uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

u Ottawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

El Ajaltouni

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.

GRADE / DEGREE

Department of Computer Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Efficient Dynamic Load Balancing techniques for Large Scale Distributed Simulations on a Grid Infrastructure

TITRE DE LA THÈSE / TITLE OF THESIS

A. Boukerche

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

El Saddik

G. Wainer

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

EFFICIENT DYNAMIC LOAD BALANCING TECHNIQUES FOR LARGE SCALE DISTRIBUTED SIMULATIONS ON A GRID INFRASTRUCTURE

by

ELIE ANTOINE EL AJALTOUNI

A THESIS SUBMITTED IN

PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

in

COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING S.I.T.E.

UNIVERSITY OF OTTAWA

April 2009

Your file Votre référence
ISBN: 978-0-494-59462-9
Our file Notre référence
ISBN: 978-0-494-59462-9

Abstract

Dynamic load balancing is a key factor in achieving high performance for large scale distributed simulations on grid infrastructures. In a grid environment, the available resources and the simulation's computation and communication behavior may experience run-time critical imbalances. Consequently, an initial static partitioning should be combined with a dynamic load balancing scheme to ensure the high performance of the distributed simulation. In this paper, we propose a dynamic load balancing scheme for distributed simulations on a grid infrastructure. Our scheme is composed of an online network analyzing service coupled with monitoring agents and a run-time model repartitioning service. We present a hierarchical scalable adaptive JXTA service based scheme and demonstrate through simulation experiments that our proposed scheme exhibits better performance in terms of the simulation execution time. Furthermore, we extend our algorithm from a local intra-cluster algorithm to a global inter-cluster algorithm and we consider studying the proposed global design through a formalized Discrete Event System Specification (DEVS) model system.

Table of Contents

List of Tables

List of Figures

List of Acronyms

HLA High Level Architecture

RTI Run Time Infrastructure

SOA Service Oriented Architecture

LAN Local Area Network

SAN System Area Network

WAN Wide Area Network

PDP Peer Discovery Protocol

PBP Pipe Binding Protocol

ERP Endpoint Routing Protocol

RVP Rendezvous Protocol

PIP Peer Information Protocol

PRP Peer Resolver Protocol

DEVS Discrete Event System Specification

OLNAS Online Network Analyzing Service

MA Monitoring Agent

RTMRPS Run-Time Model Repartitioning Service

GOLNAS Global Online Network Analyzing Service

GRTMRPS Global Run-Time Model Repartitioning Service

GMA Global Monitoring Agent

List of Publications

The following publications are relevant to the topic of this thesis and have been authored by Elie El Ajaltouni.

Conferences and Workshops:

1. Elie El Ajaltouni, Azzedine Boukerche, Ming Zhang. An Efficient Dynamic Load Balancing Scheme for Distributed Simulations on a Grid Infrastructure. In proceedings of 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'08), pages 61-69, 2008.

2. Elie El Ajaltouni, Azzedine Boukerche, Ming Zhang. A Grid-Based DEVS Approach to Dynamic Load Balancing for Large Scale Distributed Simulations. Published in 42nd Annual Simulation Symposium (ANSS 2009).

Acknowledgements

I would like to express my deep-felt gratitude to my supervisor, Prof. Azzedine Boukerche of the School of Information Technology and Engineering (S.I.T.E) at The University of Ottawa, for his advice, encouragement, enduring patience and constant support. Prof. Azzedine's financial support was a great relief for me to focus and concentrate on my research studies. He was never ceasing in his belief in me, always providing clear explanations, constantly driving me with energy and enthusiasm, and always, giving me his time, in spite of anything else that was going on. I wish all students the honor and opportunity to experience his ability to perform at that amazing and very cooperative role.

I also would like to thank all the PARADISE Research Lab members for their continuous support and collaboration. They all gave their best to help me finish this work. This includes all great people in PARADISE including Mr. Abdulaziz Al Hamidi, Mr. Ahmad Shadid, Dr. Ming Zhang, Mr. Robson Eduardo De Grande and Mr. Luqman Ahmad. No doubt that ALL PARADISE members are great people and my appreciation could never be described in words, I just say THANK YOU!

