden from the simulation analyst. It is therefore argued that access to the output variables of sub-system models—the system structural knowledge—is superfluous when analysing only the system level outputs. An

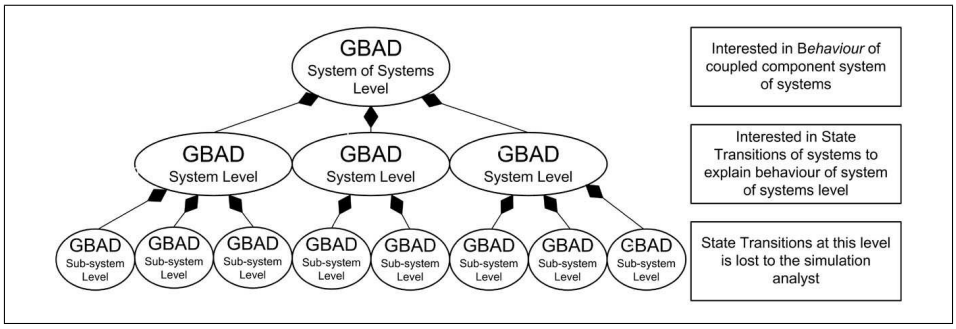

Figure 8.1: Double Structure Level of Discrete Time Simulator

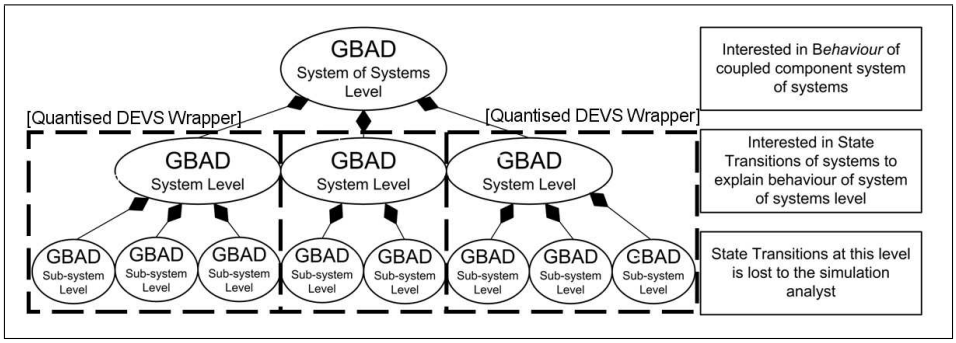

Figure 8.2: Double Structure Level of Discrete Time Simulator with Aggregation *Wrapper* Indicated

aggregated structure for the GBADS simulator with the superfluous structural knowledge hidden in a wrapper is shown in Figure 8.2. The *quantised discrete event* nature of the wrapper will be discussed in the next section.

It should be noted that the same sequential communication components that were shown to exist in the discrete time simulator also exist in a discrete event simulator. The aim is however to avoid activating these sequential components unnecessarily. *Aggregation of the DTSS models* is the act of explicitly hiding the *double layer* of intermediate GBAD system structural information within a discrete event model. The new GBAD system level discrete event model—the aggregated FU—is then a state transition system specification envelope which shields the model interconnect infrastructure from the communication overhead of the internal structure.

This aggregation of GBAD sub-system models into system level models has already logically been done in anticipation of the discrete event enveloping during the analysis of the discrete time architecture. This was done to be

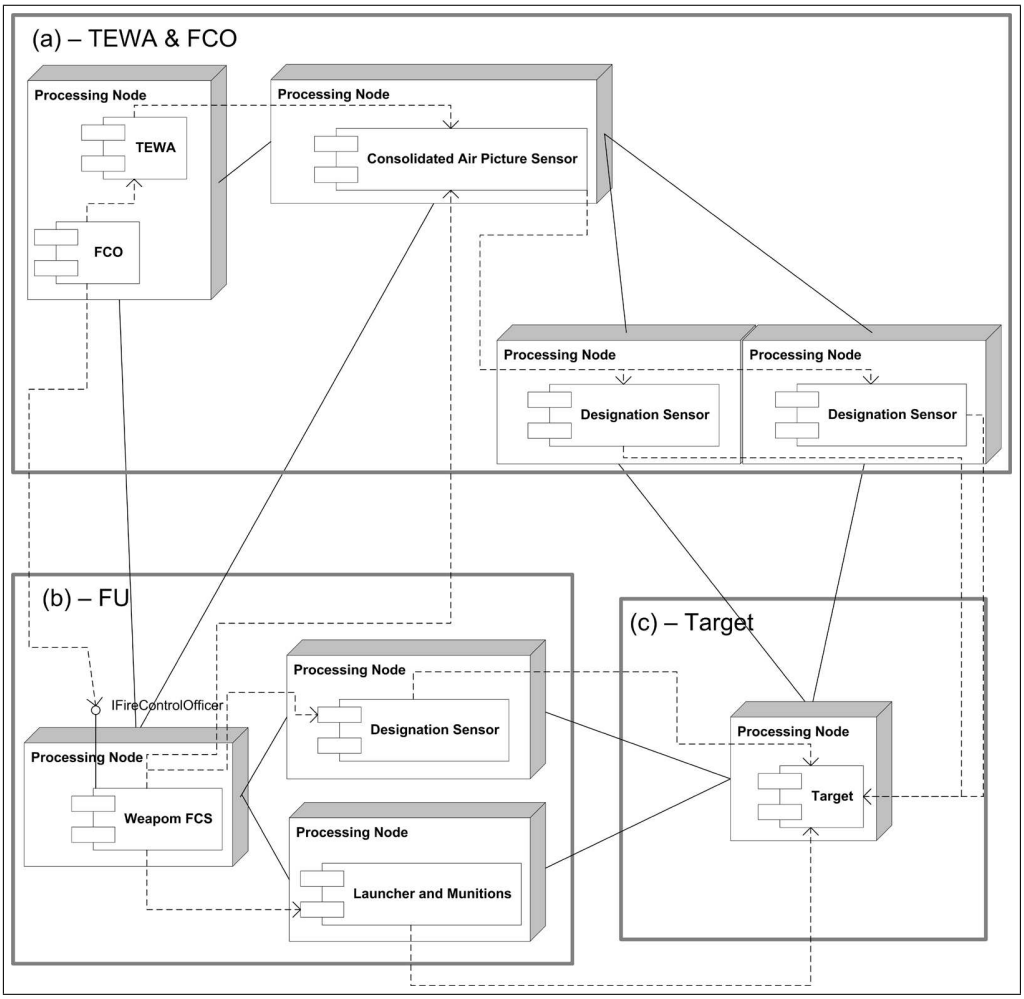

Figure 8.3: UML Component Diagrams (nested within their deployment diagrams) of the building blocks of a GBAD system of systems deployment

able to compare the scalability of the discrete time simulator with the scalability of the proposed discrete event simulator using similar GBADS scenarios in both. Figure 8.3—a reproduction of Figure 5.5—shows the UML component diagram of the *aggregated* system levels GBADS building blocks. The *use dependencies*—indicated by the dashed arrows—between system level building blocks indicate communication paths that could potentially result in network communication.

8.2 Output Quantisers and Quantised Integrators

In the previous section, aggregation was proposed to remove some of the internal FU communication overhead from the system model. The remaining system level communication—use dependencies—can be classified into two types: events (be it voice network events or data); and state-like information such as platform position, velocity and orientation. Events are already quantised. However, the discrete time state information—discrete time sampling of what would in reality be a continuous variable—may often be quantised further.

The approach followed is to wrap the aggregated system level models in discrete event envelopes, as was mentioned at the end of Section 2.4.1. Below, a discussion is provided of how an *output quantiser and quantised integrator pair* is used to connect each discrete time output to be quantised to each of its neighbouring inputs. More advanced second-order quantiser and integrator pairs may be used for outputs describing bodies in motion that have gravitational and other—possibly time varying—forces acting on them. *Dead-reckoning*—sometimes referred to as *active quantisation*—is also discussed.

Zeigler[27] and Zeigler et al.[15] define a Quantised System (QS) as having the same behaviour as a system—discrete time or continuous—*sandwiched* between input and output quantisers. According to Kofman et al. [21] a Quantised State System (QSS) is differentiated from a QS by the quantisation being performed using hysteresis. The hysteresis acts on the thresholds that define the triggers of significant events in order to ensure that a newly quantised dynamical system may be represented by a DEVS model[21]. Hysteresis has the effect of delaying the reversal of an event, limiting each event to represent a change during a finite time interval. A time discretised dynamical system has the required hysteresis for DEVS representation.

Additional discussions on the theory of quantised systems and related topics may be found in [19, 20, 26, 53, 21, 22, 23]. All of these resources also make note of the efficiency of a quantised system approach to the discrete event representation of a dynamical system.

8.2.1 Output Quantiser and Quantised Integrator Pairs

The primary goal of quantisation is to lower communication bandwidth usage. To accomplish this a model's inputs are fitted with quantised integrators and their outputs with quantisers to create a QS. Each communication chan-

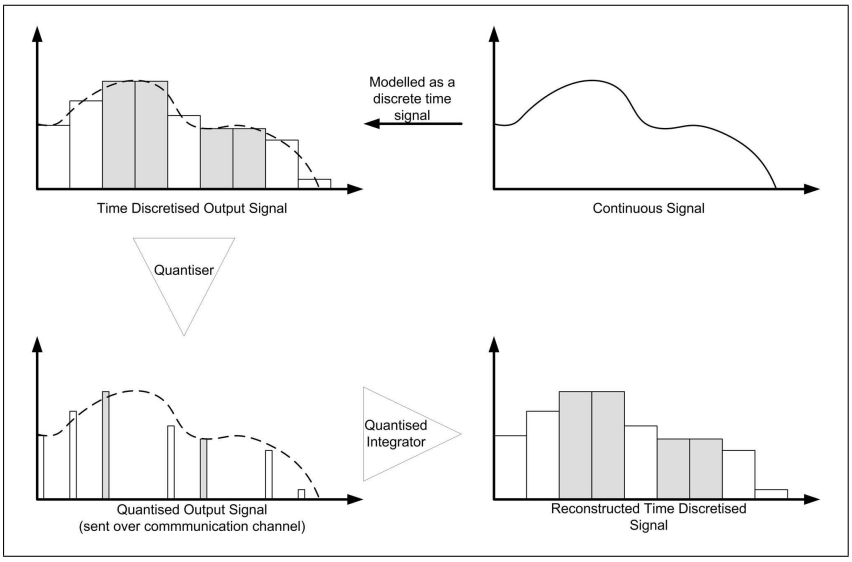

Figure 8.4: A signal modelled time discretised, then quantised and reconstructed by means of a QQIP

nel between a sender and a receiver of information is therefore fitted with a Quantiser and Quantised Integrator Pair (QQIP). A signal, time discretised and then quantised and reconstructed by means of a QQIP is shown in Figure 8.4.

The baseline for quantisation is what is called *non-predictive quantisation*. In the most basic case the quantiser uses pre-setup quantum boundaries that trigger *boundary crossing events*. The quantised integrator then reconstructs the input to a piecewise constant value. This type of reconstruction is adequate for discrete time sampled input.

Predictive quantisation[19][27] may be employed to further optimise a discrete event model's time complexity. Applying predictive filtering should therefore be considered in future when developing discrete event and DEVS models, but will not be discussed further here.

8.2.2 Dead-Reckoning

The goal of the dead-reckoning technique is to trade accuracy for lower communication bandwidth usage. This is, of course, similar to quantisation. However, dead-reckoning attains its goal in a smarter *active* way. Dead-reckoning is usually applied in the quantisation of the position—and possibly its velocity and acceleration—of a body in inertial or non-inertial motion.

The main difference from basic quantisation is that the quantised integrator also receives the algorithm to reconstruct/predict the input—to the quantiser—in between update events.

The dead-reckoning QQIP may apply a first order quantisation for following first order changes to a variable. An example is a body of mass following an inertial path. Second order quantisation may be applied for description of a body of mass under the influence of constant or time varying forces. One such an example is a ballistic munition that has a certain initial muzzle velocity and then gravity and drag forces act on it throughout its flight, giving it a curved ballistic path. Second order QQIPs is often used in the dead-reckoning.

An aircraft—for example—will, along with its position, make known to the radar how to best predict its path of motion up to x seconds into the future. The radar may then calculate for itself the aircraft's position as frequently as required. The aircraft will however keep track of where the radar thinks the aircraft is, as the aircraft knows what prediction algorithm the radar is using. As soon as the aircraft's actual and predicted positions are outside a predefined error boundary of each other, the prediction at the receiver has become *stale*. Once this happens the aircraft *actively* refreshes its current position and prediction method to the radar—the aircraft sends a *path update* event. Both the original discrete time sampling and also the dead-reckoning error threshold result in the required hysteresis in generating quantised events to ensure a valid DEVS model.

In the process of correcting the error between the predicted and actual aircraft state, a path update event may possibly cause a discontinuity—a jump—in the aircraft's position or its time derivatives at the receiver. The first time derivative of position is velocity, for example. Such a discontinuity may be tolerated by the receiver's model of the world or may be hidden from the receiver by a continuous—to any number of derivatives—transition between the stale path and the freshly predicted one. The aircraft's path update events to ALL subscribers may be synchronised if the error threshold is the same for all subscribers.

8.2.3 Implications of Quantisation

Quantisation lowers communication bandwidth and potentially model time complexity at the price of a potential increase in the accumulated error of the simulation. Zeigler[27] and Zeigler et al.[15] have derived the upper bound on the accumulated error of closed loop DTSS simulation and also closed loop quantised DEVS simulation from the theory of quantised systems. The same accumulated error results have also been arrived at experimentally by

Zeigler et al.[54] and Wainer and Zeigler[53], among others, who have done a cost/benefit analyses of reduced communication bandwidth and increased error due to quantisation.

The currently existing discrete time models have been conceptualised and developed with high resolution discrete time management in mind. Many of the models have therefore been validated within the 100Hz frame rate of the discrete time simulator. This makes them particularly sensitive to the errors introduced by QQIPs.

Many of the sensor models, for example, were built and evolved to rely on a 100Hz target update rate. The models of the tracking filters within each sensor were, as mentioned in the introduction, based on actual engineering text book and numerical method solutions. The 100Hz target data made it possible to build these types of *real world* solutions instead of the often more daunting behavioural solutions.

For the purpose of this dissertation the quantum levels that will be used for the quantisation as well as the threshold values for the dead-reckoning *path update* events will—as far as possible—be made equal to the quantum levels that were implicit in the discrete time execution of the models. The high level behaviour and selected outcome measures of the benchmark scenarios will be compared to the discrete time execution results to assess the impact of quantisation on the simulation outcome, but an accurate measure of the per model and total accumulated error for different quanta is left for future work. Choosing the quantum levels as discussed above can potentially result in quanta that are smaller than required which would in turn result in a sub-optimal bandwidth improvement. The nominated future work would improve on the optimal choice of quanta for the different communication channels and therefore optimise the bandwidth usage. It is important to note that the migration process to a real-time architecture should be driven by the requirement to achieve the same accuracy as before, but to do it with greater efficiency.

According to the DEVS literature, the computational time complexity of quantisation and quantised integration is also relatively small[15]. The quantisation time complexity overhead for the hybrid simulator is therefore assumed to be negligible.

8.3 Efficient Discrete Event Time Management

The current discrete time simulator uses a conservative time management approach that enforces causality in 100Hz time slots. Within the aggregated discrete event modelling approach a more efficient conservative time management algorithm may be possible. Null-messages with look-ahead—discussed in Section 2.4.2—is the obvious candidate for the discrete event conservative time management.

The reason why null-message time management is advantageous is because of its potential to be more efficient than discrete time management. To accomplish this does however require accurate lookahead times.

Fujimoto says on page 87 of [10] that development of a simulation and its models is affected if it is known ahead of time that accurate lookahead is important. The modelling assumptions are then specifically developed to be able to give better look-ahead times. The reverse also follows: that attempting to add lookahead to the existing discrete time simulation and models will result in a simulation that potentially has a shorter lookahead capability than possible within the chosen experimental frame. If this is the case, the null-message overhead will be far from optimal because the efficiency of null-messages is, as discussed, dependent on the lookahead times.

In the current situation, migrating to a null-message time management approach has an additional *disadvantage*. Many of the existing models already have event-like interactions with each other. If the time management is migrated away from the current discrete time approach then many more models will have to be updated than the ones identified for output quantisation.

Fujimoto [10] gives a further two *disadvantages* of using time management that relies on lookahead predictions:

- Changing the model slightly or adding new components may severely impact the lookahead calculations and therefore the simulation performance.

- The drive to be able to produce accurate lookahead times also makes the modelling effort somewhat dependent on the simulator architecture. This, in general, causes migration difficulties.

The cost for efficiently implementing null-message with lookahead time management seems prohibitive. For the purpose of migration to and the analysis of the hybrid architecture the discrete time conservative time management is therefore kept. This allows the models to only be adapted as required for quantisation purposes. Additionally, the discrete time frame

provides a controlled signal to sample efficiently the input of the quantised integrator for potential path update events.

8.4 Implementation for Running the Benchmark Scenarios

The current discrete time architecture was modified to contain quantised discrete event inputs and outputs to selected models. The discrete time and conservative time management aspects of the architecture is kept as explained in the previous section. This migration step towards a DEVS simulator is therefore similar to a *quantised DTSS* modelling approach, discussed in Section 16.4 of Zeigler et al. The quantised DTSS modelling approach has similar advantages to the DEVS modelling approach—compared to standard DTSS—in terms of communication overhead. Quantised DTSS does however not have the model time complexity advantages and accuracy of quantised DEVS for the same quantum size.

The aggregation boundary of each GBAD system level model—such as the FU—was kept a logical one as explained in Section 8.1. The state transition discrete event envelope around each GBAD system level model is therefore also a logical one, being set up by the aggregation of each GBAD system level model. The sub-system model inputs and outputs that are on the boundary of the envelope—such as the FU's launcher and munition input *use dependency* or the TEWA&FCO's designation sensor output shown in Figure 8.3—were candidates for quantisation. The communication overhead between GBAD sub-system level models—within an FU building block for example—that are located on the same processing node is minimal. This is due to such communication basically being a memory access.

The only use dependencies—indicated by the dashed UML arrows in Figure 8.3—that are chosen for application of QQIPs are the dependencies on the radar state output, the air-picture output and the target model output. These are the dependencies that require the high time resolution *state* information links between the aggregated system level models. The target flight profiles were defined as straight and level. This allowed—for analysis purposes—a simple piecewise constant QQIP to be applied to each dimension of the position, velocity and orientation of the state information, but with position extrapolated by the velocity in between path update events. The quantised *path update* events are generated at a fixed 10Hz and reconstructed at the quantised integrator—receiver—side. The reduction of the information update rate from 100Hz to 10Hz allows the quantised *use dependencies*

to only use 10% of the bandwith of the unquantised *use dependencies*.

8.5 In Summary

This chapter looked at applying a hybrid discrete-event/discrete-time modelling approach to increase the scalability of the current GBADS real-time simulation capability. The two aspects finally addressed were:

- The aggregation of the GBADS sub-system level models into system level models such as FUs and
- the quantisation of selected *use dependencies* between the aggregate models.

Figure 8.3 shows the UML component diagram of the *aggregated* system levels GBADS building blocks. The use dependencies—indicated by the dashed line arrows—between the system level building blocks indicate the communication paths that could potentially result in network communication between processing nodes. Piecewise constant QQIPs were finally applied to the inter-building-block use dependencies for the experimentation with and analysis of the hybrid architecture. The results of this experimentation and analysis are reported on in the next chapter.

Chapter 9

Hybrid Simulator Analysis and Results

This chapter analyses the new hybrid modelling approach and simulator, and then presents the performance results and some preliminary conclusions. The high level behaviour and time-line of the scenarios will be compared across the discrete time and hybrid simulators. The *number of targets shot down* will also be compared across the simulators for basic validation purposes.

9.1 Benchmark Experiment Results

The benchmark scenarios are exactly the same scenarios that were used in Chapter 7. As before, the time taken to run a scenario on *a single node* is related to the number of FUs in that scenario approximately by a factor of $\left(\frac{FuCount}{8.11}\right)^2$, referred to as the scenario's computational load. This means that a node loaded with a scenario of 8 FUs runs the given scenario approximately in real-time; a node loaded with a scenario of 16 FUs runs approximately 4 times slower than real-time; a node loaded with a scenario of 4 FUs runs approximately 4 times faster than the real-time requirement; etc.

The computational load—which is also the parallelisation speed-up values required for real-time execution of these scenarios—of various FU counts, are therefore reflected by the same graph found in Chapter 7. This graph is reproduced here as Figure 9.1 for ease of reference.

The measured maximum number of FUs that could execute at real-time on various distribution sizes was measured as before. The results, shown in Figure 9.2, are however different from what was found for the discrete time simulator, shown in Figure 7.6. Unlike the real-time FU results of the discrete time simulator, the results of the hybrid architecture show no signs of

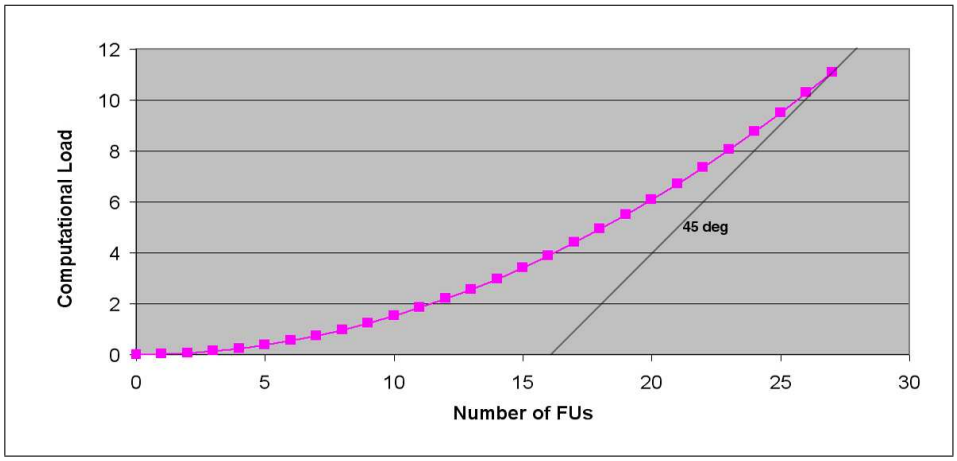

Figure 9.1: The Computational Load of Increasing Scenario Sizes

an upper bound on the number of real-time FUs for at least the 11 processing nodes available for the experiment. In fact, it is possible to fit a curve of the form $Number\ of\ FUs = k\sqrt{Number\ of\ Nodes - x_0}$ to the measured data which, if accurate, has no upper limit.

Again using the quadratic computational load relation shown in Figure 9.1, allows the parallelisation speed-up—a.k.a. the computational load—of each real-time distribution to be calculated. The resulting parallelisation speed-up performance of the hybrid simulator is shown in Figure 9.3. Note that a parallelisation speed-up ceiling *could not be found*—see the next section—over the 11 processing nodes that were available.

Also note that from Figure 9.1 it seems as if beyond scenario sizes of approximately 27 FUs, the computational load to FU gradient will rise above 45 degrees. This may be gauged from the 45 degree gradient line drawn in the figure. Beyond a gradient of 45 degrees at least one FU—and potentially every FU—has a computational load above one. The implications of this are discussed in the next section.

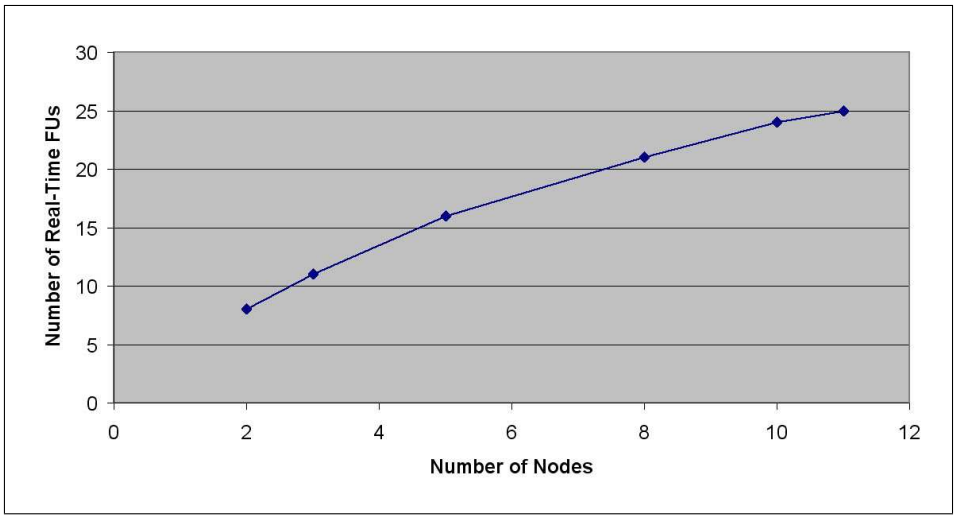

Figure 9.2: Measured Maximum Scenario Sizes for Real-Time Execution

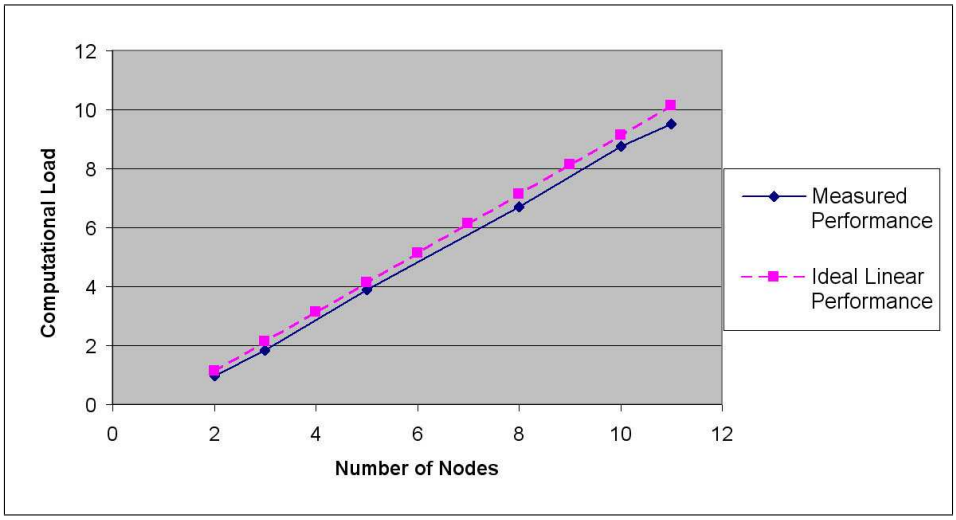

Figure 9.3: The Computational Load of Each Real-Time Distribution vs. The Computational Load of An Ideal Distributed Architecture

9.2 Analysis and Preliminary Conclusions

Figure 9.3 shows that the hybrid simulator's performance does indeed approach that of the *ideal distributed simulator*. This is the case up to at least the 11 processing nodes available for the experiment. The maximum measured parallelisation speed-up is therefore approximately 9.5, but with no sign of a turning point or scalability ceiling.

The high level behaviour and time-line of the scenarios executed on the hybrid simulator were validated against that of the results of the discrete time simulator. The number of targets killed also agreed between the two simulators allowing a very basic validation of the quantisation approach.

It was mentioned at the end of the previous section that beyond scenario sizes of approximately 27 FUs, more than one processing node would have to be added for each additional FU to still support real-time execution. This is rather important as it implies that one or more FUs would have to be split up and distributed over multiple processing nodes. Splitting an FU up into its parts will introduce a tear into the discrete event envelope constructed around each FU—discussed in the previous chapter. This is expected to change the parallelisation speed-up results somewhat for scenarios larger than the ones currently analysed. It in fact points to a potential new stumbling block when migrating to GBADS scenarios that require a parallelisation speed-up of above 11.

Part IV

Conclusion, Future Work and Final Remarks

Chapter 10

Conclusion

Distributed simulation is fast becoming the trend in simulation due to the advantages of sharing simulation resources across large distances and the potential parallelisation speed-up that distributed parallel simulation offers. A disadvantage of distribution is of coarse, that instead of only having a trade-off between accuracy and time complexity—as is the case with single machine numerical methods solutions—*a trade-off between accuracy and communication bandwidth* also exists.

The Discrete Time Simulator

The current distributed discrete time architecture was shown to be capable of providing a real-time GBADS simulation capability by meeting the 15 FU—a parallelisation speed-up of 3.42—requirement of a typical large GBADS scenario. The analysis results did however also reveal that the architecture has a maximum parallelisation speed-up of approximately 4—even when given an infinite supply of processing nodes. The goal of this dissertation was to overcome this scalability ceiling of 4 as it is just beyond the current use of the simulator—a little too close for comfort. The goal had to be accomplished within the constraint of re-using the existing models, though.

Theory of Modelling and Simulation

The M&S framework, principles and theory presented in Chapter 2 were applied in the description and analysis of the current discrete time architecture. This made possible the application of existing theories and conjectures from the M&S community. The current discrete time architecture's scalability ceiling was accordingly argued to be due to the high time resolution discrete time approach followed when originally developing the GBAD system of systems (and sub-systems) models.

It was also shown and mentioned throughout the dissertation that using a discrete event representation of a discrete time system is potentially more efficient than a DTSS—or, at the very least, of similar efficiency. To overcome the scalability ceiling of the current architecture it was therefore proposed to incorporate the advantages of a DEVS into a new hybrid discrete-event/discrete-time simulator. Chapter 3 has however raised some risks in the direct migration to a discrete event interoperability architecture such as HLA. Simply embedding the discrete time models within a discrete event architecture was shown to still bring about a technical risk in maintaining the discrete time simulator's high real-time frame rate *in distributions beyond a few processing nodes*. Also, re-articulating the GBAD system of systems models in a DEVS presents an economic risk.

The Hybrid Discrete-Event/Discrete-Time Simulator

The economic risk of using an interoperability standard results from the fact that there is currently no national interoperability drive. It remains an issue to be addressed when wanting to apply or migrate to an interoperability standard such as HLA or DIS and some references on various approaches have been included. The cost of re-articulating existing models from their DTSS to a DEVS to fully exploit the advantages of DEVS in a hybrid simulator is also prohibitively high. To mitigate this risk, a custom hybrid discrete-event/discrete-time architecture is applied and the existing models are not re-articulated in a DEVS, but rather aggregated and wrapped in discrete event envelopes. The technical risk of embedding the high frame rate discrete time models in discrete event envelopes, for distributions beyond a few nodes, are addressed by applying a quantised system approach in the discrete event representation of the envelope interfaces.

The proposed hybrid architecture was implemented, analysed and shown to have a parallelisation speed-up of approximately 9.5 for distributions over the 11 processing nodes available. A graph of the parallelisation speed-up is shown in Figure 9.3. A turning point in the parallelisation speed-up could not be established over only 11 nodes. It is therefore estimated that the parallelisation speed-up of the proposed architecture may be increased beyond 9.5 for distributions over more than 11 processing nodes and therefore that the scalability is 9.5 or higher. Figure 9.3 also shows the parallelisation speed-up of an ideal distributed simulator. Notice that the hybrid simulator approaches the performance of the ideal distributed simulator much better than the current discrete time simulator analysed in Chapter 7. This is due to the network bandwidth requirement being sufficiently reduced. As a result, the architecture is processor limited as in the ideal distributed simulator

case.

The Next Parallelisation Speed-Up Stumbling Block

It has however been noticed that, for scenario sizes requiring a parallelisation speed-up of above 11, the GBAD system level aggregation can no longer be applied. This is due to the quadratic nature of the computational load of a scenario which implies that at some point the computational load of a single system level model—such as an FU—will be larger than one and will in itself require multiple processing nodes to keep up to real-time. It indicates to the potential existence of a new stumbling block for the real-time execution of very large scale scenarios—above 27 FUs—where:

- System level models will have to be split up again into their composing sub-system models—re-exposing to the communication infrastructure the model detail that was hidden in the discrete event envelopes—and
- sub-system models will possibly have to be split up further into sub-systems of sub-systems—exposing even more model detail to the communication infrastructure.

Evaluation of UML and CSP

It is the author's opinion that UML and CSP were both useful in the representation of the different aspects of the GBADS model and the simulator architectures. The UML diagrams offered a comprehensive way to understand and visually communicate the GBAD system, simulation capability and selected aspects of the simulator architectures. The mathematical and executable nature of CSP was in turn found very useful for representing specifically the simulator architectures. Using a CSP-compatible tool it was possible to automatically prove that the discrete time simulator is deadlock free.

Hardware Infrastructure Upgrades

From the analysis results, some conclusions may be drawn on the potential value in terms of the resulting performance increases when upgrading different aspects of the hardware infrastructure. When the CPUs of the current discrete time architecture's processing nodes are upgraded—say to twice the processing power—the communication infrastructure will still limit the scalability of the architecture and the new processors will sit idle, waiting for

network data. The maximum parallelisation speed-up would in actual fact decrease, as double the work could then be done on a single node while the distributed execution performance would not be significantly affected.

Upgrading the communication infrastructure—on the other hand—will have the effect of lowering the communication overhead and could potentially increase the scalability of the current discrete time and proposed hybrid architectures. Looking at the matter slightly differently: If the parallelisation speed-up of the simulator resembles that of the ideal distributed simulator—$S(p) = p$ shown in Figure 7.8 and Figure 9.3—then the communication overhead is small and upgrading the processing nodes' CPUs will have greatest effect. If the trend of the parallelisation speed-up resembles that of the current discrete time simulator—a bounded $S(p)$ as shown in Figure 7.8—then upgrading the communication infrastructure will potentially have the greatest effect.

Migrating to a simulator architecture that has more ideal performance therefore has the additional advantage: namely, that then upgrading the infrastructure's CPUs will be a relatively inexpensive way of further improving simulator scalability, compared to upgrading to a new networking technology.

How To Do Simulation

A quantised discrete event approach to modelling has been shown to offer advantages in terms of time complexity of the model. The quantised approach has, however, had a significant impact on the scalability of the distributed GBADS simulator. This is due to the optimised use of the communication bandwidth between distributed nodes. Such an impact in applying the theory of modelling and simulation clearly indicates that the study of numerical methods for potential application to simulation is only one of the steps towards the study of how to do simulation.

Further Advantages of the New Hybrid Architecture

This work will create new opportunities within the SANDF simulation capability to potentially move to very large scale simulation environments and secondly interoperate these environment with other simulation communities. The interoperation with other simulators and simulation communities is facilitated by:

- The addition of discrete event communication capability,
- improved scalability has the side affect of allowing the use of lower bandwidth connections such as telephone lines, and

- this dissertation, which makes clear the common language and M&S framework used by the global military simulation community which is essential for interoperation.

Chapter 11

Future Work

Migrating To A New System Specification

The proposed hybrid simulator still employs discrete time models as per requirement to reuse existing models. This does however mean that the time complexity of the models and therefore the efficiency and scalability of the simulator will increase as the discrete time models are internally migrated to quantised discrete event modelling approaches. Researching a cost effective way of doing such a migration to a new system specification could lead to further increases in the simulator's scalability in future.

Inclusion of Pure DEVS Models

The migration to a new model system specification may be done for all the models in one re-articulation sweep or it may be done only for new and existing models that would have a significant impact on simulation performance. The current implementation of the hybrid simulator that was used for the benchmark tests in Chapter 9 did however reuse the internal discrete time management approach of the discrete time simulator. This was done for ease of implementation, given that only a small set of the discrete time models required quantisation and the rest could then be reused without any modification. This implementation resembles that of quantised DTSS. Future work could look at the best way to include a pure DEVS model into the currently discretised time line.

The Closed Loop Error Behaviour

A study has not been done in this dissertation on accurately finding the accumulated sub-system and system of systems model errors—the closed loop error behaviour—for different quanta. It has been shown from the literature

that the error behaviour generally improves with smaller quanta. A study is required on the smallest quanta for which real-time execution is still possible. This should be addressed before a final decision is made to migrate the simulation to the proposed hybrid architecture. It should be noted here, that the simulation has not in the past, and should not in future, sacrifice accuracy for run-time performance. Maintaining accuracy is required because the simulation results feed into higher level decisions that rely on a certain level of predictive accuracy. The mind-set behind migration to a real-time architecture is to achieve the same accuracy as before, but to do it more efficiently.

Optimistic Time Management

Optimistic time management has been shown to typically lower the time management overhead in distributed synthetic environments, compared to conservative time management. The future inclusion of optimistic time management would however require considerable changes to many of the GBADS sub-system models to support the necessary model roll-back mechanisms. An approach where new models incorporate optimistic time management and roll-back, but the older models still incorporate conservative time management could, in future, be investigated to migrate to an optimistic time management scheme.

Optimistic Time Management of HIL and OIL

HIL and OIL interfaces require a strictly increasing and linear flow of time—at least within the requirement and perception of the systems and humans to which they are interfaced. Conservative time management makes possible such a strictly increasing flow of time, but is—as mentioned—not as efficient as optimistic time management. Optimistic time management, on the other hand, may cause causality to be violated. Partaking systems are however expected to tolerate and aid rolling back of their own state, then fixing and re-executing the simulation time line. More work is required on incorporating HIL and OIL capabilities into optimistic discrete event simulations.

Collaborative Modelling and Simulation

Collaborative M&S is another important future research direction. This includes collaborative model construction, composition and collaborative simulator environments. To accomplish this it might make sense to get acquainted with DEVS programming extensions such as DEVS-C++ or DEVS-Java

which aid in and enforce the use of DEVS concepts and the use of HLA when developing simulations and their partaking models.

Chapter 12

Dissertation Self Evaluation

The process of studying the Principles, Practice and Theory of Modelling and simulation and then discussing and analysing the current simulator within this known framework and body of knowledge has in itself given some hints on the potential simplicity of the solution to the scalability problem. This, in itself, is already a giant leap within the development group, as now, in hindsight, the application of a quantised approach seems to be the obvious choice.

From the literature, the advantages of a discrete event simulator over that of a discrete time simulator, specifically in terms of distributed parallel scalability, were already known and well understood. The problem addressed in this dissertation, however, focussed on how an existing discrete time simulator could be migrated to a discrete event specification and discrete event simulator, to increase its scalability. The value addition of the work described in this dissertation is therefore in:

- Broadening of the knowledge base and capability of the development team, and
- providing a case study—accessible to the wider simulation community—of the advantages of migrating an existing discrete time simulator to one based on a quantised discrete event modelling approach.

Additionally, the hybrid discrete-event/discrete-time simulator results obtained were very encouraging, approaching performance of the ideal distributed simulator.
This dissertation followed a research approach of:

- Doing the relevant background study and, in doing so, describing the current distributed parallel simulator within the newly explored—but

well known within the international military simulation community—body of knowledge,

- experimenting with, and analysing the current discrete time simulator,
- proposing and implementing a hybrid discrete-event/discrete-time simulator, and
- experimenting with, and analysing the proposed architecture, leading to various conclusions and future work proposals.

This approach allowed the successful analysis of the discrete time simulator. It also made possible the application of existing theorems and corollaries in researching and implementing the hybrid simulator and also in analysing it as the solution to the scalability problem.

The experimental procedure that was applied to both the discrete time simulator and the hybrid simulator made a comparative performance study possible. The performance analysis may be reproduced on a hardware infrastructure similar to the one discussed in chapter 7. This would of coarse only be possible if a discrete time architecture similar to the one discussed in Chapter 6 was used.

Bibliography

[1] J. Pretorius. Feasibility considerations for a tailored simulation based acquisition (SBA) approach. Master's thesis, University of Pretoria, Pretoria, South-Africa, 2003.

[2] J. Baird and J. Nel. The evolution of M&S as part of smart acquisition using the SANDF GBADS programme as an example. In *Proceedings of the 12th European Air Defence Symposium*, Shrivenham, England, 2005.

[3] S. Naidoo and J. Nel. Modelling and simulation of a ground based air defence system and associated tactical doctrine as part of acquisition support. In *Proceedings of the 2006 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, 2006.

[4] W. le Roux. Implementing a low cost distributed architecture for real-time behavioural modelling and simulation. In *Proceedings of the 2006 European Simulation Interoperability Workshop*, Stockholm, Sweden, 2006.

[5] F. Kuhl, R. Weatherley, and J. Dahmann. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall, Upper Saddle River, NJ, USA, 1999.

[6] IEEE std 1516-2000, IEEE standard for modelling and simulation (M&S) high level architecture (HLA)—framework and rules. Technical report, IEEE, 2000.

[7] IEEE 1516 HLA compliance check list. https://www.dmso.mil/public/transition/hla/compliancetesting.

[8] E. Page and R. Smith. Introduction to military training simulation: A guide for discrete event simulationists. In *Proceedings of the 1998 Winter Simulation Conference*, Miami, Florida, USA, 1998.

[9] S. Straßburger. *Distributed Simulation Based on the High Level Architecture in Civilian Application Domains*. SCS Publishing House, Ghent, Belgium, 2000.

[10] R. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley-Interscience, New York, USA, 2000.

[11] J. Baird and J. Nel. Modelling and simulation in support of the SANDF GBADS acquisition programme - an update. In *Proceedings of the 2007 South African Joint Air Defence Symposium*, Pretoria, South Africa, 2007.

[12] A. Duvenhage. A state estimation approach for live aircraft engagement in a C2 simulation environment. In *Proceedings of the 2007 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, 2007.

[13] R. Oosthuizen. Doctrine development during systems acquisition and the importance of modelling and simulation. In *Proceedings of the 12th European Air Defence Symposium*, Shrivenham, England, 2005.

[14] J. Roodt, J. Nel, and R. Oosthuizen. A system-of-systems simulation architecture for command & control at the joint operations level: Proposed synthesis of a test-bed for development of concepts and doctrine. In *Proceedings of the 2007 Land Warfare Conference*, Adelaide, Australia, 2007.

[15] B. Zeigler, T. Kim, and H. Praehofer. *Theory of Modelling and Simulation, second edition*. Academic Press, San Diego, California, USA, 2000.

[16] S. Taylor, A. Bruzzone, R. Fujimoto, B. Gan, S. Straßburger, and R. Paul. Distributed simulation and industry: Potentials and pitfalls. In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, California, USA, 2002.

[17] A. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[18] S. Alhir. *Learning UML*. O'Reilly & Associates, Inc, California, USA, 2003.

[19] B. Zeigler and J. Lee. Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment. In *Proceedings of SPIE - Enabling Technology for Simulation Science II*, Orlando, Florida, USA, 1998.

[20] H. Sarjoughian and B. Zeigler. Collaborative modeling: the missing piece of distributed simulation. In *Proceedings of SPIE - The International Society for Optical Engineering*, Orlando, Florida, USA, 1999.

[21] E. Kofman, J. Lee, and B. Zeigler. DEVS representation of differential equation systems: Review of recent advances. In *Proceedings of the 2001 European Simulation Symposium*, Marseille, France, 2001.

[22] B. Zeigler, M. Jamshidi, and H. Sarjoughian. Robot vs robot: Biologically-inspired discrete event abstractions for cooperative groups of simple agents. *Festschrift Conference in Honor of John H. Holland*, 1999.

[23] J. Nutaro, B. Zeigler, R. Jammalamadaka, and S. Akerkar. Discrete event solution of gas dynamics within the DEVS framework. *Lecture Notes in Computer Science - Computational Science – ICCS 2003*, 2660, 2003.

[24] Y. Kim, J. Kim, and T. Kim. Heterogeneous simulation framework using DEVS BUS. *Simulation*, 79(1), 2003.

[25] H. Praehofer. *System Theoretic Foundations for Combined Discrete-Continuous System Simulations*. PhD thesis, Johannes Kepler University of Linz, Linz, Austria, 1991.

[26] B. Zeigler, H. Song, T. Kim, and H. Praehofer. DEVS framework for modelling, simulation, analysis, and design of hybrid systems. *Lecture Notes in Computer Science - Hybrid Systems II*, 999, 1995.

[27] B. Zeigler. DEVS theory of quantised systems. Technical report, University of Arizona, Tucson, Arizona, USA, 1998.

[28] G. Klir. *Architecture of Systems Problem Solving*. Plenum Press, New York, USA, 1985.

[29] R. Sargent. Verification, validation, and accreditation of simulation models. In *Proceedings of 2000 Winter Simulation Conference*, Orlando, Florida, USA, 2000.

[30] R. Sargent. An expository on verification and validation of simulation models. In *Proceedings of 1985 Winter Simulation Conference*, Orlando, Florida, USA, 1985.

[31] J. Roodt. Modelling and simulation verification, validation and accreditation process. Technical report, Council for Scientific and Industrial Research, Pretoria, South-Africa, 2001.

[32] K. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5), 1979.

[33] D. Jefferson and H. Sowizral. Fast concurrent simulation using the time warp mechanism. In *Proceedings of the 1985 SCS Distributed Simulation Conference*, San Diego, California, USA, 1985.

[34] T. McLean and R. Fujimoto. Predictable time management for real-time distributed simulation. In *Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation*, Atlanta, Georgia, USA, 2003.