Additionally, I want to thank The University of Ottawa School of Information Technology and Engineering (S.I.T.E) Department professors and staff for all their hard work and dedication, providing me the means to complete my degree and prepare for a career as a computer scientist.

to my

GRANDMOTHER, MOTHER, FATHER and BROTHERS

with love and appreciation

Chapter 1

Introduction

Running a large-scale distributed simulation generally requires a large amount of computing resources that are at geographically distributed locations. The imbalance of workload in a distributed simulation is one of the key factors that lead to the degradation of overall performance of simulation execution. Considering the hard-to-predict run time behaviors of most simulation applications, static workload partition mechanism cannot work effectively in many cases. Consequently, the dynamic load balancing mechanisms are called on, which aim to achieve performance improvement by balancing the workload during simulation runtime. Dynamic load balancing schemes use run-time system state information to make decisions on how to move work from one host to another during execution. However, the overhead due to the dynamic migration of simulation entities needs to be carefully considered in order to gain positive performance improvement. As we know, High Level Architecture (HLA) [2],[3],[4] has been the dominant distributed simulation standard for the past few years. However, the HLA relies on the Runtime Infrastructure (RTI), which limits its flexibility and scalability. Also, there is no built-in fault-tolerance or load balancing mechanisms in HLA. In order to address these problems, a lot of research have been conducted which aim at enhancing the HLA/RTI by adding load balancing capabilities through federate migration. Some other approaches focused on using the concept of Service Oriented Architecture (SOA) to encapsulate the core functionalities of HLA as Grid services. In the mean time, Peer-to-Peer based technology, such as Sun's JXTA, has also been used as middleware to support distributed simulation, which can overcome some of the existing difficulties in HLA and other distributed simulation frameworks. In this thesis, we propose a dynamic load balancing scheme in the context of a JXTA based

distributed simulation framework [1]. Our algorithm focuses on several JXTA services including a time management service, an on-line network analyzing service and a run-time model repartitioning service. We use monitoring agents to collect runtime system state data, and use an on-line network analyzing service to achieve a dynamic load balancing strategy. Finally, we use the run-time model repartitioning service to apply the dynamically generated load balancing strategy to the system. In our proposed algorithm, both computation and communication workloads are considered as sources of potential imbalances that affect the performance of the distributed simulation. Moreover, our proposed scheme follows the hierarchical design approach to meet the requirement of large-scale grid based distributed simulation applications. As a mater of fact, in this thesis, we propose and demonstrate a dynamic load balancing scheme for more efficient distributed simulations. We will verify our idea through simulation experiments conducted on an IBM Linux cluster located at PARADISE Lab. The remainder of this thesis is organized as follows: Chapter 2 depicts the background for our work. Chapter 3 presents some related work. Chapter 4 describes in detail our proposed dynamic load balancing scheme. Chapter 5 demonstrates our simulation experiments results. Chapter 6 extends our local intra-cluster scheme into a global inter-cluster scheme. Chapter 7 studies the proposed global design through a formalized Discrete Event System Specification (DEVS) model system. Finally, we conclude this thesis and state some future work in Chapter 8.

Chapter 2

Background

In this chapter, we introduce the background of our work. We first define JXTA, specify its characteristics and discuss its feasibility for building large scale grid based distributed simulation frameworks. We then describe a proposed solution [1] of a service oriented JXTA based framework in which we implement our dynamic load balancing algorithm.

2.1 JXTA'S Feasibility for Building Distributed Simulation Frameworks

2.1.1 JXTA Specifications and Characteristics

Peer-to-peer based network computing infrastructure has attracted more attention with the development of internet-scale collaborated distributed computing in recent years. JXTA is a product of many years of effort to standardize the essential functionalities of peer to peer networks, see Figure 2.1. In this section, we will describe some of the key aspects of SUN's JXTA technology focusing on its ability and possibility for building large-scale and dynamic reconfigurable distributed computing systems. We will also see how it fits into the foundational framework for building and designing grid-based distributed simulation systems.