[35] J. Magee and J. Kramer. *Concurrency: State Models & Java Programs*. John Wiley & Sons, New York, USA, 1999.

[36] R. Fujimoto. Distributed simulation systems. In *Proceedings of the 2003 Winter Simulation Conference*, New Orleans, Louisiana, USA, 2003.

[37] R. Fujimoto. Parallel and distributed simulation. In *Proceedings of the 2001 Winter Simulation Conference*, Arlington, Virginia, USA, 2001.

[38] IEEE std 1516.1-2000, IEEE standard for modelling and simulation (M&S) high level architecture (HLA)—federate interface specification. Technical report, IEEE, 2000.

[39] IEEE std 1516.2-2000, IEEE standard for modelling and simulation (M&S) high level architecture (HLA)—object model template (OMT) specification. Technical report, IEEE, 2000.

[40] IEEE std 1516.3-2003, IEEE recommended practice for high level architecture (HLA) federation development and execution process (FEDEP). Technical report, IEEE, 2003.

[41] B. Zeigler, S. Hall, and H. Sarjoughian. Exploiting HLA and DEVS to promote interoperability and reuse in lockheed's corporate environment. *Simulation*, 73(5), 1999.

[42] M. Ogata, A. Higashide, M. Cammarano, and T. Takagi. RTI performance in the distributed real-time vehicle model simulation in a 3-D graphical environment. In *Proceedings of the 2001 European Simulation Interoperability Workshop*, Harrow, England, 2001.

[43] S. Jolibois, T. Joubert, and H. Wentzler. New HLA based technologies and methods for an advanced air to air combat simulation. In *Proceedings of the 2003 European Simulation Interoperability Workshop*, Stockholm, Sweden, 2003.

[44] R. Fujimoto and P. Hoare. HLA RTI performance in high speed LAN environments. In *Proceedings of the 1998 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, 1998.

[45] B. Watrous, L. Granowetter, and D. Wood. HLA federation performance: What really matters? In *Proceedings of the 2006 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, 2006.

[46] T. Schulze, S. Straßburger, and U. Klein. Migration of HLA into the civil domains. *Simulation*, 73(5), 1999.

[47] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[48] G. Amdahl. Validity of the single-processor approach to achieving large-scale computing capabilities. In *Proceedings of AFIPS 1967*, Atlantic City, New Jersey, USA, 1967.

[49] B. Duvenhage and J. Nel. The contribution of static and dynamic load balancing to a real-time distributed air defence simulation. In *Proceedings of the 2008 SimTecT Conference*, Melbourne, Victoria, Australia, 2008.

[50] B. Duvenhage and W. le Roux. TCP simulation architecture investigation. Technical report, Council for Scientific and Industrial Research, Pretoria, South Africa, 2004.

[51] B. Duvenhage and D. Kourie. Migrating to a real-time distributed parallel simulator architecture. In *Proceedings of the 2007 Summer Computer Simulation Conference*, San Diego, California, USA, 2007.

[52] B. Wilkinson and M. Allen. *Parallel Programming, second edition*. Pearson Education, Inc., Upper Saddle River, New Jersey, 2005.

[53] G. Wainer and B. Zeigler. Experimental results of timed cell-DEVS quantisation. In *Proceedings of AIS'2000*, Tucson, Arizona, USA, 2000.

[54] B. Zeigler, G. Ball, H. Cho, J. Lee, and H. Sarjoughian. Bandwidth utilization/fidelity tradeoffs in predictive filtering. In *Proceedings of the 1999 Fall SISO Simulation Interoperability Workshop*, Orlando, Florida, USA, 1999.

A Peer-to-Peer Simulation Architecture

Bernardt Duvenhage and Willem H. le Roux

Council for Scientific and Industrial Research
Pretoria, South Africa
Email: bduvenhage,whleroux@csir.co.za

Abstract— **A distributed parallel and soft real-time simulation architecture is presented. It employs a publish-subscribe communication framework layered on a peer-to-peer Transport Control Protocol-based message passing architecture. Mechanisms for efficient implementation and control of information flow between simulated entities form part of the architecture. A light-weight base simulation object model is also employed to provide maximum modularity and extensibility while keeping complexity manageable. The simulation architecture evolved over time to allow for the efficient implementation of a system of systems, virtual simulation. It has been successfully applied in an air defence simulation as a decision support tool and for standard operating procedure concept evaluation.**

KEYWORDS

Distributed, Parallel, Soft Real-Time, Simulation, Architectures, Peer-to-peer, Publish-Subscribe.

I. INTRODUCTION

In a decision support environment, system of systems level simulations are applied to provide end-users with the capabilities to identify, define, implement (virtual) and evaluate concepts that would otherwise be costly, time-consuming or impractical. Systems of systems level simulators typically involve multiple modelled entities with complex interactions and are executed in virtual or constructive simulation modes [1].

This paper presents a simulation architecture that evolved over time during decision support to an air defence procurement programme [2, 3]. Although there was no need for a generic, re-usable architecture – It was to be only used for a single simulation environment – it still had to provide standardised interfaces for efficient integration of models, services and other simulation-logistical functions. It should also support both constructive and virtual simulations, hence it should at least be soft real-time compatible when performing operator-in-the-loop simulations. Operators will mainly interact with the simulation via integrated mock-up consoles and not full immersive synthetic environments, which may be supported by integrated, external systems. Furthermore, it must allow both distributed and non-distributed simulation execution. The first is required to maintain soft real-time compliance when employing the virtual simulation mode if model processing loads are high. The latter is required for easier test and debugging as well as batch executions for statistical analyses. A conservative, discrete stepped time management mode forms an inherent part of the simulation architecture, as almost all of the models used in the air defence simulation environment are discrete time-stepped. To provide an efficient and effective decision support capability, specifically during system conceptualisation and field exercises, the architecture should allow for quick implementation and integration of new models. The same holds for the integration of external systems. Interoperability with other simulations is not an absolute requirement, but should not be excluded by design.

Several peer-to-peer architectures are reported in the literature, of which some are aimed at internet-based information sharing, discrete event simulations [4]–[7] or cooperative computing, such as solving processing intensive problems with *ad hoc* peer-to-peer networks [8]. Some architectures are also aimed at massively multi-player online role playing and other games [9, 10]. Giesecke [11] quantitatively investigated availability in peer-to-peer systems for prediction and identified basic characteristics to derive a formal model for describing architectures. Kotilainen [12] reports on an efficient peer-to-peer network simulator used to study artificial neural network algorithms.

Other simulation architectures or frameworks include the Aggregate Level Simulation Protocol (ALSP) [13], Distributed Interactive Systems (DIS) [14] and the High Level Architecture (HLA) [15]. The first two are seen as precursors to HLA. Although all three were developed for the defence community of the United State of America, HLA was intended to be adopted by the wider simulation community. The Standard Simulation Architecture [16] provides an additional framework to HLA to allow more flexibility, but be more cost-effective without paying performance penalties. The Open Simulation Architecture (OSA) [17] is a discrete event simulation architecture that promises integration of new and existing contributions at all levels. Hawley [18] proposes an object-oriented simulation architecture that separates the implementation of the dynamic system being modelled (application layer) from the simulation management functions (executive layer). The Extensible Modelling and Simulation Framework (XMSF) [19] aims to harness web-based technologies to promote interoperable simulations and provides mechanisms for systems to discover and use web services.

Of these architectures only HLA was evaluated since it is a fully fledged approach that covers all aspects of the simulation life cycle. However, in the South African defence environment it was not the optimal choice at the time. Although HLA promotes interoperability, the federation object model should still

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

be agreed or translated when two simulations are integrated. This was not always the case, therefore interoperability was not easily achievable [3].

The simulation architecture presented is not offered as an alternative for the above-mentioned architectures or frameworks, but rather to highlight the mechanisms used to implement an efficient architecture against the backdrop of system of systems simulation criteria. Efficiency in terms of implementation is required since a very small development team was used. In terms of simulation execution, soft real-time execution for multiple entities with update rates of 100Hz are used, requiring an architecture with low overheads.

II. System of Systems-level Simulation Architecture Needs

This section addresses the specific needs for a simulation architecture to meet the criteria as outlined in Section I and is discussed in the following subsections.

A. System of Systems Simulation

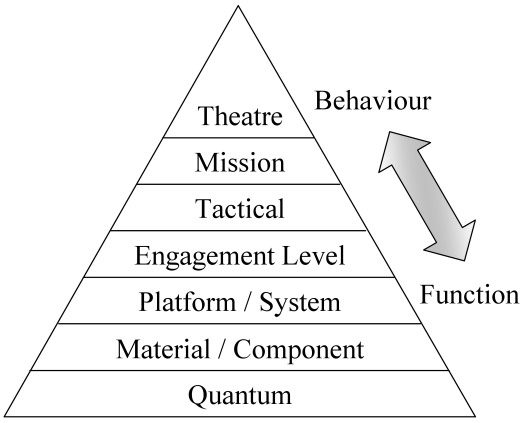

Fig. 1: The Systems Hierarchy applied to Modelling and Simulation (Adapted from [1, 20])

An adapted version of the systems hierarchy is shown in Figure 1 as applied to modelling and simulation [1, 20]. Systems of systems simulations are typically applicable at the engagement and tactical levels where entities are modelled at equipment and individual operator level. Individual entities are modelled, but forms "systems" when grouped organisationally. Tactical simulations also require one-on-one engagements, whereas higher order simulations require strategic actions of aggregated entities. For mission and theatre simulations entities are aggregated into units and generally referred to as war-gaming simulations [1]. Lower system hierarchy simulations tend to focus more on the function of entities, whereas the higher levels on the behaviour of entities.

B. Constructive and Virtual Simulation Mode Support

Tactical level simulations (Figure 1) imply that not only equipment is simulated, but also the use of equipment, including standard operating procedures. This again implies that the human operators, human-equipment and human-human interactions be modelled, so that it becomes a constructive simulation. Simulation execution requirements for constructive simulations tend to be less stringent, but the faster a simulation executes, the more applicable it becomes as a what-if type analyses tool, as it allows a simulation user to test and evaluate scenarios quickly. Virtual simulations need to be at least soft real-time compliant to maintain realism [21].

C. Distributed and Non-Distributed Simulation

Non-distributed simulations are generally less complex to test and debug than distributed simulations, as simulation execution does not have to be traced across multiple processing nodes. However, non-distributed simulations may be soft real-time incompatible when model processing loads become too high for a single processing node.

Distributed simulations on the other hand require efficient inter-process communication frameworks, such that the inter-processing node communication overheads do not counter the advantage of extra processing nodes. Virtual simulations with multiple entities that are modelled at system of systems level, typically require distributed simulation to either provide faster than or real-time compatibility. The ideal simulation architecture would support both distributed and non-distributed simulation execution without having to alter the implementation, but only its configuration.

D. Modularity and Extensibility

Since the simulation architecture is used in a decision support environment, including the evaluation of system concepts, it should be efficient to add, maintain and upgrade models of equipment, operator terminals, external system interfaces and operators. In addition to the entities that participate in a synthetic environment, it should also be efficient to extend, maintain and upgrade the synthetic environment itself. Services such as inter-entity line of sight calculations should be inherently part of the synthetic environment. External system interfacing should be supported, but note that external systems may have requirements that cannot be met by the simulation architecture, such as hard real-time compatibility.

E. Time-stepped Simulation

Conservative time management is an integral part of the simulation architecture and is enforced by using a discrete time-stepped mechanism. Spatio-temporal properties play a pivotal role in any air defence system, therefore time-line accuracy is of importance in a simulation environment, and hence architecture.

Although models may internally use predictive event-based time management, their external interfaces should support conservative time management. This is necessary as both predictable and non-predictable events occur in an air defence simulation environment, of which the non-predictable events may violate causality, if non-conservative time management is used.

Most of the external systems that will be integrated, produce spatio-temporal data, be it in the form of positions of an

aircraft from a flight simulator, or time-stamped detections from a sensor such as a radar. External systems that may be integrated includes equipment, data sources and simulators.

III. High-Level Simulation Architecture Design

In order to meet the needs as identified in Section II, a simulation architecture evolved from a single application to a fully distributed simulation architecture. After providing a short overview of the present architecture, each part is carefully explained in subsequent subsections.

The simulation architecture is based on an inter-process communication (IPC) framework using the Transfer Control Protocol (TCP). Processing nodes are fully connected in a peer-to-peer fashion and message-passing is managed via a publish-subscribe mechanism. Processes that need to communicate within the simulation architecture are:

- Models - Models of equipment, humans and operator consoles (interfaces).
- Services - Includes line-of-sight, terrain elevation and peripheral services such as data loggers.
- Consoles or Gateways - All external systems that need to be integrated with the simulation architecture is implemented via a gateway which in effect translates the protocol of the external system into the simulation object and spatial reference models of the simulation architecture. Mock-up operator consoles are also integrated via this mechanism.

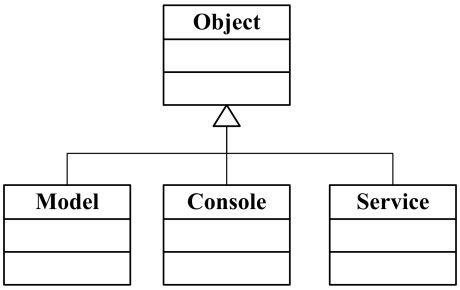

Fig. 2: Base Simulation Object Model

The above list of items is grouped under a base object to form the simulation object model of the architecture as indicated in Figure 2. An economical simulation object model (SOM) has been designed to curb implementation complexity.

A. Publish-Subscribe Object Communication Framework

All models, services and consoles (hereafter collectively referred to as objects) should be able to communicate efficiently in a distributed environment. Furthermore, it should be easy to establish and manage the communication channels between objects. A layered approach will reduce future migration effort to other IPC frameworks: keep the object implementation and communication logistics of an object separate. The object communication framework should also hide the underlying distributed implementation from the user of the framework and not dictate model fidelity or simulation granularity. The framework should allow distributed and non-distributed execution for easier test and debugging without requiring implementation changes of the framework or client software.

The publish-subscribe mechanism employed in the object communication framework is analogous to magazine subscriptions and also similar to object management in HLA [22]:

- Different publishers advertise their sets of titles available for subscription.
- Subscribers may subscribe to titles of their choice when they would like to.
- Publishers will then publish issues at regular intervals, which all subscribers will receive.
- All subscribers receive identical copies of an issue of a title.
- Publishers may also add titles at any given time to their collections. Cancellation of titles is not supported at present.
- A subscriber can read a copy of an issue as many times as they would like until a new one arrives.
- A subscriber may also elect to ignore old issues and only read the latest. Note that this is one of two supported modes, the second is an extension to the analogy (See next list).
- A publisher cannot change the content of an issue once it has been published and received by its subscribers.

The implementation of the publish-subscribe mechanism is somewhat extended over the analogy to allow more flexibility:

- Subscribers can select at what interval (rate) they want to receive issues. The maximum update rate is limited by the smallest time increment (frame) of the simulation.
- Subscribers can select if they want all issues of a given publisher, or just selected titles, without subscribing separately to each title.
- The subscriber may select what will happen with copies of issues that are received but not read. If kept, all copies from the oldest to the most recent have to be read to get to the latest issue. If not kept, the most recent copy is always available to be read. An in between mode is not supported.
- A specialised extension to support modelled communications between objects (modelled equipment or operators, not IPC related communications) is provided with additional parameters to support transmission functionality (status, delays, sender/receiver identification, etc.). Messages destined for transmission are passed immediately to the receiver where they are delayed in a cache to model the correct transmission delay, until delivery. The minimum transmission delay of a message is limited by the minimum time increment of the simulation. Messages are also rather delayed at the receiver than the transmitter, as the receiver knows, implicitly, its own position and, from the issue meta-data, the sender's position which are both required by the communications model.

Functionality as listed in the above two lists, are directly supported in the simulation architecture as part of the base object model and the simulation backbone, which is a set of

classes and functions providing the necessary mechanisms.

B. Peer-to-peer Processing Node Architecture

A peer-to-peer processing node architecture was ultimately selected above the client-server architecture, as the server may form a bottleneck due to the double latency and bandwidth usage for messages transmitted from a client to the server and then to the receiving client from the server (Figure 3(a)).

To minimise traffic at a server, an intermediate layer of servers were considered before using the peer-to-peer architecture. The intermediate servers (Figure 3(b)) have less traffic to route, and will only transmit messages to other intermediate servers via the top-level server if a receiving client requires it. The scheme is efficient, but has one major drawback: The architecture is not domain independent, as the clustering of clients per intermediate server requires prior knowledge of clients that can be grouped by type or anticipated traffic.

Note that the peer-to-peer architecture will result in a single latency for messages passed between nodes (indicated as peers in Figure 3(c)), but IPC connections have to be brokered or configured in some way before a simulation execution starts. In the client-server case, all clients connect to the same server. The peer-to-peer architecture suffers from the same domain knowledge challenge as the intermediate server solution, i.e. which models may be grouped for acceptable execution performance. The ultimate architecture would allow for both the automatic distribution of models across processing nodes, as well as automatically introducing intermediate server layers for optimal execution performance. Each processing node executes a subset of all objects (models, consoles and services). In the case where a single processing node is used, all objects are executed on it. As conservative time management is used in a time-stepped fashion, the slowest or most processing intensive object governs the global execution performance of the simulation. Load balancing is therefore necessary and is supported either as a static configuration or with dynamic load management [23]. The latter requires passing of objects in-process between processing nodes during run-time.

The peer-to-peer architecture is fully connected, thus each node is connected to each other node at start-up using TCP. This results in $\frac{n(n-1)}{2}$ connections, where n is the number of nodes (peers). For the client-server case the number of connections equals the number of clients. Connections between objects are made using a proprietary, binary-packed protocol, irrespective if objects are on the same processing node or not, or if only one processing node is used.

C. TCP Implementation Details

Specific TCP implementation tweaks to ensure lower message latency between nodes are discussed in this subsection.

TCP messages are grouped per destination node and sent off together instead of sending each message separately. Message latency still turned out to be a problem due to TCP's Nagel algorithm [24]. The Nagel algorithm usually improves bandwidth (saves on message header overhead) by caching short messages for a certain time-out or until they are big enough to fill a complete data packet before sending the data. Typically message groups were much smaller than the normal packet size of 1.5 kilobyte. Turning off the Nagel algorithm gave the simulation architecture complete control over message sending times which decreased latencies considerably. The communication model would cause a node to send information to every other node once every simulation time increment which means that the shorter the latencies the faster the simulation can execute.

The TCP sending buffer was also made bigger than the default to allow the simulation architecture to push messages into the sending buffer without blocking to allow the simulation to continue processing while the TCP operating system thread continues sending. This approach saves the overhead of implementing the simulation's TCP sending code in a separate processing thread.

To start a distributed simulation, all the nodes except the first node may be started up in any order. As soon as the first node is started it makes connections to all the other nodes, which in turn make connections to each other and finally start the simulation.

IV. Results

Initial experiments with a non-distributed and intermediate server-client (see Section III) architecture showed that in order to execute large enough simulations, distributed processing would be required to maintain soft real-time compatibility [25]. Approximately 6-8 processing nodes were estimated for real-time compliance, but less could be used, as the model loading metrics were very conservative. Between 40 and 100 objects, with varying levels of fidelity were anticipated. With the actual architecture, a fully populated scenario translates to 177 entities that require processing. The architecture is still efficient enough to execute this at approximately soft real-time. Of the entities, 160 are models, 8 consoles and 9 services.

Soft real-time execution is maintained by synchronising with the local processing node clock. Processing time is yielded not to exceed real-time. However, this only works when the processing nodes are not overloaded, i.e. able to process all models within a simulation time frame, otherwise extra processing nodes may be added. If this still does not help, soft real-time compatibility cannot be maintained.

Distributed performance tests were done with a processing intensive test object that takes exactly 1ms PC time to increment and publishes a single title which is a text string of length 512 bytes. Each test object subscribes to the titles of all the other test objects, including its own title. The communication setup is thus fully connected over all objects. A 100Hz closed loop distributed simulation is run over one to six machines in as fast as possible mode. Simulation distribution and communication overhead results are presented in Table I. The simulation frames are 10ms in length, equating to a 100Hz update rate, giving ten 1ms slots for a maximum of ten models per node to sustain soft real-time execution. It can be seen that a single node is very efficient, running at

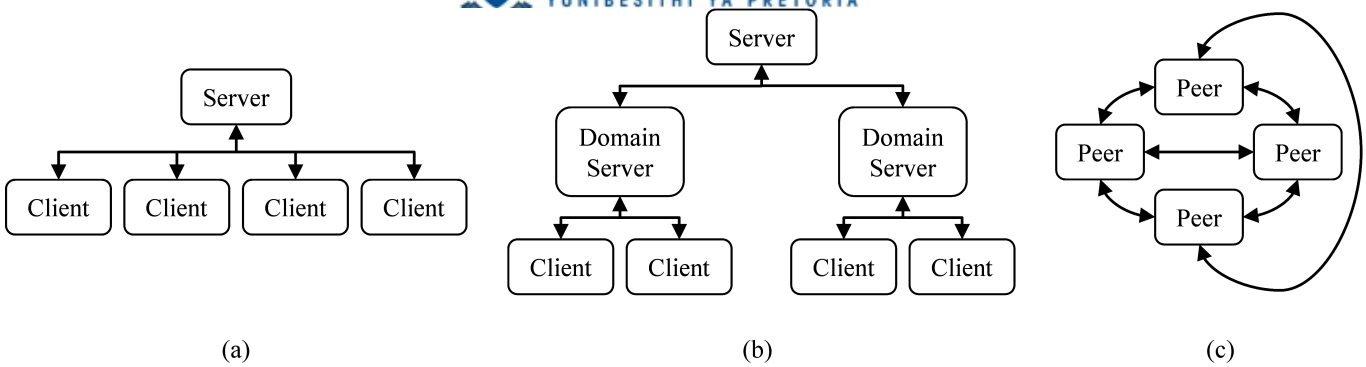

Fig. 3: Client-Server (a) Intermediate-Server (b) and Peer-to-Peer (c) Processing Node Architectures

99% of real time with 10 models which translates to a 1% communication overhead. Table I also shows that:

1) to run up to 9 test objects real time at least 1 node is required;
2) to run up to 18 test objects real time at least 2 nodes are required;
3) to run up to 24 test objects real time at least 3 nodes are required;
4) to run up to 32 test objects real time at least 4 nodes are required;
5) to run up to 35 test objects real time at least 5 nodes are required;
6) to run up to 42 test objects real time at least 6 nodes are required.