As a peer-to-peer based networking framework, JXTA is a set of protocols for designing highly dynamic and self-aware heterogeneous virtual networks. The key attributes of JXTA are its interoperability, platform independency, and ubiquity [23]. JXTA employs XML for common message encoding and service advertisement, and has a built-in service discovery

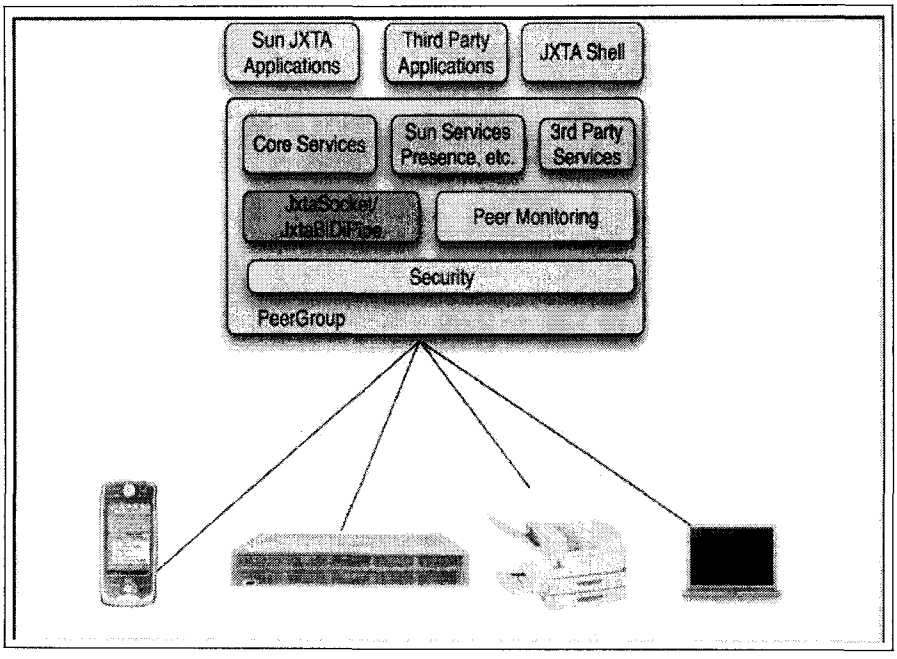

Figure 2.1: **JXTA layered Architecture by Sun [23]**

protocol and message routing protocol. Therefore, a native JXTA based virtual distributed network is capable to support dynamic join/resign simulation entities through the peer discovery protocol. As well as it is able to support fault-tolerance through peer group service. The six core JXTA protocols are the following:

- Peer Discovery Protocol (PDP)
- Pipe Binding Protocol (PBP)
- Endpoint Routing Protocol (ERP)
- Rendezvous Protocol (RVP)
- Peer Information Protocol (PIP)
- Peer Resolver Protocol (PRP)

The concept view of overall JXTA architecture is shown in Figure 2.2.