To run 42 test objects at real time at least 6 nodes are required running 7 objects each. This translates to an average communication overhead of just under 30%. Network usage was measured by using Windows XP's task manager network performance window. Note that the tests performed are worst case scenarios. All objects subscribe to all other objects and themselves (n^2 subscriptions) and the issue size is 512 bytes which is big enough for 21 double precision 3D coordinate triplets or a list of approximately 64 English words.

The simulation architecture has been successfully applied in an air defence simulation as a decision support tool and for standard operating procedure concept evaluation. It has been applied in extensions of the air defence simulation that include satellite-based optical and radar sensors for maritime surveillance concept development. It was also used for integration with various external systems, including situational air picture systems, operator console mock-ups, air traffic radars, flight simulators and similar simulations.

V. Future Work

A key challenge with soft real-time simulations is what should be done if the simulation slips on world-time? This often occurs, as models tend to have spurious high processing requirements. What techniques should be used to catch up again on world-time, specifically when a simulation is connected to external systems, such as a flight simulator or air picture systems? One solution is to use innovative schemes in console implementations to external systems to cater for data that arrives at the wrong-time, i.e. too late, or to too early. In the first case, prediction algorithms are necessary and in the latter buffering schemes.

Future work includes conducting comparative studies between peer-to-peer, intermediate server and client-server architectures. There is also ample opportunity for load balancing research: How to measure model loading per processing node efficiently and effectively, and load balancing algorithms.

VI. Conclusion

The ability to execute an entire simulation in an all-in-one mode on a single processing node (desktop computer) is a key advantage in how the simulation architecture is used. It is efficient and quick to configure scenarios for simulation, and to visually verify them using peripheral two and three dimensional viewers. Similar techniques are used to verify and validate newly integrated models, consoles or services. It is then merely a matter of changing a configuration to execute the simulation distributed to achieve soft real-time execution. The simulation architecture is suitable for parallel execution on a small to medium scale infra-structure.

The publish-subscribe architecture is very flexible in terms of objects and connection management. However, care should be taken to adhere to a standard way of implementing objects and not to abuse the flexibility.

The soft real-time performance figures obtained with worst-case object loadings indicate that the architecture is adequate for a small number of nodes with less than 10 objects per node. It is expected that network overheads will limit scalability to less than 100 objects in total, and therefore, the architecture is not considered to be scalable for large simulations (100's of objects), requiring soft real-time performance.

Author Biographies

BERNARDT DUVENHAGE has been with the CSIR within the Mathematical and Computational Modelling Research Group, that's part of the Defence, Peace, Safety and Security (DPSS) Research Group, since January 2004. Past responsibilities have included the development of a distributed simulation architecture, development of an optimized Line of

TABLE I: Distribution and Communication Overhead Results

Number of Nodes	Objects per Node	Issue Size	Percentage Real-Time	Network Utilization	Total Objects
1	7	512 bytes	142%	0%	7
2	7	512 bytes	132%	6%	14
3	7	512 bytes	126%	12%	21
4	7	512 bytes	117%	16%	28
5	7	512 bytes	110%	20%	35
6	7	512 bytes	102%	25%	42
1	8	512 bytes	124%	0%	8
2	8	512 bytes	116%	7%	16
3	8	512 bytes	110%	14%	24
4	8	512 bytes	100%	18%	32
5	8	512 bytes	92%	23%	40
6	8	512 bytes	85%	27%	48
1	9	512 bytes	110%	0%	9
2	9	512 bytes	104%	8%	18
3	9	512 bytes	99%	15%	27
4	9	512 bytes	90%	21%	36
5	9	512 bytes	81%	26%	45
6	9	512 bytes	75%	27%	54
1	10	512 bytes	99%	0%	10
2	10	512 bytes	93%	9%	20
3	10	512 bytes	88%	17%	30
4	10	512 bytes	77%	23%	40
5	10	512 bytes	70%	28%	50
6	10	512 bytes	60%	32%	60

Sight (LOS) algorithm and LOS service, development of a terrain attitude and altitude service, development of models in cooperation with OEMs and development of a 3D analysis tool based on the Open source Scene Graph (OSG). He completed his BSc(hons) in Computer Science in 2005 and is currently pursuing an MSc in Computer Science on real time simulation architectures.

HERMAN LE ROUX has been with the South African Council for Scientific and Industrial Research since April 1998 and is at present a Principal Engineer in the Mathematical and Computational Modelling Research Group. He is involved in Modelling and Simulation-based Acquisition Decision Support, specifically for the South African National Defence Force. Interests include information fusion, biometrics, artificial intelligence and software engineering. Le Roux completed a Masters Degree in Computer Engineering at the University of Pretoria in 1999 and is currently pursuing a PhD in Information Fusion.

ACKNOWLEDGEMENTS

The authors would like to thank Dr Jackie Phahlamohlaka, Cobus Nel, Anita Louis and Arno Duvenhage, all colleagues at the CSIR, for their valuable inputs, as well as the Armaments Corporation of South Africa for supporting this work.

REFERENCES

[1] DOD, "Department of Defense: Modelling and Simulation Master Plan," Under Secratary of Defense for Acquisition and Technology, DoD 5000.59-P, Alexandria, VA, October 1995.

[2] J. J. Nel and H. J. Baird, "The evolution of M&S as part of smart acquisition using the SANDF GBADS programme as example," in *Twelfth European Air Defence Symposium*, Shrivenham, June 2005.

[3] W. H. le Roux, "Implementing a low cost distributed architecture for real-time behavioural modelling and simulation," in *Proceedings of the 2006 European Simulation Interoperability Workshop*. Stockholm: Simulation Interoperability Standards Organization, June 2006.

[4] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, and I. Wakeman, "Towards yet another peer-to-peer simulator," FOURTH INTERNATIONAL WORKING CONFERENCE PERFORMANCE MODELLING AND EVALUATION OF HETEROGENEOUS NETWORKS, West Yorkshire, September 2006.

[5] A. Montresor, G. D. Caro, and P. E. Heegaard, "Architecture of the simulation environment," Information Society Technologies, Italy, Project Report IST-2001-38923, January 2004.

[6] S. Cheon, C. Seo, S. Park, and B. P. Zeigler, "Design and implementation of distributed DEVS simulation ina peer to peer network system," in *Proceedings of the 2004 Military, Government and Aerospace Simulation Symposium*, Virginia, April 2004.

[7] B. Gedik and L. Liu, "A scalable peer-to-peer architecture for distributed information monitoring applications," *IEEE Transactions on Computers*, vol. 54, no. 6, pp. 767–783, June 2006.

[8] G. D. Costa, "Peer-to-peer simulation," Presentation at the Federal University of the Rio Grande Do Sul, Porto Alegre, 2002.

[9] Y. Kawahara, H. Morikawa, and T. Aoyama, "A peer-to-peer message exchange scheme for large scale networked virtual environments," Proceedings of the 8th IEEE International Conference on Communications Systems (ICCS 2002), Amsterdam, April 2002.

[10] S.-P. A. van Houten and P. H. M. Jacobs, "An architecture for distributed simulation games," in *Proceedings of the 2004 Winter Simulation Conference*, R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, Eds., Washington DC, December 2004.

[11] S. Giesecke, T. Warns, and W. Hasselbring, "Availability simulation of peer-to-peer architectural styles," in *Proceedings of the 2005 workshop on Architecting dependable systems (WADS 2005)*. Waterloo: ACM Press, 2005, pp. 1–6.

[12] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, and J. Vuori, "P2prealm peer-to-peer network simulator," in *Proceedings of the 11th Intenational Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*, June 2006, pp. 93–99.

[13] R. Weatherly, D. Seidel, and J. Weissman, "Aggregate Level Simulation Protocol," Presented at the 1991 Summer Computer Simulation Conference, Baltimore, July 1991.

[14] "IEEE standard for information technology - protocols for distributed interactive simulations applications," IEEE Std 1278-1993, 1993.

[15] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, ser. Artificial Intelligence. Upper Saddle River: Prentice Hall, 1999.

[16] J. S. Steinman and D. R. Hardy, "Evolution of the standard simulation architecture," http://www.modelingandsimulation.org/issue10/SISO/steinman.html, vol. 3, nr. 2 , issue 10, April-June 2004, Accessed 16 January 2007.

[17] O. Dalle, "OSA: an open Component-based architecture for Discrete-event Simulation," INRIA, Tech. Rep. RR-5762, February 2006, available from http://www.inria.fr/rrrt/rr-5762.html.

[18] P. A. Hawley and T. J. Urban, "An object-oriented simulation architecture," AIAA Modeling and Simulation Technologies Conference and Exhibit, Providence, August 2004.

[19] D. Brutzman, M. Zyda, J. M. Pullen, and K. L. Morse, "Extensible modeling and simulation framework (XMSF) challenges for web-based modeling and simulation," TECHNICAL CHALLENGES WORKSHOP, STRATEGIC OPPORTUNITIES SYMPOSIUM, www.movesinstitute.org/xmsf, San Diego, October 2002.

[20] J. H. S. Roodt, L. Berglund, and P. Klum, Worksession between FOI and CSIR, Linköping, 2001.

[21] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "NPSNET: A network software architecture for large scale virtual environments," *Presence*, vol. 3, no. 4, 1994.

[22] "High Level Architecture interface specification: Version 1.3," U.S. Department of Defense, April 1998.

[23] B. Duvenhage, "The contribution of static and dynamic load balancing in a real-time distributed air defence simulation," 2007, to be submitted to the Summer Computer Simulation Conference, San Diego.

[24] B. A. Forouzan, *TCP/IP Protocol Suite*, 2nd ed., ser. Forouzan Networking. Boston: McGraw-Hill, 2003.

[25] B. Duvenhage and W. H. le Roux, "MobADS: TCP-based distributed simulation architecture investigation," Council for Scientific and Industrial Research, Tech. Rep. DEFT-MSADS-00151, July 2004.

Migrating to a Real-Time Distributed Parallel Simulator Architecture

Bernardt Duvenhage and
Derrick G Kourie (Role of Masters degree supervisor)
Espresso Group, University of Pretoria
bduvenhage@csir.co.za and dkourie@cs.up.ac.za

Keywords: peer-to-peer, parallel simulation, discrete time step simulation, publish-subscribe, high real-time frame-rate

Abstract

A legacy non-distributed logical time simulator is migrated to a distributed architecture to parallelise execution. The existing Discrete Time System Specification (DTSS) modelling formalism is retained to simplify the reuse of existing models. This decision, however means that the high simulation frame rate of 100Hz used in the legacy system has to be retained in the distributed one—a known difficulty for existing distribution technologies due to inter-process communication latency.

A specialised publish-subscribe simulation model is used for the new simulator architecture. The simulation model, including the process synchronisation, is implemented using a low latency peer-to-peer TCP messaging protocol. The TCP send and receive buffers and TCP's Nagle algorithm are also tweaked to ensure low latency communication. Gigabit Ethernet is used at the hardware layer. A parallelised execution speed-up of four to five times is reached with six to eight machines at a simulation frame rate of 100Hz.

1. INTRODUCTION

The South-African National Defence Force's (SANDF's) need for decision support and concurrent tactical doctrine development within a Ground Based Air Defence System (GBADS) acquisition program offered an ideal opportunity to establish an indigenous and credible modelling and simulation capability within the South-African defence acquisition environment [1][2]. The broad requirement of the capability is to simulate a GBADS battery of existing and still to be acquired (possibly still under development) equipment and their related human operators at a system of systems level within a realistic Synthetic Environment (SE). During the concept and definition phases of the acquisition life cycle [3] the capability was successfully provided by a non-distributed simulator and its architectural predecessors [4]. A selection of the models were derived from high fidelity engineering models, some by OEMs, and developed within a 100Hz logical Discrete Time System Specification (DTSS) [5] that simplified the time and causality management. The non-distributed simulator evolved within this 100Hz logical time DTSS modelling formalism and was implemented to run As Fast As Possible (AFAP).

Real-time simulation execution became a prioritised requirement during the development phase of the acquisition life cycle due to the realised impact of *realistic* human-simulation interaction when doing tactical doctrine development. Human interaction would happen through an Operator In the Loop (OIL) console with the possibility to record the operator's actions to be re-used in statistical simulation runs when and as required. To support the real-time requirement it was decided to parallelise the simulator across multiple Commercial Off the Shelf (COTS) PC nodes connected with Gigabit Ethernet.

For simplicity in the economical reusability of all the existing models it was also decided to retain the 100Hz logical time and DTSS modelling formalism. To achieve real-time execution, the parallelised logical time DTSS simulator is run AFAP, but the execution is throttled to not exceed real-time. To guarantee causal message delivery for the discrete time execution though, severe real-time constraints is placed on the minimum required inter-node communication latency as each simulation frame is a mere 10ms. The next section introduces existing distributed and parallel simulator technologies and their applicability to a 100Hz DTSS modelling formalism. The following sections then pose a research question on a new simulator architecture and develop the proposed architecture to inform the research question through analysis. The paper finds resolution in the flexibility and performance analysis of the new architecture and a concluding assessment of the architecture's suitability as a distributed parallel, high resolution logical time, DTSS simulator.

Within the Mathematical and Computational Modelling (MCM) Research Group of the Defence, Peace, Safety and Security (DPSS) operating unit of the South-African Council for Scientific and Industrial Research (CSIR) the main author had the responsibility of developing the new distributed parallel simulation architecture. This architecture is under investigation as part of an MSc project in Computer Science with the co-author (role of the Masters degree supervisor) from The University of Pretoria.

2. EXISTING DISTRIBUTED AND PARALLEL SIMULATOR TECHNOLOGIES

This section introduces existing distributed and parallel simulator technologies and their applicability to a 100Hz log-

 ISBN # 1-56555-316-0

ical time DTSS modelling formalism. Specifically technologies suitable for deployment on a distributed COTS PC infrastructure are discussed.

Looking at the literature, the most popular and thoroughly analysed, distributed simulation technologies within the military domain [6] seem to be Distributed Interactive Simulation (DIS) and the High Level Architecture (HLA), which implement Discrete Event System Specifications (DEVS) rather than a DTSS. The HLA is a generalisation and extension of DIS and the Aggregate Level Simulation Protocol (ALSP), both of which evolved from the Simulator Networking (SIMNET) project. SIMNET is, according to Page and Smith [6], the first meaningful attempt to interoperate military simulators within the United States' Department of Defence.

DTSS may however be embedded [5] within DEVS. Ogata, et al. [7] tested the real-time performance of DIS and different versions of the RTI-NG HLA Run-Time Infrastructure (RTI). Their real-time vehicle model simulation within a 3D graphical environment reached a frame rate ceiling of around 30Hz with both DIS and HLA implementations.

The HLA's real-time performance, for both RTI-NG and DMSO RTI implementations, is also studied by Jolibois, et al [8] in the context of a beyond visual range air to air combat simulation. The performance is shown to be less than ideal for 10Hz and higher simulation frame rates. This is due to message latency, object time advance latency and message deliveries leaking into adjacent simulation time steps.

Fujimoto and Hoare [9] investigated an alternative for the current versions of the HLA RTIs that can achieve latencies that are suitable for high simulation frame rates, but these are based on a low latency Gigabit Myrinet [http://www.myricom.com/myrinet/overview/] hardware layer and specialised RTIs. When Fujimoto and Hoare analysed the latency for DMSO RTI1.3 over an Ethernet TCP and UDP implementation it was found to be in the order of 10ms, which is too large to sustain a 100Hz simulation frame rate. They also found that the DMSO RTI supports a time advance frequency of more than 2000 per second between two nodes, but for three and more nodes the time advance frequency unfortunately dropped sharply to values as low as 10Hz with even only a few objects per node.

Watrous, et al. [10] explain that in the HLA, and in fact in any distributed algorithm, a time management scheme, such as the logical time DTSS modelling formalism, which requires contributions from all other nodes are relatively expensive. For this reason the HLA allows federates (simulation components) to employ their own unconstrained time management to avoid the time synchronisation overhead. In such an unconstrained case each model will synchronise itself against its simulator's wall clock without explicit synchronisation with other models, or between simulators, but at the risk of loosing message causality.

A recent distributed parallel simulator architecture is the Aurora master/worker architecture [11]. According to Park and Fujimoto the goal of Aurora is to harness available computation time from a large number of machines rather than strictly achieving high speed-ups on dedicated hardware. Aurora is unique among its class of distributed parallel architectures in the fact that it is aimed at parallel discrete event simulation (PDES) and, for example, includes simulation time management. It is unfortunately, like HLA, not well suited to tightly coupled high resolution discrete time simulations.

From the indicated examples it is clear that using existing distributed and parallel technologies such as HLA or DIS for implementation of a logical time DTS parallel simulator might introduce technical risks in getting the frame rate to or beyond 100Hz while still ensuring message causality. It is worth noting that this is purely due to HLA, DIS and similar architectures not being designed for high resolution logical time communicating parallel processes. These architectures, in particular HLA, seem rather to be aimed at providing simulation development efficiency, economical interoperability and geographical distribution, typically implementing a Discrete Event System Specification (DEVS) modelling formalism [5].

An important architecture driver is that of simulator interoperability. It is not a present or foreseeable priority [4] within this specific acquisition environment and is not a national imperative at this stage. According to Straßburger[12] these factors further decrease the drive behind and thus viability of the use of interoperability standards.

3. RESEARCH QUESTION

Can a peer-to-peer publish-subscribe simulator architecture implement a logical time DTSS modelling formalism to support a four to five times parallelised 100Hz closed loop simulation?

The research question firstly enquires whether or not a publish-subscribe simulation model can provide the required flexibility to support the simulation capability. Secondly it enquires whether or not a TCP message passing implementation of the simulation model and simulation synchronisation can maintain real-time execution of the 100Hz logical time DTSS at increased levels of parallelisation.

The publish-subscribe paradigm is chosen for its simplicity and familiarity. The TCP message passing implementation is chosen for TCPs stream based and reliable nature and the fact that COTS PCs with Gigabit Ethernet Network Interface Cards (NICs) and a commercial switch will be used at the physical and data-link layers. Initial TCP messaging tests [13] also revealed that a Gigabit TCP connection could quite possibly support the required low message latencies.

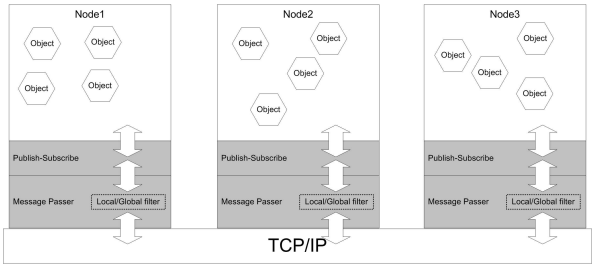

Figure 1. Layered Peer-to-Peer Simulator Architecture

4. THE PUBLISH-SUBSCRIBE DISTRIBUTED PARALLEL SIMULATOR ARCHITECTURE

The discussion on the new publish-subscribe simulator architecture is structured around the layered architecture of the simulator (shown in Figure 1), which includes a publish subscribe simulation layer, a message passing implementation of the simulation model and at the bottom layer a low latency TCP messaging protocol for Gigabit Ethernet.

The attraction of the layered architecture was the separation of concerns, in terms of design, between the simulation model and the distributed execution thereof. An additional advantage is of course the ability to change the implementation of the bottom layers without affecting the top layer simulation application.

4.1. Publish-Subscribe Simulation Model

The top layer simulation model encompasses a couple of aspects, which include the simulation time management, the system specification modelling formalism, the object communication framework and the synthetic environment services.

As mentioned, the pre-existing models have been implemented within a conservative logical time management scheme and a DTSS modelling formalism. It was decided to keep these aspects unchanged to simplify the reuse of the existing models. The object communication framework that is under investigation for the simulation model is a specialised publish-subscribe framework to be discussed next. Discussions on the synthetic environment services will then follow.

4.1.1. The Publish-Subscribe Object Communication Framework

The publish-subscribe paradigm is well known to anyone that has ever needed to organise to get information, for example a magazine, on his or her topic of interest on a regular basis. Each magazine within your topic (category) of interest has a title and a regular interval at which the categorical information is made available (published). You, the subscriber, may request that the information be delivered to your doorstep in the form of, say, a weekly or a monthly magazine issue.

The publish-subscribe simulation framework is a direct analogy to the magazine example. An instance of a simulation model (an object) may express its desire to receive information within a certain category of interest, e.g. aircraft positions, by adding the category (and title name, if known) to its *Subscription Wish List*. An object may also express its willingness to share information within a certain category, such as its own position, by adding a title (name and category) to its *Owned Title List*. A subscribing object has no guarantee that any object will share information under the title category or name in which it is interested. Similarly a publisher object has no guarantee that any other objects will be interested in the information that it is willing to share. At simulation start-up each object's owned title list is made known to the rest of the simulation. The titles are then processed against the objects' wish list subscriptions. Each title matching a wish list subscription generates a subscription which is sent back to the title owner to be added to the title's subscriber list.

At simulation run-time each object will go through regular increment, publish and gather cycles. Within the DTSS modelling formalism an object is incremented every n'th discrete time simulation frame where n is the object's *trigger frame*. Each wish list subscription, and thus each subscriber in a title's subscriber list, is also associated with a trigger frame. During a simulation frame, each subscriber of each owned title will be visited and an issue sent to the subscriber if it is the subscription's trigger frame. An important publish rule that is required to ensure consistent issues is that the contents of a title issue may only be updated during the publisher's increment cycle.

Objects may at simulation run-time express their wish to share a new category of information or a new title within an existing category. This is done by submitting a *run-time title* to the communication framework. Similarly objects may express interest in categories (or titles within categories) of information at run-time by submitting a *late subscription*

An object has an issue pigeon hole for each of its wish list subscriptions. When an issue is received (gather phase) it is placed in the appropriate pigeon hole. A pigeon hole may have subscription history turned off or on. If history is off then a newer version of an issue replaces all old issues that may remain in the pigeon hole. If history is turned on then issues will be added to the pigeon hole in chronological order. The object may then read issues and manually delete them as required during increment cycles. Turning history on for a specific wish list subscription is typically required when a subscriber doesn't want to miss any important updates (events) for that subscription. Having history off allows the subscriber to always have access to the current issue without the overhead of always caching and processing a subscription's recent

history.

4.1.2. The Synthetic Environment Services

The two types of simulation services supported are, firstly, low level services that are built into the simulation model and, secondly, high level services that run on top of the simulation model as simulation objects. The only low level service currently implemented is that of delayed issues. An issue may be given a future delivery time by either the publisher, or the subscriber upon delivery. Such an issue would be delivered to the subscriber immediately, but once there it resides in a delayed issue list until the time of delivery arrives at which point the issue is put into the appropriate pigeon hole of the subscriber. Delayed issues are handy if transmission delays of messages within the SE are to be modelled. In the current simulator the issue delays of tactical communication subscriptions are, when required, calculated by a radio and cable network model.

High level synthetic environment services subscribe to the objects' state titles and then apply environmental tools such as Line Of Sight (LOS) and terrain engines to give each object individual feedback on its height, which objects it can see, etc. To accomplish the personalised feedback a service advertises what is called a differentiated title. Each time a subscription is made to a differentiated title the simulator automatically creates a personalised title and subscription for the subscriber. The service may then use the created titles to publish to individual objects.

A service need not always publish data back to the simulation, though. Logging, for example, is a high level service that accumulates object states and other information. The logging service may then apply user configured data analysers to the accumulated data and log the results to disk.

4.2. Peer-to-Peer Message Passing and Node Synchronisation

The publish-subscribe communication framework and the simulator synchronisation is implemented with a peer-to-peer message passing architecture. A peer-to-peer architecture is specifically preferred above a client-server architecture to avoid the double latency that exists when communicating via a server to a third machine. The messaging implementation of the publish-subscribe communication framework is presented, followed by the implementation of the simulation synchronisation.

4.2.1. Messaging Implementation of Publish-Subscribe

The publish-subscribe framework naturally translates to a messaging architecture containing only three message types. A title may be advertised as a title message containing all the title and publisher details. A wish list subscription may similarly be a message containing the details of the wish list subscription and the subscriber. The third message type is an issue message that contains the subscriber's node-number delivery address, the targeted wish list subscription pigeon hole and the actual issue payload. The messaging implementation has a local/global filter (see Figure 1) that will loop a node's self addressed messages back to be cached for the next simulation frame without passing anything down to the TCP layer.

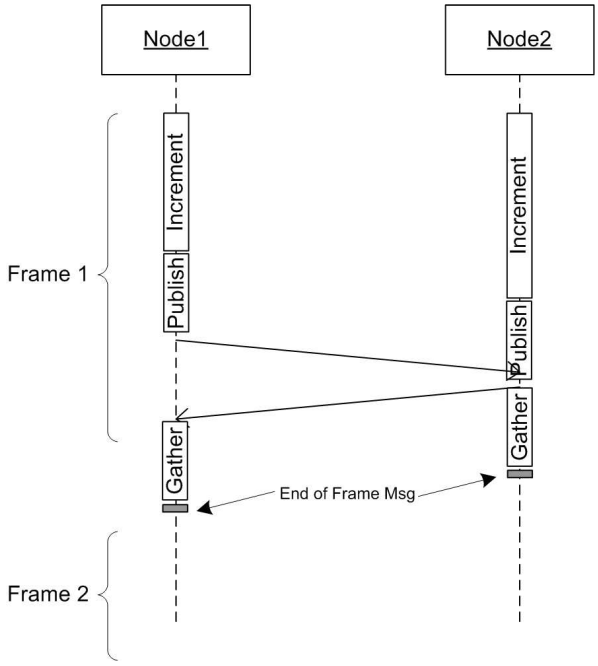

Figure 2. Peer-to-Peer Message Passing and Simulation Synchronisation

4.2.2. Peer-to-Peer Node Synchronisation

The peer-to-peer synchronisation scheme is shown in Figure 2. Each simulation frame has three consecutive execution phases. Within the first phase, which is the *increment phase*, all the objects are put through their increment-publish cycles. The published issues are not messaged directly, but are grouped per destination node and cached until the second, so called publish, phase. The cached issue groups may now be sent to their respective destination nodes. The publish phase must be followed by a time-stamped end-of-frame message to each peer node to signify that all the issues for the current simulation frame have been sent. The end-of-frame messages perform a similar function as Chandy-Misra null messages [14] for dead-lock avoidance and time management in DEVS implementations. A simulator node will wait in the gather phase until it has received and processed an end-of-

frame message from each of the other simulator nodes after which it starts with the increment phase of the next simulation frame.

4.3. TCP Message Passing Implementation

The TCP messaging implementation consists of two components. The first of which is an address translation from destination node number to destination IP and port before any message can be sent via TCP. This translation is preconfigured and fixed for each distribution configuration.

The second component is a two-tiered approach to lowering TCP message latency. The first tier is to ensure that as much as possible of the TCP send and receive overhead happens in parallel to the node execution. This is accomplished by increasing TCP's send and receive buffers to an adequate size such that the buffers have enough space for two simulation frames worth of data. This ensures that all TCP sends are non-blocking. It also facilitates CPU time, from a second CPU or hyper-thread or that's not used by the simulation, to be used to transport as much data as possible from the nodes' send buffers across TCP to their receive buffers for quick retrieval when needed.

The second tier takes control of the TCP message send times. TCP's Nagle algorithm tries to optimise bandwidth usage by conglomerating sent messages in the send buffer until it is large enough to fill a TCP packet or until a certain timeout is reached. The unfortunate side effect of the Nagle algorithm is that control over message latencies is lost. To give control over the message latency back to the simulator the Nagle algorithm is disabled.

5. ANALYSIS AND RESULTS

The two aspects of the simulator architecture to be analysed in support of the research question are, firstly, the simulator's applicability to a 100Hz DTSS modelling formalism and, secondly, the flexibility of the publish-subscribe simulation model and its suitability for a system of systems tactical and SE simulation.

5.1. Experimental Setup

The simulator nodes are similar Pentium 4 3.2GHz machines with 2GB of dual-channel RAM each and WindowsXP SP2. The network infrastructure is, as mentioned, Gigabit Ethernet with a D-Link DGS-3324SR managed switch. Each node has an Intel D945PAW mother board with an on-board Intel Pro/1000 PM Gigabit Ethernet network card.

The simulator nodes will be populated with instances of a "test" model. The test model has a fixed processing requirement of 1ms per 10ms simulation frame and an owned title with a fixed issue size of 512 bytes. Furthermore each instance of the test model subscribes to every other instance, creating the worst case communication scenario of a fully connected communication graph.

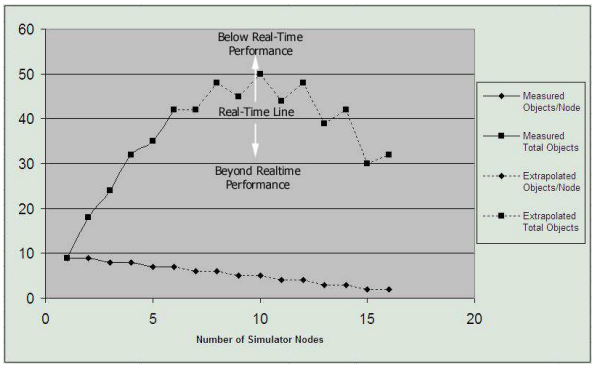

Figure 3. Total Object Performance of 100Hz Peer-to-Peer Simulator

5.2. Applicability of the Peer-to-Peer Simulator to a 100Hz DTSS Modelling Formalism

The proposed architecture's real-time performance is analysed over distributions of one to six simulator nodes on the target infrastructure. With each node configuration the number of objects per node will be limited to achieve a real-time frame-rate. Finally a simple predictive model for the distributed performance behaviour is derived from the analysis data and used to do a first order estimate of the simulator's scalability to seven and more nodes. Accurate evaluation over more nodes should however be part of the future work section to verify the speculation about the simulator's scalability.

The performance result that is recorded is the maximum number of objects per node (see Figure 3) such that the simulation can still reach real-time. If the total number of objects are increased above the "Total Objects" graph, the performance will drop below real-time. Conversely, if the total number of objects are decreased below the "Total Objects" graph, the performance will grow beyond real-time. Both the total number of objects and the performance speed-up graphs are derived from the measured objects-per-node graph (Figure 3 and Figure 4).

Quantifying the measured communication overhead it seems that each time a simulator node is added, the number of model instances per node must be decreased by an average of 0.5 to maintain real-time which is a 0.5% overhead of the 10ms simulation frame. The explanation of these results is quite likely not a simple task, see the Section 7. on future work, as it may be dependent on multiple factors such as message structure and grouping. However, assuming for the purpose of first order performance predictions, that the results do indeed indicate a linear distribution overhead

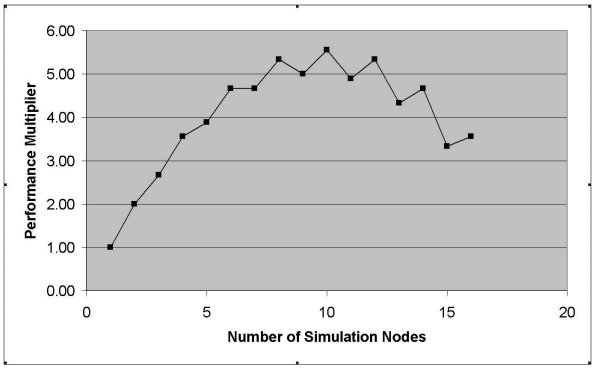

Figure 4. Real-Time Performance Speed-Up of 100Hz Peer-to-Peer Simulator

of 5% for each simulator node added, such a linear overhead would most probably be in the receive loop of each simulation frame. Amdahl's Law [www.wikipedia.org] specifies that the speed-up attainable by parallel execution is limited by the sequential components of the system which in this case is a single NIC and thus a single, though full-duplex, communication channel per simulator node.

A linear performance might seem counter intuitive to what is expected of an n node and fully connected peer-to-peer structure where the total number of connections grows by n^2. The linear nature does however make sense if one remembers that the processing is done by n nodes resulting in a processing time of $\frac{n^2}{c.n}$ which is proportional to n and therefore linear. In other words, each node must receive data from each of the other nodes in turn, limiting the potential parallelisation.

The first order objects-per-node performance for seven and more nodes is estimated by linearly extrapolating the measured objects-per-node curve (Figure 3) under the previous assumption. The spikiness of the performance graphs is due to the granularity of the objects which, in general, leaves a fragmented processing slot (idle time) on each node. The linear extrapolation provides an estimate for the scalability of the simulator, but as the number of nodes increases to beyond 10 the total number of objects eventually start to decrease which implies that the communication bandwidth will also decrease again. Around this point it is expected that the linear nature of the objects-per-node curve might change which requires, as mentioned, analysis over more nodes to draw accurate scalability conclusions beyond 10 nodes.

5.3. Suitability and Flexibility of the Publish-Subscribe Simulation Model

The flexibility of the publish-subscribe simulation model and its suitability for a DTSS system of systems tactical and SE simulation is analysed to resolve the first part of the research question. The publish-subscribe simulation model is shown to be suitable for the implementation of a DTSS modelling formalism and the simulator synchronisation to be free of deadlock which is sometimes a problem for distributed simulators. Finally the measure of flexibility of the simulation model is defined and demonstrated as the range of application domains within which the simulator have been successful.

Table 1. Successful Application Domains of the Peer-to-Peer Publish-Subscribe Simulator Architecture

Application Domain	Description
GBADS	Ground Based Air Defence System Performance Analysis
MobADS	Mobile Air Defence System Performance Analysis
Navy SBADS	Concept Demonstrator for Developing Tactical Doctrine for Naval Air Defence Scenarios
Sensor Webs (Awarenet)	Concept Demonstrator for value of additional coastal surveillance in creating situational awareness when patrolling South-Africa's fishing zones

The publish-subscribe simulation model sets up virtual communication channels between the objects. These channels provide a way for a subscriber to sample the publisher's state periodically as required within the DTSS modelling formalism [5]. This is accomplished by the subscriber periodically receiving the current issue of a certain title of the publisher.

The end of frame notifications are a special case of the Chandy-Misra null messages [14] with a fixed look ahead equal to the discrete time step. The simulation model may thus, according to Chandy-Misra, be shown to be deadlock free. An advantage of the DTSS implementation of null messages is that the number of null messages is limited to only one per simulator node per simulation frame. This improves on the typical null-message overhead within DEVS implementations which may generate excessive null-messages [15].

The flexibility of the simulation model is defined as the range of application domains, (see Table 1), within which this publish-subscribe simulation model and parallel peer-to-peer simulator have been successful. Each of these application domains has been validated against reality by running experiments that analyse various aspects of the system of systems. For each experiment the systems' simulated emergent behaviour is compared against the pre-defined expected behaviour and motivated, or corrected, by subject matter experts or from what is already known of the system. A list of external systems successfully integrated with the simulator may be found in Table 2.

Table 2. External Systems Successfully integrated with the Peer-to-Peer Publish-Subscribe Simulator Architecture

External System	Description
Simulation Viewers	Integration with 2D and 3D online and offline visual analysis tools
OIL Consoles	Integration with mock-up OIL consoles for realistic real-time operator-equipment interaction
Hardware In the Loop (HIL) Tracking Sensor	Integration with a Mechanised Optical and Radar Tracker (MecORT) for real sensor input when doing system analysis and validation
HIL Air Picture Sources	Integration with civilian and military air pictures supported by run-time simulated aircraft generation
Flight Simulator	Integration with a flight simulator for inclusion of realistic reactive pilot behaviour when doing system analysis.

6. CONCLUSION

The new 100Hz logical time DTSS publish-subscribe peer-to-peer simulator architecture achieves a measured speed increase, due to execution parallelisation, of above 4.5 when distributed over six simulator nodes. This equates to a distribution efficiency of 75%. The wide success of the application of the simulator architecture is used to motivate the publish-subscribe simulation model's suitability as a general purpose DTSS simulation model. The authors therefore conclude that the peer-to-peer publish-subscribe simulator is suitable to support the real-time execution of the 100Hz logical time DTSS simulation requirement.

The simulator's 100Hz logical time performance is also explained and modelled in a simple way which provides a first order estimate on the scalability of the parallelisation. The simulator is estimated to reach a parallelisation ceiling at a speed increase of approximately 5.5 which is achieved when distributed over 10 simulator nodes. This equates to a distribution efficiency of 55%. The scalability of such a high resolution logical time parallel simulator thus seems to be limited to applications requiring low to medium levels of parallelisation. The limiting factor for the parallelisation is, as mentioned, the sequential nature of the network communication.

The DTSS modelling formalism does seem to pose a technical difficulty in implementing large scale parallelisation of high resolution logical time simulations. In hind sight it seems like a good idea to rather develop a hybrid DTSS-DEVS modelling formalism, that has a DEVS layer enveloping the DTSS layer, to further migrate this specific simulation capability towards supporting large scale parallelisation. The two layer approach allows the existing DTSS models to be grouped and aggregated into systems level models for example, which may then be better suited to a DEVS modelling formalism. The DEVS layer then communicates only what is required and its parallelisation is not constrained by the underlying DTSS layer's logical time resolution.

7. FUTURE WORK

The reason for retaining the DTSS modelling formalism was for ease of reuse of existing discrete time models. In other words the authors believe that developing this parallel distributed simulator was more viable than, for example, migrating the entire modelling formalism to DEVS. This is a valid position for the currently required simulation capability, but future work to increase the scalability and usability of the simulator should include:

- Further investigation and a proper explanation of the scalability behaviour which might reveal ways of improving that behaviour,
- investigation into the different timings and groupings for execute-send cycles, possibly interleaving send with execute in a different way,
- investigation into using lower overhead UDP message passing, because in a dedicated and lightly loaded switched Gigabit Ethernet scenario it is known [16] that UDP packet losses occur very seldom, and
- investigation into the possible migration of the system specification modelling formalism to a hybrid DEVS-DTSS modelling formalism.

8. ACKNOWLEDGEMENTS

The authors would like to thank both the Armaments Corporation (Armscor) of South-Africa and the CSIR for supporting this research. Additionally the authors thank Anita Louis (CSIR) and Arno Duvenhage (CSIR) for their valuable review comments.

REFERENCES

[1] Johannes Lodewikus Pretorius. Feasibility considerations for a tailored simulation based acquisition (sba) approach. Master's thesis, University of Pretoria, 2003.

[2] Jacques Baird and Cobus Nel. The evolution of m&s as part of smart acquisition using the sandf gbads programme as an example. In *Proceedings of the 12th European Air Defence Symposium*, volume 3694, pages 173–182, 2005.

[3] Shahen Naidoo and Cobus Nel. Modelling and simulation of a ground based air defence system and associated tactical doctrine as part of acquisition support. In *Proceedings of the 2006 Fall Simulation Interoperability Workshop*, 2006.

[4] Willem H. le Roux. Implementing a low cost distributed architecture for real-time behavioural modelling and simulation. In *Proceedings of the 2006 European Simulation Interoperability Workshop*, 2006.

[5] Bernard P. Zeigler. *Theory of Modelling and Simulation*. Academic Press, 2000.

[6] Ernest H. Page and Roger Smith. Introduction to military training simulation: A guide for discrete event simulationists. In *Proceedings of the 1998 Winter Simulation Conference*, 1998.

[7] Michihiko Ogata, Akira Higashide, Mike Cammarano, and Toshinao Takagi. Rti performance in the distributed real-time vehicle model simulation in a 3-d graphical environment. In *Proceedings of the 2001 European Simulation Interoperability Workshop*, 2001.

[8] Stephane Jolibois, Thierry Joubert, and Herve Wentzler. New hla based technologies and methods for an advanced air to air combat simulation. In *Proceedings of the 2003 European Simulation Interoperability Workshop*, 2003.

[9] Richard Fujimoto and Peter Hoare. Hla rti performance in high speed lan environments. In *Proceedings of the 1998 Fall Simulation Interoperability Workshop*, 1998.

[10] Ben Watrous, Len Granowetter, and Douglas Wood. Hla federation performance: What really matters? In *Proceedings of the 2006 Fall Simulation Interoperability Workshop*, 2006.

[11] Alfred Park and Richard M. Fujimoto. Aurora: An approach to high throughput parallel simulation. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, 2006.

[12] Steffen Straßburger. *Advances in Simulation*. SCS Publishing House, 2000.

[13] Bernardt Duvenhage and Herman W. le Roux. Tcp simulation architecture investigation. Technical report, Council for Scientific and Industrial Research, 2004.

[14] K. Chandy and Jayadev Misra. Distrubuted simulation: A case study in design and verification of distributed programs. In *IEEE Transactions on Software Engineering*, volume SE-5, 2003.

[15] Richard M. Fujimoto. Parallel and distributed simulation. In *Proceedings of the 1999 Winter Simulation Conference*, 1999.

[16] Behrouz A. Forouzan. *TCP/IP Protocol Suite*. McGraw-Hill, Inc., New York, NY, USA, 2002.

Biography

Bernardt Duvenhage obtained his B.Sc (Honour) degree in Computer Science from the University of Pretoria in 2005 and is currently pursuing a Masters Degree. While part of the Mathematical and Computational Modelling Research Group of the Council for Scientific and Industrial Research (CSIR) in South Africa, he played a key role in developing the group's distributed simulator architecture; the simulation's terrain and LOS services; and the 3D visualisation and analysis tool of the synthetic environment. He is currently employed in the Optronic Sensor Systems Competency Area of a division within the CSIR. He intends further research in virtual environment simulation and visualisation.

Derrick Kourie lectures in the Computer Science department at Pretoria University. While his academic roots are in operations research, his current interests include, but are not limited to software engineering and algorithm development. He is student adviser to some 20 postgraduate students working in these and related areas. He is editor of the South African Computer Journal and serves on various national and international academic committees.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Migrating to a Real-Time Distributed Parallel Simulator Architecture

An Update *

Bernardt Duvenhage
The Computer Science Department of the University of Pretoria
Pretoria, South Africa
bduvenhage@csir.co.za

Categories and Subject Descriptors

J.7 [**Computers in Other Systems**]: *Command and Control*; J.7 [**Computers in Other Systems**]: *Military*

Keywords

discrete time, DTSS, discrete event, DEVS, distributed parallel simulation

ABSTRACT

A legacy non-distributed logical time simulator was previously migrated to a distributed architecture to parallelise execution. The existing Discrete Time System Specification (DTSS) modelling formalism was retained to simplify the reuse of existing models. This decision, however means that the high simulation frame rate of 100Hz used in the legacy system has to be retained in the distributed one—a known difficulty for existing distribution technologies due to inter-process communication latency.

The specialised discrete time distributed peer-to-peer message passing architecture that resulted to support the parallelised simulator requirements is analysed and the questions surrounding its performance and flexibility answered. The architecture is shown to be a suitable and cost effective distributed simulator architecture for supporting a four to five times parallelised implementation of a 100 Hz logical time DTSS modelling formalism.

From the analysis results it is however clear that the discrete time architecture poses a significant technical challenge in supporting large scale distributed parallel simulations. This is mainly due to sequential communication components within the discrete time architecture and system specification that cannot be parallelised. A hybrid DTSS/Discrete Event System Specification (DEVS) modelling formalism and simulator is proposed to lower the communication and synchronisation overhead between models and improve on the scalability of the discrete time simulator while still economically reusing the existing models.

The proposed hybrid architecture is discussed. Ideas on implementing and then analysing the new architecture to complete the author's masters dissertation are then touched upon.

*SAICSIT 2007 Student Paper - Progress of Masters dissertation following on original article[1] by the same name.

1. INTRODUCTION

The South-African National Defence Force's (SANDF's) need for decision support and concurrent tactical doctrine development within a Ground Based Air Defence System (GBADS) acquisition program offered an ideal opportunity to establish an indigenous and credible modelling and simulation capability within the South-African defence acquisition environment [2][3]. The broad requirement of the capability is to simulate a GBADS battery of existing and still to be acquired (possibly still under development) equipment and their related human operators at a system of systems level within a realistic Synthetic Environment (SE). A GBADS deployment, shown in Figure 1, usually consists of a layered air defence. The outer layer typically consists of eight very short range missile systems, each having a virtual operator and an accompanying buddy with a wide angle pair of binoculars. The second layer has four gun systems, each consisting of two guns, a tracking radar, a designation radar, a fire control system and at least three operators to operate the guns. The inner layer of defence usually has two short range missile systems, each consisting of a ground based launcher, a designation sensor, a fire control system and a couple of virtual operators. The deployment would defend some asset (vulnerable point) against an airborne threat scenario. A typical threat scenario would consist of one to many incoming attack aircraft which are the 'targets' to be engaged by the air defence system.

During the concept and definition phases of the acquisition life cycle [4] the capability was successfully provided by a non-distributed simulator and its architectural predecessors [5] developed by the CSIR. A selection of the models were derived from high fidelity engineering models, some by OEMs, and developed within a 100Hz logical Discrete Time System Specification (DTSS) [6] that simplified the time and causality management. The non-distributed simulator evolved within this 100Hz logical time DTSS modelling formalism

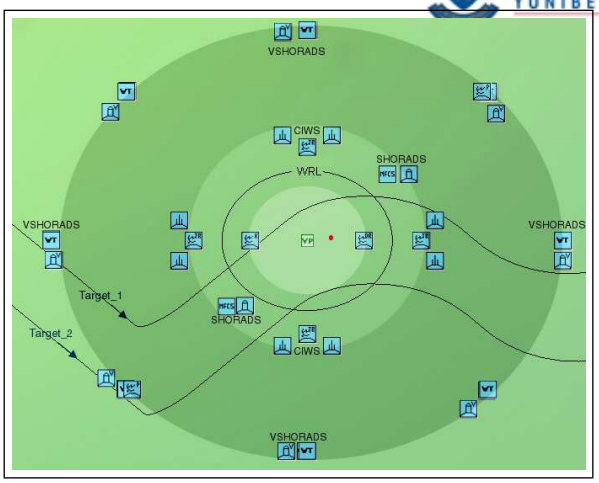

Figure 1: A Typical GBADS Deployment

and was implemented to run As Fast As Possible (AFAP).

Real-time simulation execution became a prioritised requirement during the development phase of the acquisition life cycle due to the realised impact of *realistic* human-simulation interaction when doing tactical doctrine development. Human interaction would happen through an Operator In the Loop (OIL) console with the possibility to record the operator's actions to be re-used in statistical simulation runs when and as required. To support the real-time requirement it was decided to parallelise the simulator across multiple Commercial Off the Shelf (COTS) PC nodes connected with Gigabit Ethernet. For economical reusability of all the existing models it was also decided to retain the 100Hz logical time and DTSS modelling formalism.

Several case studies of value for embedding a discrete time modelling approach within existing simulator distribution technologies, presented by Duvenhage and Kourie [1], showed that these reach a frame rate ceiling of 20 to 30Hz. A specialised 100Hz logical discrete time distributed simulator was developed to bridge the gap from 30Hz to 100Hz and this simulator is currently in use to provide the required simulation capability. To achieve real-time execution the logical time simulator's execution is throttled to not exceed real-time. Guaranteeing that at least real-time execution can be reached does place severe real-time constraints on the inter-node communication latency though, as each simulation frame is a mere 10ms.

The distributed discrete time simulator design is briefly discussed in the next section followed by some performance analysis results. The results are discussed and shown to indicate a technical difficulty in the future scalability of the discrete time simulator. A hybrid DTSS/Discrete Event System Specification (DEVS) modelling approach and simulator is then proposed to improve on the scalability of the discrete time simulator. Ideas on implementing and then analysing the new architecture to complete the author's Masters dissertation is finally presented.

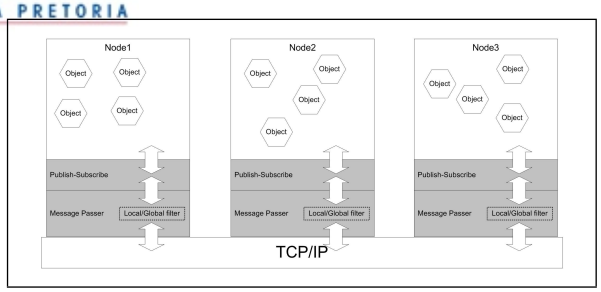

Figure 2: Layered Peer-to-Peer Simulator Architecture

2. THE DISTRIBUTED DISCRETE TIME SIMULATOR

The discussion on the new publish-subscribe simulator architecture is structured around the layered architecture of the simulator (shown in Figure 2), which includes a publish subscribe simulation layer, a message passing implementation of the simulation model and at the bottom layer a low latency TCP messaging protocol for Gigabit Ethernet.

The attraction of the layered architecture was the separation of concerns, in terms of design, between the simulation model and the distributed execution thereof. An additional advantage is of course the ability to change the implementation of the bottom layers without affecting the top layer simulation application.

2.1 Publish-Subscribe Simulation Model

The top layer simulation model encompasses a couple of aspects, which include the simulation time management, the system specification modelling formalism, the object communication framework and the synthetic environment services.

As mentioned, the pre-existing models have been implemented within a conservative logical time management scheme and a DTSS modelling formalism. It was decided to keep these aspects unchanged to simplify the reuse of the existing models. The object communication framework that is under investigation for the simulation model is a specialised publish-subscribe framework to be discussed next. Discussions on the synthetic environment services will then follow.

2.1.1 The Object Communication Framework

The publish-subscribe paradigm is well known to anyone that has ever needed to organise to get information, for example a magazine, on his or her topic of interest on a regular basis. Each magazine within your topic (category) of interest has a title and a regular interval at which the categorical information is made available (published). You, the subscriber, may request that the information be delivered to your doorstep in the form of, say, a weekly or a monthly magazine issue.

The publish-subscribe simulation framework is a direct analogy to the magazine example. An instance of a simulation model (an object) may express its desire to receive information within a certain category of interest, e.g. aircraft pos-

itions, by adding the category (and title name, if known) to its *Subscription Wish List*. An object may also express its willingness to share information within a certain category, such as its own position, by adding a title (name and category) to its *Owned Title List*.

At simulation run-time each object will go through regular increment, publish and gather cycles. Within the DTSS modelling formalism an object is incremented every n'th discrete time simulation frame where n is the object's *trigger frame*. Each wish list subscription, and thus each subscriber in a title's subscriber list, is also associated with a trigger frame. During a simulation frame, each subscriber of each owned title will be visited and an issue sent to the subscriber if it is the subscription's trigger frame.

An object has an issue pigeon hole for each of its wish list subscriptions. When an issue is received (gather phase) it is placed in the appropriate pigeon hole. A pigeon hole may have subscription history turned off or on. If history is off then a newer version of an issue replaces all old issues that may remain in the pigeon hole. If history is turned on then issues will be added to the pigeon hole in chronological order. The object may then read issues and manually delete them as required during increment cycles. Turning history on for a specific wish list subscription is typically required when a subscriber doesn't want to miss any important updates (events) for that subscription. Having history off allows the subscriber to always have access to the current issue without the overhead of always caching and processing a subscription's recent history.

2.1.2 *The Synthetic Environment Services*

The two types of simulation services supported are, firstly, low level services that are built into the simulation model and, secondly, high level services that run on top of the simulation model as simulation objects. The only low level service currently implemented is that of delayed issues. An issue may be given a future delivery time by either the publisher, or the subscriber upon delivery. Such an issue would be delivered to the subscriber immediately, but once there it resides in a delayed issue list until the time of delivery arrives at which point the issue is put into the appropriate pigeon hole of the subscriber. Delayed issues are handy if transmission delays of messages within the SE are to be modelled. In the current simulator the issue delays of tactical communication subscriptions are, when required, calculated by a radio and cable network model.

High level synthetic environment services subscribe to the objects' state titles and then apply environmental tools such as Line Of Sight (LOS) and terrain engines to give each object individual feedback on its height, which objects it can see, etc. To accomplish the personalised feedback a service advertises what is called a differentiated title. Each time a subscription is made to a differentiated title the simulator automatically creates a personalised title and subscription for the subscriber. The service may then use the created titles to publish to individual objects.

A service need not always publish data back to the simulation, though. Logging, for example, is a high level service that accumulates object states and other information. The logging service may then apply user configured data analysers to the accumulated data and log the results to disk.

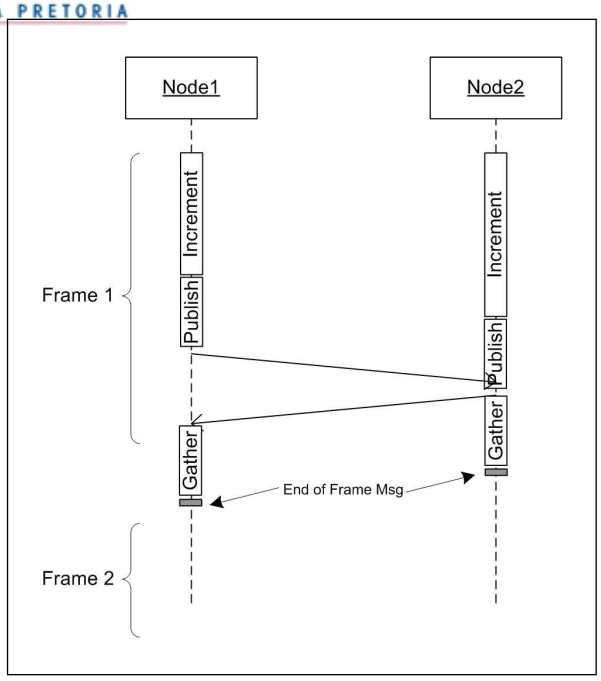

Figure 3: Peer-to-Peer Message Passing and Simulation Synchronisation

2.2 Peer-to-Peer Message Passing and Node Synchronisation

The publish-subscribe communication framework and the simulator synchronisation is implemented with a peer-to-peer message passing architecture. A peer-to-peer architecture is specifically preferred above a client-server architecture to avoid the double latency that exists when communicating via a server to a third machine. The messaging implementation of the publish-subscribe communication framework is presented, followed by the implementation of the simulation synchronisation.

2.2.1 *Messaging Implementation of Publish-Subscribe*

The publish-subscribe framework naturally translates to a messaging architecture containing only three message types. A title may be advertised as a title message containing all the title and publisher details. A wish list subscription may similarly be a message containing the details of the wish list subscription and the subscriber. The third message type is an issue message that contains the subscriber's node-number delivery address, the targeted wish list subscription pigeon hole and the actual issue payload. The messaging implementation has a local/global filter (see Figure 2) that will loop a node's self addressed messages back to be cached for the next simulation frame without passing anything down to the TCP layer.

2.2.2 *Peer-to-Peer Node Synchronisation*

The peer-to-peer synchronisation scheme is shown in Figure 3. Each simulation frame has three consecutive exe-

cution phases. Within the first phase, which is the *increment phase*, all the objects are put through their increment-publish cycles. The published issues are not messaged directly, but are grouped per destination node and cached until the second, so called publish, phase. The cached issue groups may now be sent to their respective destination nodes. The publish phase must be followed by a time-stamped end-of-frame message to each peer node to signify that all the issues for the current simulation frame have been sent. The end-of-frame messages perform a similar function as Chandy-Misra null messages [7] for dead-lock avoidance and time management in DEVS implementations. A simulator node will wait in the gather phase until it has received and processed an end-of-frame message from each of the other simulator nodes after which it starts with the increment phase of the next simulation frame.

2.3 TCP Message Passing Implementation

The TCP messaging implementation consists of two components. The first of which is an address translation from destination node number to destination IP and port before any message can be sent via TCP. This translation is pre-configured and fixed for each distribution configuration.

The second component is a two-tiered approach to lowering TCP message latency. The first tier is to ensure that as much as possible of the TCP send and receive overhead happens in parallel to the node execution. This is accomplished by increasing TCP's send and receive buffers to an adequate size such that the buffers have enough space for two simulation frames worth of data. This ensures that all TCP sends are non-blocking. It also facilitates CPU time, from a second CPU or hyper-thread or that's not used by the simulation, to be used to transport as much data as possible from the nodes' send buffers across TCP to their receive buffers for quick retrieval when needed.

The second tier takes control of the TCP message send times. TCP's Nagle algorithm tries to optimise bandwidth usage by conglomerating sent messages in the send buffer until it is large enough to fill a TCP packet or until a certain time-out is reached. The unfortunate side effect of the Nagle algorithm is that control over message latencies is lost. To give control over the message latency back to the simulator the Nagle algorithm is disabled.

2.4 Analysis and Results

The proposed architecture's real-time performance is analysed over distributions of one to six simulator nodes on the target infrastructure. With each node configuration the number of objects per node will be limited to achieve a real-time frame-rate. Finally a simple predictive model for the distributed performance behaviour is derived from the analysis data and used to do a first order estimate of the simulator's scalability to seven and more nodes. Accurate evaluation over more nodes should however be part of the future work section if such accuracy is required.

For the experimental setup the simulator nodes are similar Pentium 4 3.2GHz machines with 2GB of dual-channel RAM each and WindowsXP SP2. The network infrastructure is, as mentioned, Gigabit Ethernet with a D-Link DGS-3324SR managed switch. Each node has an Intel D945PAW mother

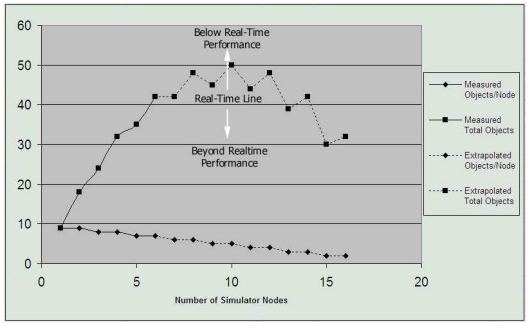

Figure 4: Total Object Performance of 100Hz Peer-to-Peer Simulator

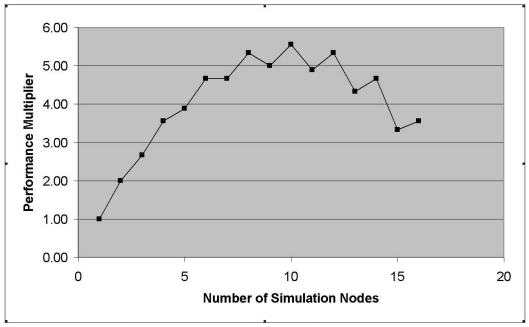

Figure 5: Real-Time Performance Speed-Up of 100Hz Peer-to-Peer Simulator

board with an on-board Intel Pro/1000 PM Gigabit Ethernet network card. The simulator nodes will be populated with instances of a "test" model. The test model has a fixed processing requirement of 1ms per 10ms simulation frame and an owned title with a fixed issue size of 512 bytes. Furthermore each instance of the test model subscribes to every other instance, creating the worst case communication scenario of a fully connected communication graph.

The performance result that is recorded is the maximum number of objects per node (see Figure 4) such that the simulation can still reach real-time. If the total number of objects are increased above the "Total Objects" graph, the performance will drop below real-time. Conversely, if the total number of objects are decreased below the "Total Objects" graph, the performance will grow beyond real-time. Both the total number of objects and the performance speed-up graphs are derived from the measured objects-per-node graph (Figure 4 and Figure 5).

Quantifying the measured communication overhead it seems that each time a simulator node is added, the number of model instances per node must be decreased by an average of 0.5 to maintain real-time which is a 0.5% overhead of the 10ms simulation frame. Assuming for the purpose of first order performance predictions, that the results do indeed indicate a linear distribution overhead of 5% for each simulator

node added, such a linear overhead would most probably be in the receive loop of each simulation frame. Amdahl's Law [www.wikipedia.org] states that the speed-up attainable by parallel execution is limited by the sequential components of the system which in this case is proposed to be the single NIC, and thus single communication channel, per simulator node.

A linear performance might seem counter intuitive to what is expected of an n node and fully connected peer-to-peer structure where the total number of connections grows by n^2. The linear nature does however make sense if one remembers that the processing is done by n nodes resulting in a processing time of $\frac{n^2}{c.n}$ which is proportional to n and therefore linear. In other words, each node must strictly receive data from each of the other nodes in turn, limiting the potential parallelisation.

The first order objects-per-node performance for seven and more nodes is estimated by linearly extrapolating the measured objects-per-node curve (Figure 4) under the previous assumption. The spikiness of the performance graphs is due to the granularity of the objects which, in general, leaves a fragmented processing slot (idle time) on each node. The linear extrapolation provides an estimate for the scalability of the simulator, but as the number of nodes increases to beyond 10 the total number of objects eventually start to decrease which implies that the communication bandwidth will also decrease again. Around this point it is expected that the linear nature of the objects-per-node curve might change which requires, as mentioned earlier, analysis over more nodes to draw accurate scalability conclusions beyond 10 nodes.

From the analysis of the results it seems that the new 100Hz logical time DTSS publish-subscribe peer-to-peer simulator architecture achieves a measured speed increase, due to execution parallelisation, of above 4.5 when distributed over six simulator nodes, but not higher than approximately 6 even when distributed over ten and more nodes. This simulator is currently in use and working as expected, but the DTSS modelling formalism does seem to pose a technical difficulty in implementing still larger scale parallelisation of high resolution logical time simulations due to the sequential components of the architecture that cannot be parallelised.

3. A HYBRID DEVS/DTSS MODELLING APPROACH

In hind sight it seems like a good idea to rather develop a hybrid DTSS-DEVS modelling formalism, that has a DEVS layer enveloping the DTSS layer, to further migrate this specific simulation capability towards supporting large scale parallelisation. The two layer approach allows the existing DTSS models to be grouped and aggregated into systems level models for example, which may then be better suited to a DEVS modelling formalism. The DEVS layer then communicates only what is required and its parallelisation is not constrained by the underlying DTSS layer's logical time resolution which would require strict high resolution time synchronisation between nodes.

It is known that a DTSS may be embedded within a DEVS [6].

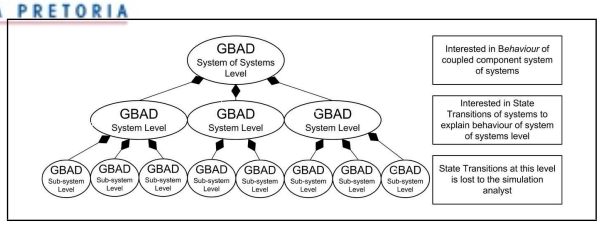

Figure 6: Double Structure Level of Discrete Time Simulator

This alone, however, does not improve the scalability of the simulator. The proposed aspects to improve the scalability is *Aggregating DTSS Models into DEVS Models* and *Using Dead-Reckoning Techniques* on the remaining high resolution data links. It is worth noting again here that reusing the current discrete time models, their publish-subcribe communication framework and the peer-to-peer messaging on TCP architecture is still of importance for economical reasons, minimising duplication of effort.

3.1 Aggregating DTSS Models into DEVS Models

The typical layout of a GBADS battery was described in the introduction. Most of the current GBADS models are at the level of GBAD sub-systems of systems within the battery. These models are typically modelled at a state transition system specification level or higher. At the GBAD system level the sub-system models are brought together to create system level models at a coupled system specification level as shown in Figure 6. The GBAD system models are then coupled again to create a GBAD system of systems level model also at the coupled component system specification level.

The nature of the system of systems simulation experiments requires objectives and outcome measures at the GBAD system level. In such an experimental frame the output variables within the GBAD system level coupled component models (the system structural knowledge) is hidden from the simulation analyst and argued to therefore be superfluous. The aim of *aggregation of the DTSS models* is the act of explicitly hiding the *double layer* of intermediate GBAD system structural information within a DEVS model. The new GBAD system level DEVS model is then a state transition system specification envelope which shields the model interconnect infrastructure from the communication overhead of the internal structure.

3.2 Using Dead-Reckoning Techniques

Some models, such as the incoming aircraft, are already at a GBAD system level. Nevertheless, these models still communicate at a high data rate to some of the other GBAD systems. The current tracking radar models, for example, require high time resolution target position input for their tracking filters to operate properly. This is due to the tracking radar, a GBAD sub-system, internally also being modelled at a coupled component level. This requires the correct external stimulation for all the components to operate together within the experimental frame for which they were originally developed.

The aim of the dead-reckoning technique is to trade accuracy for communication bandwidth, but in a clever way. An aircraft will, along with its position, make known to the radar how to best predict its path of motion up to x seconds into the future. The radar may then calculate for itself the aircraft's position as frequently as required. The aircraft will however keep track of were the radar thinks the aircraft is as the aircraft knows what prediction algorithm the radar is using. As soon as the aircraft's actual and predicted positions are outside a predefined error boundary of each other, the aircraft refreshes its current position and prediction method to the radar.

4. CONCLUSION

The analysis results of the discrete time simulator indicated that sequential hardware communication components of the infrastructure limit its scalability. A hybrid discrete time and discrete event modelling approach is proposed that will increase the scalability of the simulator by making more efficient use of the communication infrastructure while reusing the existing discrete time GBAD sub-system and system level models.

The proposed modelling aspects that will implement the hybrid modelling approach is *Aggregating DTSS Models into DEVS Models* and *Using Dead-Reckoning*. However, the open issues to investigate further are:

- The effect of this modelling approach on the simulation fidelity eg. the performance of the tracking filters once dead-reckoning is included, and
- how to compare the scalability of the discrete event simulator to that of the discrete time simulator.

Main advantage of this work is of course greater scalability of the real-time simulation capability, but additional advantages of the DEVS based simulator is:

- Improved scalability is the result of making more efficient use of the distribution infrastructure which implies that the simulator may now be distributed over longer distance lower bandwidth connections, and
- easier migration to The High Level Architecture (HLA), an IEEE standard for large scale distributed simulation interoperability, which is based on DEVS and popular within the military simulation domain.

5. REFERENCES

[1] B. Duvenhage and D. Kourie. Migrating to a real-time distributed parallel simulator architecture. In *Proceedings of the 2007 Summer Computer Simulation Conference*, San Diego, California, USA, 2007.

[2] J. Pretorius. Feasibility considerations for a tailored simulation based acquisition (SBA) approach. Master's thesis, University of Pretoria, Pretoria, South-Africa, 2003.

[3] J. Baird and J. Nel. The evolution of M&S as part of smart acquisition using the SANDF GBADS programme as an example. In *Proceedings of the 12th European Air Defence Symposium*, Shrivenham, England, 2005.

[4] S. Naidoo and J. Nel. Modelling and simulation of a ground based air defence system and associated tactical doctrine as part of acquisition support. In *Proceedings of the 2006 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, 2006.

[5] W. le Roux. Implementing a low cost distributed architecture for real-time behavioural modelling and simulation. In *Proceedings of the 2006 European Simulation Interoperability Workshop*, Stockholm, Sweden, 2006.

[6] B. Zeigler, T. Kim, and H. Praehofer. *Theory of Modelling and Simulation, second edition*. Academic Press, San Diego, California, USA, 2000.

[7] K. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5), 1979.

Biography

Bernardt Duvenhage obtained his B.Sc (Honours) degree in Computer Science from the University of Pretoria in 2005 and is currently pursuing a Masters Degree. While part of the Mathematical and Computational Modelling Research Group of the Council for Scientific and Industrial Research (CSIR) in South Africa, he played a key role in developing the group's distributed simulator architecture; the simulation's terrain and LOS services; and the 3D visualisation and analysis tool of the synthetic environment. He is currently employed in the Optronic Sensor Systems Competency Area of a division within the CSIR. He intends further research in virtual environment simulation and visualisation.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Quantising the Network Communication Between Discrete-Time Coupled Components

Bernardt Duvenhage and
Derrick G. Kourie
Espresso Group, University of Pretoria, South Africa
bduvenhage@csir.co.za and dkourie@cs.up.ac.za

June 25, 2008

SIMULATION Cover Letter

A coupled-component model of a system of systems (and of sub-systems) is currently in use to do high level predictive performance analysis. The current distributed parallel simulator and component models have evolved from a 100Hz discrete and logical time non-distributed simulation. Thus, the current internal model processing and the inter-model communication are *legacies* of an ancestral *high resolution discrete time* approach.

Experiments indicate that the discrete time simulator cannot support a parallelisation speed-up above 4—irrespective of the number of processing nodes. The reason for the scalability ceiling is found to be high communication latency due to the depletion of the giga-bit/s communication bandwidth that is available between processing nodes.

A *quantised system* approach to the *discrete event* representation of the model of a system of dynamical systems is proposed. Regardless of the benefits of the quantised system approach, the investment in the existing discrete time components and simulation architecture is a legacy to be carried forward by the simulator. This limits the feasibility of migrating the simulator and coupled component model to a fully quantised discrete event representation. (Related to the work by Zeigler et al.[1] and Kofman et al.[2] on the Theory of Modelling and Simulation and especially the efficiency of a quantised approach to the discrete event representation of a dynamical system. Related to the work by Schulze et al.[3] as it and other works by one of the authors, Straßburger, led to the broader investigation of: 1) the advantages of quantised discrete event, and 2) the advantages and cost of using a standardised interoperability architecture such as HLA. Related to the work by Wainer and Zeigler[4] with respect to their investigations of the closed loop error behaviour of quantised systems.)

In this article the authors have proposed a new approach in which partial quantisation can be deployed to potentially improve parallelisation speed-up in contexts which do not allow for full quantisation *or*, as in the authors' case, where full quantisation is not economically possible. This approach was successful in improving the parallelisation speed-up performance by lowering the required network bandwidth between distributed coupled components. The improved parallelisation speed-up performance approaches that of the ideal simulator, which has no distribution/communication overhead.

References

[1] B. Zeigler, T. Kim, and H. Praehofer. *Theory of Modelling and Simulation, second edition.* Academic Press, San Diego, California, USA, 2000.

[2] E. Kofman, J. Lee, and B. Zeigler. DEVS representation of differential equation systems: Review of recent advances. In *Proceedings of the 2001 European Simulation Symposium*, Marseille, France, 2001.

[3] T. Schulze, S. Straßburger, and U. Klein. Migration of HLA into the civil domains. *Simulation*, 73(5), 1999.

[4] G. Wainer and B. Zeigler. Experimental results of timed cell-DEVS quantisation. In *Proceedings of AIS'2000*, Tucson, Arizona, USA, 2000.

Biography

Bernardt Duvenhage obtained his B.Sc (Honours) degree in Computer Science from the University of Pretoria in 2005 and is currently pursuing a Masters Degree. Since being employed at the Council for Scientific and Industrial Research (CSIR) in South Africa, he has played a key role in developing a distributed simulator architecture; the simulation's terrain and line-of-sight services; and a 3D visualisation and analysis tool. He is currently involved in developing a physics-based synthetic environment and imaging simulator and intends further research in virtual environment simulation.

Derrick Kourie lectures in the Computer Science department at Pretoria University. While his academic roots are in operations research, his current interests include, but are not limited to software engineering and algorithm development. He is student adviser to some 20 postgraduate students working in these and related areas. He is editor of the South African Computer Journal and serves on various national and international academic committees.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Quantising the Network Communication Between Discrete-Time Coupled Components

Bernardt Duvenhage and
Derrick G. Kourie
Espresso Group, University of Pretoria, South Africa
bduvenhage@csir.co.za and dkourie@cs.up.ac.za
(Submitted to the journal SIMULATION, 2008-06-25)

Keywords: peer-to-peer, distributed parallel simulation, publish-subscribe, discrete-time, high real-time frame-rate, quantised system, discrete event

Abstract

Ongoing performance analysis experiments have relied on a coupled-component model of a system of dynamical systems. High resolution *discrete and logical time* distributed parallel components and simulator have previously been developed for execution of the model on a cluster of PCs. Earlier analysis of a typical experimental scenario had shown that the components and simulator were network bandwidth limited resulting in a parallelisation speed-up ceiling of 4, irrespective of the number of PCs in the cluster.

This article reports on a quantised system approach to more efficiently distribute the execution of such a coupled- component model. The coupled-components were aggregated into larger system-level models and coupled via quantiser and quantised integrator pairs (QQIPs). This new quantised architecture was analysed and found to no longer be bandwidth limited, but processor limited. The new architecture supports a scalable parallelisation speed-up that is nearly proportional to the number of processing nodes.

1. INTRODUCTION

A coupled-component model of a system of systems (and of sub-systems) is currently in use to do high level predictive performance analysis. Section 2. explains what is meant by a dynamical system and by predictive performance analysis. The notion of a coupled-component model is also discussed in reference to a real world example. The real-time requirement stems from the need to incorporate realistic human behaviour in the simulation.

The current distributed parallel simulator and component models have evolved from a 100Hz discrete and logical time non-distributed simulation. Thus, the current internal model processing and the inter-model communication are *legacies* of an ancestral *high resolution discrete time* approach. Section 3. details what is meant by discrete and logical time, and gives the definition of real-time as used in the article. The discrete time simulator is then discussed with extended treatment of the particulars.

Experiments indicate that the discrete time simulator cannot support a parallelisation speed-up above 4—irrespective of the number of processing nodes. The discrete time simulator is therefore unable to support the real-time execution of a large scale system of dynamical systems. Section 4. elaborates on parallelisation speed-up. It also discusses the setup and results of performance experiments regarding the large scale real-time use case of the discrete time simulator. Again a real-world example is used. The hardware infrastructure is a class 1 Beowulf[1] type cluster on a Gigabit Ethernet communication backbone. The reason for the scalability ceiling is found to be high communication latency due to the depletion of the giga-bit/s communication bandwidth that is available between processing nodes.

A *quantised system* approach to the *discrete event* representation of the model of a system of dynamical systems is proposed. Its advantage is that it requires a lower communication bandwidth and processing time than the discrete time coupled component model. Section 5. explains in greater detail what is meant by a quantised system approach, what is meant by quantised discrete event representation and points out the associated advantages. Regardless of the benefits of the quantised system approach, the investment in the existing discrete time components and simulation architecture is a legacy to be carried forward by the simulator. This limits the feasibility of migrating the simulator and coupled component model to a fully quantised discrete event representation.

However, to improve the parallelisation speed-up ceiling of the discrete time simulator, a quantised approach to *enveloping* groups of discrete time models is proposed. The reasoning behind and implementation of the model aggregation and the quantising envelopes is discussed in Section 6..

The performance results then indicate that the proposed modifications to the simulator architecture, changes the parallelisation speed-up to now be proportional to the number of processing nodes. This is because the model aggregation and the communication quantisation lowers the bandwidth requirements. Section 7. details the performance experiments and analysis. The results indicate that the updated architecture successfully allows a much larger system of dynamical systems models to be executed in real-time.

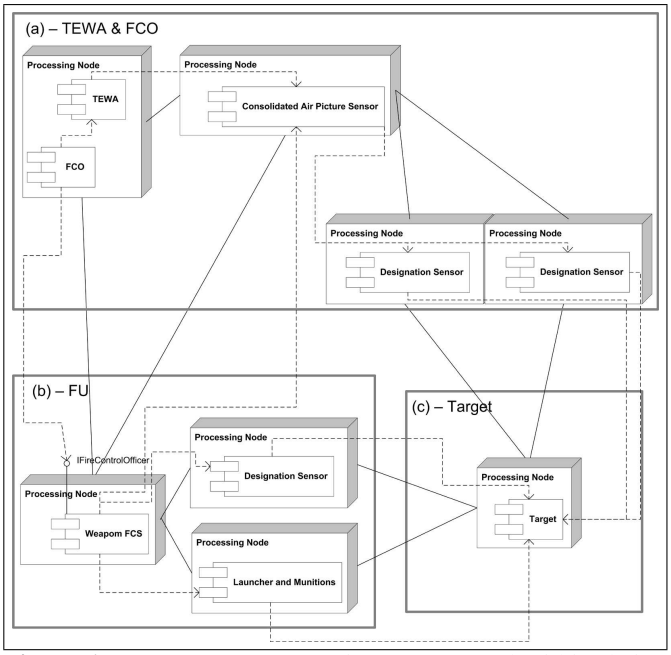

Figure 1. UML Component Diagrams (nested within their deployment diagrams) of the building blocks of a GBAD system of systems deployment

Section 8. concludes and offers suggestions on topics requiring further research.

2. THE COUPLED-COMPONENT MODEL AND DISTRIBUTED SIMULATOR

The dynamical system concept is a mathematical formalization for any fixed rule which describes the time dependence of a point's (could be a real number) position in its ambient space. An example is a mathematical model of the ballistic path of a projectile under a gravitational force.

According to Zeigler et al's M&S framework[2], a model of a dynamical system is only *valid* within a chosen experimental frame. Such a model is referred to as a lumped model, in that it is a simplification of the source system and might have lumped together—simplified—various potentially differentiatable aspects of the source system. The experimental frame therefore represents a subset of reality, but is valid for the experiments needed to be performed. Doing predictive performance analysis requires that the lumped model accurately, within the experimental frame, predicts behaviour of the source dynamical system not observed yet. This requires *predictive* validity [2] between the model, experimental frame and the source system.

An example of a *coupled component model* of a system of dynamical systems is shown in Figure 1. This figure depicts a coupled component model of a Ground Based Air Defence System (GBADS). Each component is a lumped model of some dynamical system.

The GBADS model is represented as a system of coupled air defence systems. Typically, each system's components are also differentiated into composing sub-systems. Figure 1 shows three system level components, indicated by the (a),(b) and (c), as groupings of sub-components. The Threat Evaluation and Weapon Allocation (TEWA) component provides information and potential weapon-to-threat allocations to the Fire Control Officer (FCO) terminal (or virtual operator). The TEWA gathers information from a consolidated air picture which contains the fused information of multiple designation sensors. The Fire Unit (FU) system level component receives *ordered* weapon allocations from the FCO. The generic FU is itself composed of three subsystems, namely the weapon Fire Control System (FCS), possibly another *local* designation sensor, and a launcher-and-munitions component. The targets (incoming airborne threats) are each modelled as an individual component which feeds the other system level components with the run-time threat scenario information. The indicated system level components are the building blocks of a full GBADS deployment which has one TEWA and FCO, but typically many FUs and multiple targets. The real-time requirement stems from the need to incorporate realistic human behaviour in the simulation through mock-up equipment consoles of, for example, the FCO's terminal. The couplings to the designation sensor outputs, the air picture outputs and especially the target outputs potentially *require relatively high bandwidth connections*.

The coupling of the systems provides *structural* validity to the coupled-component model. Structural validity is a stronger modelling relation than predictive validity and therefore also allows predictive performance analysis.

3. THE DISCRETE AND LOGICAL TIME IMPLEMENTATION

Logical time management requires the simulation to execute every simulated clock tick, stepping the entire system from one state to the next. Logical time simulation is sometimes referred to as, *As-Fast-As-Possible (AFAP)* simulation, because there is no need to synchronise the simulation time with a real world clock—a simulation merely takes as long as is required to simulate any given scenario.

Real-time simulation, on the other hand, actively tries to keep the simulation time synchronised with an external real world (wall) clock. A real-time simulation continually *jumps* forward in time as far as is needed to keep executing in real-time. As the intensity of the simulation increases (in respect of the number of simulation events and messages that have to be handled) more processing time is needed before the next time jump can be made. The size of the *time jumps*, however, has a direct relation to the accuracy—or time resolution—to which simulation causality can be guaranteed. At some point,

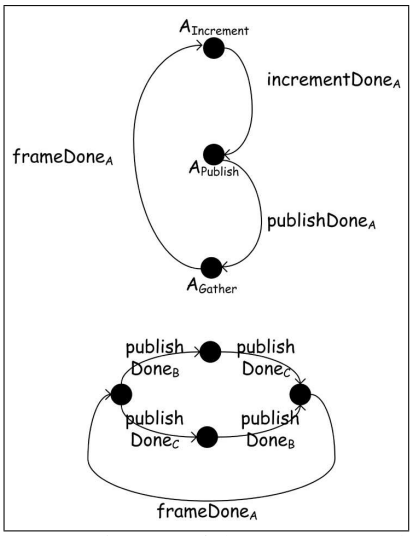

Figure 2. A CSP Diagram of the Increment Publish Gather Cycle composed of A and A_{sync}

either real-time execution must be sacrificed or causality is lost, along with the credibility of the simulation. Nevertheless, provided that simulation requirements are not excessive relative to the available hardware and software, real-time simulation may be based on *logical time* management. Such logical time simulation is designed to ensure, within performance limits, that its execution is throttled so that it does not exceed real-time. Whenever *real-time* is mentioned in the rest of this article, throttled logical time is implied except when indicated otherwise.

Discrete and logical time refers to the fact that the *clock ticks* of the simulation take place at regular simulation time intervals. In the case of the 100Hz discrete and logical time simulation under investigated, the simulation clock ticks happen every 0.01 seconds in simulation time. In the case of a coupled-component model, discrete time implies that the interactions between components also happen at regular simulation times—in the present case, at 100Hz.

At simulation run-time, each processing node goes through regular *increment, publish and gather cycles*. The *increment phase* updates each object's internal state from its *cached* inputs. The *publish phase* generates each object's output issues. Each issue is sent to the messaging layer, and is addressed to the appropriate subscriber (or subscribers, if there are multiple). The *gather phase* follows, during which each object (now in role of subscriber) *receives* the published issues from all other objects via the messaging layer, thereby refreshing the subscriber's input cache.

Such object interaction may be described and analysed using a formalism such as CSP[3]. To this end, let three interacting objects be represented by three discrete time processing nodes A, B and C. The CSP description for process A (visually shown in Figure 2) is:

$$\begin{aligned} A_{Increment} &= incrementDone_A \rightarrow A_{Publish} \\ A_{Publish} &= publishDone_A \rightarrow A_{Gather} \\ A_{Gather} &= frameDone_A \rightarrow A_{Increment} \end{aligned}$$

$$\begin{aligned} A_{sync} = (\quad & publishDone_B \rightarrow publishDone_C \rightarrow frameDone_A \rightarrow A_{sync} \quad | \\ & publishDone_C \rightarrow publishDone_B \rightarrow frameDone_A \rightarrow A_{sync} \quad) \end{aligned}$$

Similarly for process B:

$$\begin{aligned} B_{Increment} &= incrementDone_B \rightarrow B_{Publish} \\ B_{Publish} &= publishDone_B \rightarrow B_{Gather} \\ B_{Gather} &= frameDone_B \rightarrow B_{Increment} \end{aligned}$$

$$\begin{aligned} B_{sync} = (\quad & publishDone_A \rightarrow publishDone_C \rightarrow frameDone_B \rightarrow B_{sync} \quad | \\ & publishDone_C \rightarrow publishDone_A \rightarrow frameDone_B \rightarrow B_{sync} \quad) \end{aligned}$$

and for process C:

$$\begin{aligned} C_{Increment} &= incrementDone_C \rightarrow C_{Publish} \\ C_{Publish} &= publishDone_C \rightarrow C_{Gather} \\ C_{Gather} &= frameDone_C \rightarrow C_{Increment} \end{aligned}$$

$$\begin{aligned} C_{sync} = (\quad & publishDone_A \rightarrow publishDone_B \rightarrow frameDone_C \rightarrow C_{sync} \quad | \\ & publishDone_B \rightarrow publishDone_A \rightarrow frameDone_C \rightarrow C_{sync} \quad) \end{aligned}$$

$A = initialise \rightarrow A_{Increment}$ is the mutual recursive increment, publish and gather cycle of processing node A. For B and C defined similarly to A, the parallel composition $(A \,||\, A_{sync}) \,||\, (B \,||\, B_{sync}) \,||\, (C \,||\, C_{sync})$ describes the way in which the three processes jointly interact with one another. The composition is the description of the discrete time architecture and it can be shown that the overall interaction between the processes is deadlock-free[4]

Figure 3 shows a UML sequence diagram of the increment, publish and gather cycles and the *frameDone* messages. All messages are passed to the messaging layer that runs on TCP/IP and Gigabit Ethernet. The TCP/IP Nagle algorithm, as explained in [5], is disabled to allow the simulator finer control over message aggregation and therefore more control over message latencies.

Note that Figure 1 shows *processing node* components. These are in fact *virtual* processing nodes. They may be deployed on individual physical nodes of the cluster or they may be aggregated in various ways—the groups marked (a), (b) and (c) being one possible example.

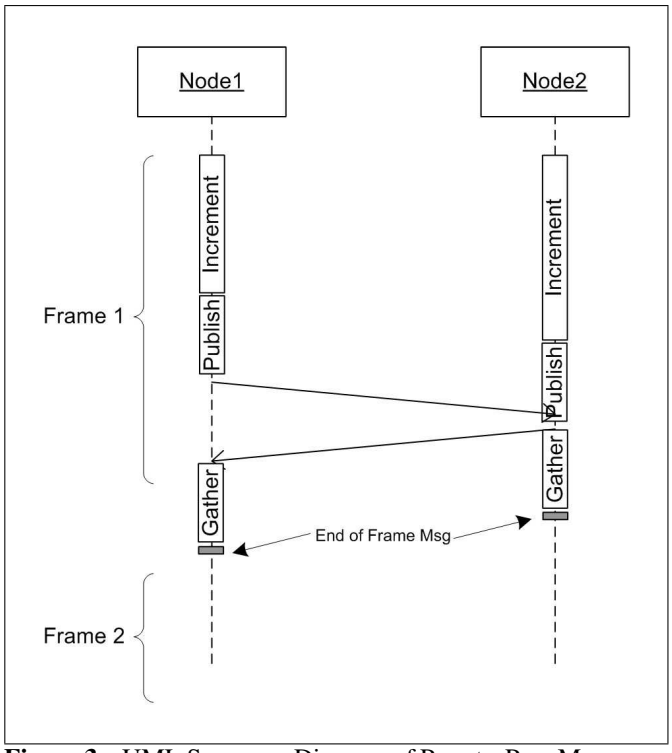

Figure 3. UML Sequence Diagram of Peer-to-Peer Message Passing and Simulation Synchronisation

4. PERFORMANCE ANALYSIS

The hardware infrastructure for the performance test is a class 1 Beowulf[1] type cluster of 11 PCs on a Gigabit Ethernet communication backbone. Each PC is a 3.0GHz Pentium 4 with 2GB RAM and an motherboard integrated Gigabit Ethernet NIC. Parallelisation speed-up of the simulation is used as a performance measure. In [1], parallelisation speed-up of a simulation on a cluster of p processing nodes, is defined as:

$$S(p) = \frac{Execution\ time\ on\ one\ processor}{Execution\ time\ on\ p\ processors}$$

The GBADS's benchmark scenarios' *building blocks* have been identified in Section 2.. The scale of each benchmark scenario is measured by the number of FUs in the scenario. Table 1 shows the list of objects in two of the smaller benchmark scenarios that are used. An FU component consist of a collection of sub-systems such as Gun1_Barrel1, Gun1_Barrel2, GFCS1 and PERFECT_TR1. Scenario A has one FU, and scenario B has two.

In the various performance analysis experiments, the FUs within each scenario were distributed approximately equally to the processing nodes. The number of FUs that were allocated per node was chosen so as to keep the distributed execution performance to just within real-time. Note that in the experiments, whenever an FU is added into a scenario, two targets are also added into the scenario. It will be seen that this is indeed also the case for the two scenarios shown in Table 1.

Table 1. List of Objects in Two of the Smaller Benchmark Scenarios

Scenario A	Scenario B
AIR_PICTURE_BOX	AIR_PICTURE_BOX
TEWA_BOX	TEWA_BOX
OIL	OIL
PERFECT_DR1	PERFECT_DR1
Gun1_Barrel1	Gun1_Barrel1
Gun1_Barrel2	Gun1_Barrel2
GFCS1	GFCS1
PERFECT_TR1	PERFECT_TR1
-	Gun2_Barrel1
-	Gun2_Barrel2
-	GFCS2
-	PERFECT_TR2
Target_Waypath1	Target_Waypath1
Target_Waypath2	Target_Waypath2
-	Target_Waypath3
-	Target_Waypath4

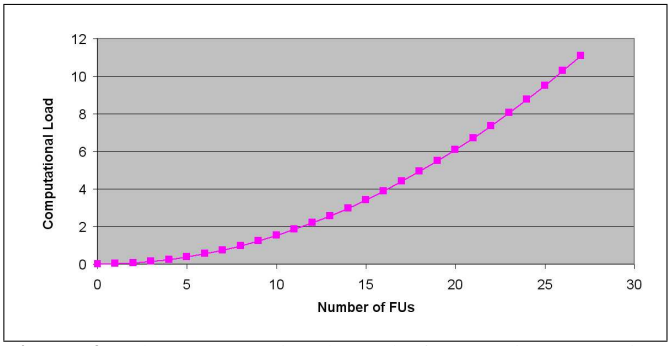

Figure 4. The Computational Load of Increasing Scenario Sizes

The measured AFAP execution time of a scenario is not linearly related to the number of FUs. The AFAP execution times for various scenarios run on a single processing node were measured. The scenarios varied in size from 0 to 27 FUs. These measured times were scaled by the duration of the scenario in simulation time, giving the so-called computational load of each scenario—i.e.

$$Comp.Load = \frac{Execution\ time\ of\ scenario\ on\ 1\ node}{Duration\ of\ scenario\ in\ simulation\ time}$$

Figure 4 shows the results obtained.

Curve fitting on the data obtained suggests the following approximate relationship between computational load and

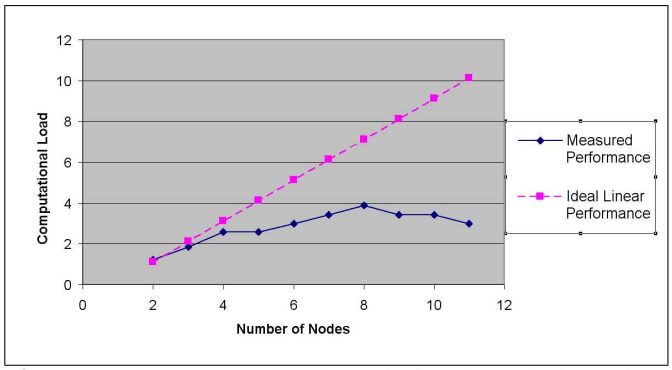

Figure 5. The Computational Load of Each Real-Time Distribution vs. The Computational Load of An Ideal Distributed Architecture

number of FUs in a scenario:

$$ComputationalLoad = \left(\frac{FuCount}{8.11}\right)^2$$

This means that a single node loaded with a scenario of 8 FUs runs the given scenario approximately in real-time; a node loaded with a scenario of 16 FUs runs approximately 4 times slower than real-time; a node loaded with a scenario of 4 FUs runs approximately 4 times faster than the real-time requirement; etc.

The computational load value for a scenario containing a given number of FUs corresponds *exactly* to the parallisation speed-up, S, that is required to execute the scenario in real-time. To see that this is indeed the case, observe that:

- if a scenario is executing in real-time on p processors, then the *duration of scenario in simulation time* exactly corresponds to the scenario's *execution time using p processors*; and
- under these circumstances, the formulas for parallelisation speed-up and computational load as defined above, are identical.

In the rest of the article real-time simulation is implied—except if otherwise stated—and computational load is used synonymously with parallelisation speed-up.

Using the quadratic computational load relation in Figure 4 allows the computational load, shown in Figure 5, of each measured real-time distribution (measured values not shown) to be estimated from the number of FUs in the distribution. Figure 5 shows that the maximum computational load—the maximum required parallelisation speed-up OR the scalability—achievable in the benchmark scenarios is about 4. Further, distributions over more than eight nodes become less efficient.

It is important to emphasise here that the measured parallelisation speed-up ceiling is only applicable to this specific application—thi specific processing to communication bandwidth ratio requirements. The parallelisation speed-up ceiling does however hold whether the scenario is meant to run real-time, half-real-time, twice real-time, etc. *In other words, any size scenario will execute in a shorter time on eight nodes than it would on any other number of nodes.*

Figure 5 shows not only the measured scalability performance of the current discrete time simulator, but also the theoretical scalability performance of the ideal distributed simulator. An ideal distributed simulator has a parallelisation speed-up of $S(p) = p$ and thus an unbounded scalability as the number of processing nodes is indefinitely increased. The difference between the current architecture and the ideal distributed architecture is due to the communication and time management overhead.

Duvenhage and Kourie[6] argue that the scalability ceiling of 4 is due to the sequential communication channel of each processing node. One limit of the sequential nature of the communication channel is a measured 41 message per simulation frame limit irrespective of message size. A second limit is the bandwidth upper limit of the underlying network technology which is in the order of 100 MBytes/sec for Gigabit Ethernet.

One way to overcome this limitation is to install multiple network cards into each PC, effectively connecting each node to its peers with two, three, four or more disjoint networks. This solution will improve the available bandwidth between peers by two, three, four or more times. However, the number of network cards that may be installed into a standard PC—especially with a technology like Gigabit Ethernet that requires a lot of system resources—is often limited to three or less. The simulator architecture modifications proposed below aims instead, drastically to improve the pattern of use of the sequential communication resources in order to steer the scalability behaviour towards the ideal distributed simulator case.

5. A QUANTISED SYSTEM APPROACH

As mentioned, a *quantised system* approach to the *discrete event* representation of the model of a system of dynamical systems, has the advantage of a lower communication bandwidth and processing time requirement than provided by the discrete time coupled component model.

Zeigler et al.[2] describe quantisation as providing a process for representing and simulating continuous systems that is an alternative to the more conventional discretisation of the time axis. It can also be said that time discretisation leads to discrete time systems and quantisation leads to discrete event systems. Additional discussions on the theory of quantised systems and related topics may be found in [7, 8, 9, 10, 11, 12, 13]. All of these resources also make note of the efficiency of a quantised system approach to the dis-

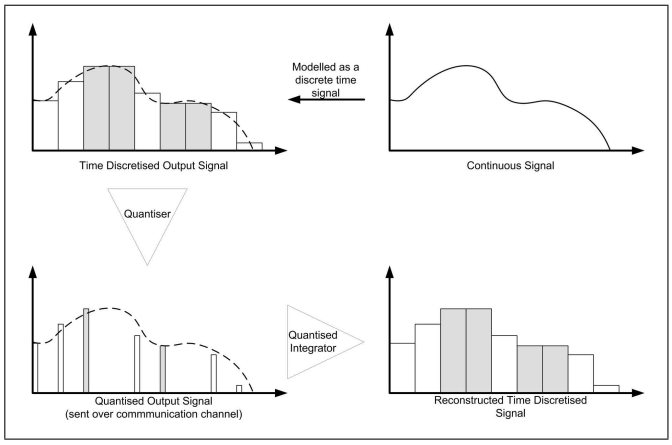

Figure 6. A signal modelled time discretised, then quantised and reconstructed by means of a QQIP

crete event representation of a dynamical system. Zeigler et al.[2] further say that a Quantised System (QS) can be decomposed into, and hence has the same behaviour as, a system—discrete time or continuous—*sandwiched* between Quantiser and Quantised Integrator Pairs (QQIPs).

As mentioned, this article focusses on a quantised system approach to an existing *discrete time* representation of a dynamical system. The quantisation is therefore *only* applied on the coupling of the components through QQIPs as discussed in the next section.

The primary goal of quantisation the couplings is to lower communication bandwidth usage. To accomplish this a model's inputs are fitted with quantised integrators and their outputs with quantisers to create a QS. Each communication channel between a sender and a receiver of information is therefore fitted with a Quantiser and Quantised Integrator Pair (QQIP). A signal, time discretised and then quantised and reconstructed by means of a QQIP is shown in Figure 6.

Quantisation lowers communication bandwidth at the price of a potential increase in the accumulated error of the simulation. Zeigler[14] and Zeigler et al.[2] have derived the upper bound on the accumulated error of closed loop DTSS simulation and also closed loop quantised DEVS simulation from the theory of quantised systems. The same accumulated error results have also been arrived at experimentally by Zeigler et al.[15] and Wainer and Zeigler[10], among others, who have done a cost/benefit analyses of reduced communication bandwidth and increased error due to quantisation.

The goal of the dead-reckoning technique[16], for example, is to trade accuracy for lower communication bandwidth usage. This is, of course, similar to quantisation. However, dead-reckoning attains its goal in a smarter *active* way. Dead-reckoning is usually applied in the quantisation of the position—and possibly its velocity and acceleration—of a body in inertial or non-inertial motion. The main difference from basic quantisation is that the quantised integrator also receives the algorithm to reconstruct/predict what the input to the quantiser would have been in between update events.

The dead-reckoning QQIP may apply a first order quantisation for following first order changes to a variable. An example is a body of mass following an inertial path. Second order quantisation may be applied to describe a body of mass under the influence of constant or time varying forces. One such an example is a ballistic munition that has a certain initial muzzle velocity and then gravity and drag forces act on it throughout its flight, giving it a curved ballistic path. Second order QQIPs is often used in the dead-reckoning.

For example, an aircraft may inform the radar not only of its current position, but also about how to best predict its path of motion up to x seconds into the future. The radar then calculates for itself the aircraft's position as frequently as required. However, the aircraft also keeps track of where the radar thinks the aircraft is, since the aircraft knows the prediction algorithm that the radar is using. As soon as the aircraft's actual and predicted positions are outside a predefined error boundary, the prediction at the receiver has become *stale*. Once this happens the aircraft *actively* refreshes its current position and prediction method to the radar—the aircraft sends a *path update* event. Both the original discrete time sampling and also the dead-reckoning error threshold result in the required discrete event hysteresis to ensure a valid DEVS model.

6. MIGRATING TO A QUANTISED SYSTEM APPROACH

It has been mentioned that, regardless of the benefits of the quantised system approach, the investment in the existing discrete time components and simulation architecture is a legacy to be carried forward by the simulator. This limits the feasibility of migrating the simulator and coupled-component model to a fully quantised discrete event representation. However, to improve the parallelisation speed-up ceiling of the discrete time simulator, a quantised approach to *grouping* and *enveloping* the discrete time models is proposed.

It has also been mentioned that quantisation lowers communication bandwidth and, potentially, the model's time complexity, at the price of a potential increase in the accumulated error of the simulation. The existing discrete time models have been conceptualised and developed with high resolution discrete time management in mind. Many of the models have therefore been validated within the 100Hz frame rate of the discrete time simulator. This makes them particularly sensitive to the errors introduced by QQIPs. Many of the sensor models, for example, were built and evolved to rely on a 100Hz target update rate. The models of the tracking filters of some of these sensor, for example, were based on actual

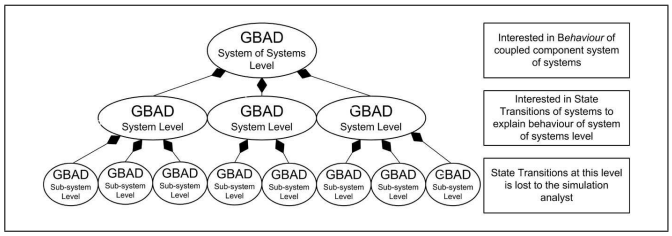

Figure 7. Double Structure Level of Discrete Time Simulator

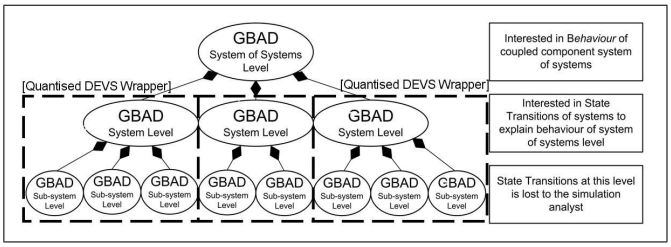

Figure 8. Double Structure Level of Discrete Time Simulator with Aggregation *Wrapper* Indicated

engineering text book and numerical method solutions. The 100Hz target data made it possible to build these types of *real world* solutions instead of the often more daunting behavioural solutions.

For the purpose of this article the quantum levels that are used for the quantisation as well as the threshold values for the dead-reckoning *path update* events are—as far as possible—made equal to the typical quantum levels that were implicit in the discrete time execution of the models. Choosing the quantum levels as discussed above can potentially result in quanta that are smaller than required which would in turn result in a sub-optimal bandwidth improvement. The nominated future work would improve on the optimal choice of quanta for the different communication channels and therefore optimise the bandwidth usage further. It is important to note that the current migration process to a real-time architecture is driven by the requirement to achieve the same accuracy as before, but to do so with greater efficiency.

6.1. Aggregation of Sub-System Models

Most of the current GBADS models are at the level of GBAD sub-systems of systems, such as the gun or FCS subsystems of the FU. These models are typically modelled at a state transition system specification level or higher. At the GBAD system level the sub-system models are brought together to create system level models—FUs—at a coupled system specification level as shown in Figure 7. The GBAD system models are then coupled again to create a GBAD system of systems level model—the GBADS deployment—also at the coupled-component system specification level.

The nature of the system of systems simulation experiments typically requires objectives and outcome measures at the GBAD system level—FU level. In such an *experimental frame* the output variables of sub-systems within the GBAD system level coupled-component models (the system structural knowledge) is hidden from the simulation analyst. It is therefore argued that access to the output variables of subsystem models—the system structural knowledge—is superfluous when analysing only the system level outputs. An *aggregated* structure for the GBADS simulator with the superfluous structural knowledge hidden in a wrapper is shown in Figure 8. The *quantised discrete event* nature of the wrapper will be discussed in the next subsection.

It should be noted that the same sequential communication components that were shown to exist in the discrete time simulator also exist in a discrete event simulator. The aim, however, is to avoid activating these sequential components unnecessarily. *Aggregation of the DTSS models* is the act of explicitly hiding the *double layer* of intermediate GBAD system structural information within a discrete event model. The new GBAD system level discrete event model—the aggregated FU—is then a state transition system specification envelope which shields the model interconnect infrastructure from the communication overhead of the internal structure.

Figure 1 shows the UML component diagram of the *aggregated* system level GBADS building blocks. The *use dependencies*—indicated by the dashed arrows—between system level building blocks indicate communication paths that could potentially still result in network communication.

6.2. Output Quantisers and Quantised Integrators

In the previous subsection, aggregation was proposed to remove some of the internal FU communication overhead from the system model. The remaining system level communication—use dependencies—can be classified into two types: events (be it voice network events or data); and state-like information such as platform position, velocity and orientation. Events are already quantised. However, the discrete time state information—discrete time sampling of what would in reality be a continuous variable—may often be quantised further.

The only use dependencies—indicated by the dashed UML arrows in Figure 1—that are chosen for application of QQIPs are the dependencies on the radar state output, the air-picture output and the target model output. These are the dependencies that require the high time resolution *state* information links between the aggregated system level models. The target flight profiles were defined as straight and level. This allowed—for analysis purposes—a simple piecewise constant QQIP to be applied to each dimension of the position,

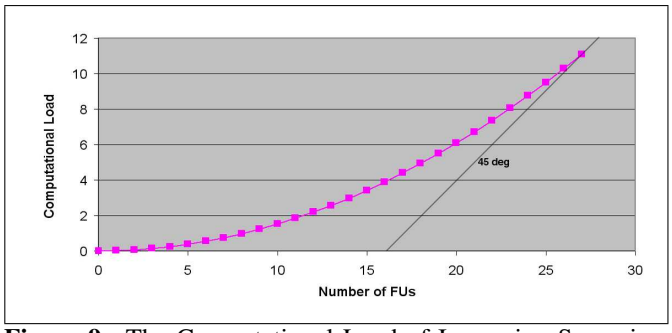

Figure 9. The Computational Load of Increasing Scenario Sizes

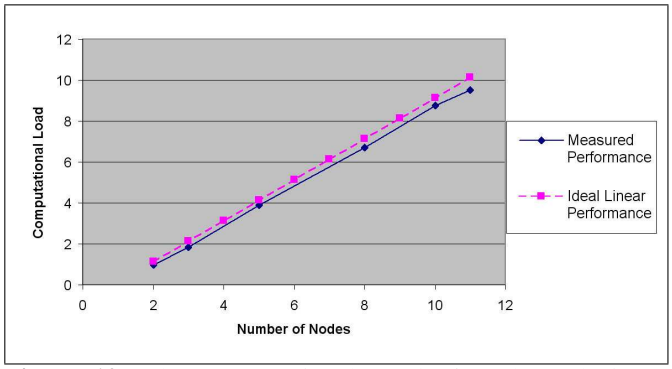

Figure 10. The Computational Load of Each Real-Time Distribution vs. The Computational Load of An Ideal Distributed Architecture

velocity and orientation of the state information, but with position extrapolated by the velocity in between path update events. The quantised *path update* events are generated at a fixed 10Hz and reconstructed at the quantised integrator—receiver—side. The reduction of the information update rate from 100Hz to 10Hz allows the quantised *use dependencies* to only use 10% of the bandwith of the unquantised *use dependencies*.

7. PERFORMANCE ANALYSIS

The benchmark scenarios are exactly the same scenarios that were used for the discrete time simulator in Section 4.. As before, the time taken to run a scenario on *a single node* is related to the number of FUs in that scenario approximately by a factor of $\left(\frac{FuCount}{8.11}\right)^2$. This factor referred to as the scenario's computational load. The computational load—which is also the parallelisation speed-up values required for real-time execution of these scenarios—of various FU counts, are therefore reflected by the same graph found in Section 4. This graph is reproduced here as Figure 9 for ease of reference and with the addition of a 45 degree reference line.

As in Section 4., using the quadratic computational load relation shown in Figure 9, allows the required parallelisation speed-up of each measured real-time distribution (measured values not shown) to be calculated; *the required parallelisation speed-up is in effect expressed as a function of the number of FUs*. The resulting parallelisation speed-up performance of the quantised simulator is shown in Figure 10. Note that a parallelisation speed-up ceiling *could not be found* over the 11 processing nodes that were available. Figure 10 shows that the quantised simulator's performance does indeed approach that of the *ideal distributed simulator*.

Also note that from Figure 9 it seems as if beyond scenario sizes of at least 27 FUs the computational load to FU gradient will rise above 45 degrees. This may be gauged from the 45 degree gradient line drawn in the figure. Taking the derivative of the computational load approximation of,

$$ComputationalLoad = \left(\frac{FuCount}{8.11}\right)^2$$

, to calculate the function gradient in fact gives

$$\frac{2}{8.11^2} FuCount$$

Solving this for a gradient of 1 results in an FuCount of 32.88. Beyond a gradient of 45 degrees at least one FU—and potentially every FU—has a computational load above one.

This is rather important as it implies that one or more FUs would have to be split up and be distributed over multiple processing nodes. Splitting an FU up into its parts will introduce a tear into the discrete event envelope constructed around each FU. This is expected to change the parallelisation speed-up results somewhat for scenarios larger than the ones currently analysed. It in fact points to a potential new stumbling block when migrating to GBADS scenarios that require a parallelisation speed-up of near and above 16.

8. CONCLUSION AND FUTURE WORK

The authors have proposed a new approach in which partial quantisation can be deployed to potentially improve parallelisation speed-up in contexts which do not allow for full quantisation *or*, as in the authors' case, where full quantisation is not economically possible. This approach was successful in improving the parallelisation speed-up performance by lowering the required network bandwidth between distributed coupled components. The improved parallelisation speed-up performance approaches that of the ideal simulator, which has no distribution/communication overhead. Some issues however remain to be addressed.

A study has not yet been done on accurately finding the accumulated sub-system and system of systems model errors—the closed loop error behaviour—for different quanta. It has been shown from the literature that the error behaviour generally improves with smaller quanta. A study is required on

the smallest quanta for which real-time execution is still possible. This should be addressed before a final decision is made to migrate the simulation to the proposed hybrid architecture. It should be noted here, that the simulation has not in the past, and should not in future, sacrifice accuracy for run-time performance. Maintaining accuracy is required because the simulation results feed into higher level decisions that rely on a certain level of predictive accuracy. The mind-set behind migration to a real-time architecture is to achieve the same accuracy as before, but to do it more efficiently.

The proposed hybrid simulator still employs discrete time models as per requirement to reuse existing models. This does however mean that the time complexity of the models and therefore the efficiency and scalability of the simulator will increase as the discrete time models are internally migrated to quantised discrete event modelling approaches. Researching a cost effective way of *also* doing such a migration to a new system specification could lead to further increases in the simulator's scalability in future.

Conservative time management—used in the current simulator—ensures a strictly increasing flow of time, but is not as efficient as optimistic time management. Optimistic time management may further improve simulation execution performance, but causality violations could potentially occur. When this happens, participating systems are expected to tolerate the violation and roll back their own state, and then fix and re-execute the simulation time line. The Operator In the Loop (OIL) interfaces, such as the FCO terminal does however require a strictly increasing and linear flow of time—at least within the requirement and perception of the systems and humans to which they are interfaced. More work is possibly required on incorporating Hardware In the Loop (HIL) and OIL capabilities into optimistic discrete event simulations before being able to take advantage of its added efficiency.

9. ACKNOWLEDGEMENTS

The authors would like to acknowledge the funding provided by the Armaments Corporation (ARMSCOR) of South-Africa, the South-African DoD, the CSIR, the University of Pretoria and the South African National Research Foundation without which this research would not have been possible.

REFERENCES

[1] B. Wilkinson and M. Allen. *Parallel Programming, second edition*. Pearson Education, Inc., Upper Saddle River, New Jersey, 2005.

[2] B. Zeigler, T. Kim, and H. Praehofer. *Theory of Modelling and Simulation, second edition*. Academic Press, San Diego, California, USA, 2000.

[3] A. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[4] B. Duvenhage. Migrating to a real-time distributed parallel simulator architecture. Master's thesis, University of Pretoria, Pretoria, South-Africa, 2008.

[5] B. Forouzan. *TCP/IP Protocol Suite*. McGraw-Hill, Inc., New York, NY, USA, 2002.

[6] B. Duvenhage and D. Kourie. Migrating to a real-time distributed parallel simulator architecture. In *Proceedings of the 2007 Summer Computer Simulation Conference*, San Diego, California, USA, 2007.

[7] B. Zeigler and J. Lee. Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment. In *Proceedings of SPIE - Enabling Technology for Simulation Science II*, Orlando, Florida, USA, 1998.

[8] H. Sarjoughian and B. Zeigler. Collaborative modeling: the missing piece of distributed simulation. In *Proceedings of SPIE - The International Society for Optical Engineering*, Orlando, Florida, USA, 1999.

[9] B. Zeigler, H. Song, T. Kim, and H. Praehofer. DEVS framework for modelling, simulation, analysis, and design of hybrid systems. *Lecture Notes in Computer Science - Hybrid Systems II*, 999, 1995.

[10] G. Wainer and B. Zeigler. Experimental results of timed cell-DEVS quantisation. In *Proceedings of AIS'2000*, Tucson, Arizona, USA, 2000.

[11] E. Kofman, J. Lee, and B. Zeigler. DEVS representation of differential equation systems: Review of recent advances. In *Proceedings of the 2001 European Simulation Symposium*, Marseille, France, 2001.

[12] B. Zeigler, M. Jamshidi, and H. Sarjoughian. Robot vs robot: Biologically-inspired discrete event abstractions for cooperative groups of simple agents. *Festschrift Conference in Honor of John H. Holland*, 1999.

[13] J. Nutaro, B. Zeigler, R. Jammalamadaka, and S. Akerkar. Discrete event solution of gas dynamics within the DEVS framework. *Lecture Notes in Computer Science - Computational Science – ICCS 2003*, 2660, 2003.

[14] B. Zeigler. DEVS theory of quantised systems. Technical report, University of Arizona, Tucson, Arizona, USA, 1998.

[15] B. Zeigler, G. Ball, H. Cho, J. Lee, and H. Sarjoughian. Bandwidth utilization/fidelity tradeoffs in predictive filtering. In *Proceedings of the 1999 Fall SISO Simulation Interoperability Workshop*, Orlando, Florida, USA, 1999.

[16] R. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley-Interscience, New York, USA, 2000.

Biography

Bernardt Duvenhage obtained his B.Sc (Honours) degree in Computer Science from the University of Pretoria in 2005 and is currently pursuing a Masters Degree. Since being employed at the Council for Scientific and Industrial Research (CSIR) in South Africa, he has played a key role in developing a distributed simulator architecture; the simulation's terrain and line-of-sight services; and a 3D visualisation and analysis tool. He is currently involved in developing a physics-based synthetic environment and imaging simulator and intends further research in virtual environment simulation.

Derrick Kourie lectures in the Computer Science department at Pretoria University. While his academic roots are in operations research, his current interests include, but are not limited to software engineering and algorithm development. He is student adviser to some 20 postgraduate students working in these and related areas. He is editor of the South African Computer Journal and serves on various national and international academic committees.