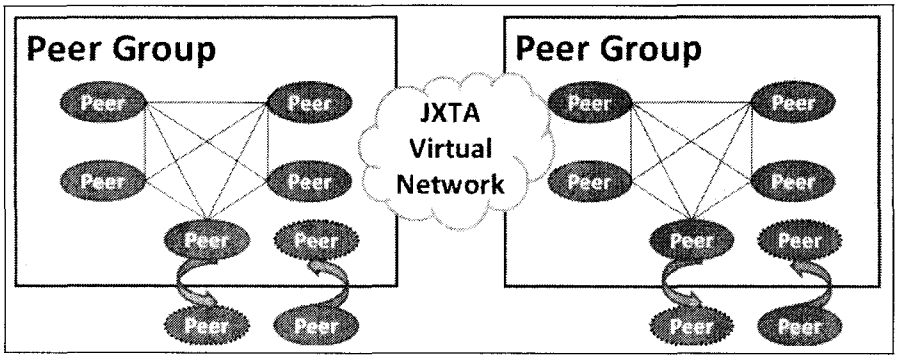

Figure 2.2: **JXTA Concept View[1]**

2.1.2 Previous Work using JXTA Technology

Some progress has been made with regard to applying JXTA technology to some existing distributed simulation frameworks. The authors in [28] has proposed a JXTA based peer-to-peer resource management scheme for HLA based distributed simulations and confirmed the feasibility of using JXTA for improving the reusability of simulation components. Their framework supported the pre-runtime setup of HLA federates using downloadable configuration files. The authors in [29] discussed and studied the feasibility of JXTA for collaborative game applications, and concluded that JXTA is quite suitable for building such collaborative distributed applications. This also implies the feasibility of using JXTA to construct distributed interactive simulations. In terms of real-time capable distributed systems, the authors in [30] studied the feasibility of using JXTA to build a near real-time Peer-to-Peer distributed system, and found that the resolver service is suitable for real-time systems whereas the default XML handling protocol is unacceptable for real-time constraints due to using a light XML parser aimed at small memory devices. However, JXTA is an extremely flexible framework, and almost any available network transporting protocol and XML based message handler can be used to improve its real-time capability. As a result, a high-performance XML parser and high-speed network protocols could be applied to satisfy the requirement for real-time simulation applications. JXTA based network services are different from standard web services; however, it is not difficult to create

a proxy to merge them into a unified framework. The authors in [31] discussed the interoperability of the JXTA based framework with the standard Web service components. They proposed and implemented a web service proxy for their original designed JXTA based framework, through which JXTA based service components can talk with standard web services. Other research work has been done to benchmark JXTA in different networking environments to address the performance of JXTA in these environments. The authors in [25] benchmarked JXTA in a single LAN network environment using JXTA Java binding, and the results of the experiments showed that the startup overhead of JXTA network is relatively high compared to other operations. The pipe binding is also a high cost operation, but can be reduced by reuse the components. The authors in [26] evaluated the JXTA performance in terms of its suitability in a high-performance grid based network infrastructure. They discovered and found that JXTA is suitable for delivering high performance in a System Area Network (SAN) or Wide Area Network (WAN) if and only if JXTA is fine tuned for communication performance. Furthermore, the authors in [32] benchmarked the JXTA communication layer in terms of JXTA Java and C language bindings over a fast local Ethernet environment, and showed useful hints for how to effectively use the JXTA communication layer to suit different application domains. As we have seen from the above discussions, the feasibility of using JXTA to build grid based high-performance real-time distributed simulation frameworks can be confirmed, although such systems need to be precisely designed to meet the required specifications.

2.2 Proposed Solution of a Service Oriented JXTA Based Framework

2.2.1 System Architecture

As shown in Figure 2.3, the authors in [1] proposed and discussed JXTA based hierarchical distributed simulation framework. This framework consists of a three layered

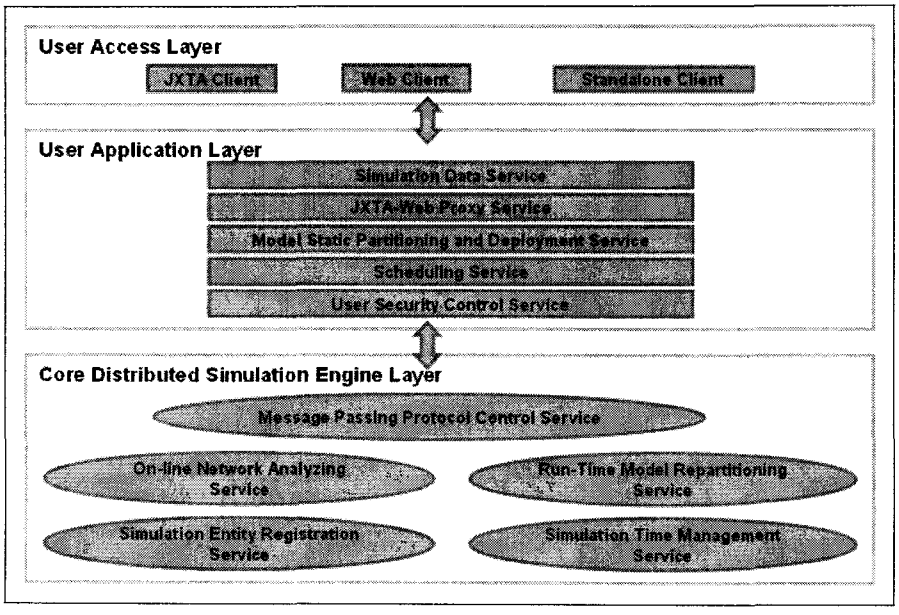

Figure 2.3: **System View as a Multi-layered JXTA-core Service Oriented Architecture[1]**

architecture which is composed of:

- User Access Layer
- User Application Layer
- Core Distributed Simulation Engine Layer.

For The User Access Layer, it has different types of user applications incorporating JXTA based, standalone, or browser based client. The purpose of the User Access Layer is to submit simulation tasks and jobs to the User Application Layer services. These services include simulation model or model Meta data, application requirement for example real-time constraints.

The User Application Layer is considered as bridge between the User Access Layer and Core Distributed Simulation Engine Layer. For the Core Distributed Simulation Engine Layer, it consists of a set of services and proxy service to handle special and different types

of user applications. The services in the User Application Layer are responsible for sending and receiving simulation data between user access and the core simulation engine. There are numerous key services in this layer. These services are:

- User Security Control Service
- Scheduling Service
- Model Static Partitioning and Deployment Service
- Simulation Data Service
- JXTA-Web Proxy Service

The Core Distributed Simulation Engine Layer is the mainly significant layer in the overall framework. It is completely JXTA based in that it comprises of interconnected simulation clusters. The key services running in this layer include:

- Simulation Entities Registration Service
- Simulation Time Management Service
- On-Line Network Analyzing Service
- Run-Time Model Repartitioning Service
- Message Passing Protocol Control Service

As a result, this overall proposed framework is specially designed for large-scale grid based computing infrastructure.

2.2.2 System Sevices

The system services of Multi-layered JXTA-core Service Oriented Architecture are as follows:

1. User Application Layer Service The services in this layer provide the fundamental functionalities for scheduling and deploying user simulation applications to the Core Distributed Simulation Engine, as well as support the user authentication, simulation data acquisition, and non-JXTA based user applications. A small description of each service is given below as follows:

A. User Security Control Service (USCS): This service applies certain security mechanisms to recognize users before user applications can submit simulation jobs. It also provides protection against accessing crucial simulation data.

B. Scheduling Service (SC): This service checks the status of clusters used in the simulation, and then produces suitable lists of clusters and nodes for usage by the Model Static Partitioning and Deployment Service. The simulation job also is scheduled for a future time in case of not enough clusters computing resources are obtainable.

C. Model Static Partitioning and Deployment Service (MSPDS): This service obtains the submitted model or model meta data from the user application to generate an initial and static model partition plan to the available clusters, and then load the models to the group heads and local heads nodes of participating clusters, which then deploy the obtained sub-models to the computing nodes.

D. Simulation Data Service (SDS): This service communicates directly with the group heads or local heads in the Distributed Core Simulation Engine to get the essential simulation data that the user applications need. It can as well obtain simulation data from any participating data sources within the virtual network. This service is the key data service between the user applications and core simulation engine.

E. JXTA-Web Proxy service (JWPS): This proxy service is particularly intended to maintain standard Web clients that do not address JXTA. It is used to enlarge the user applications to any probable type.

2. Core Distributed Simulation Engine Layer Services In this fully JXTA based layer, all of the key services are implemented as JXTA peer group services, through which any single failure of the key service will not affect the continued execution of the overall distributed simulation. This is due to the original purpose of JXTA's design: to provide fault tolerance for crucial peer group services.

In this layer, [1] defines two peer groups: local peer group and global peer group. Local peer group is a cluster based group that is formed by the participating computing nodes within it. There are at least two local heads in each local peer group, which are implemented as Rendezvous Peers and run the key services. The global peer group consists of the nodes that are the local heads of these local peer groups. There are at least two global heads that are located in one of the participating clusters, which run the key distributed simulation related services. It is worth noting that the local peer group services are extended from the service of the global peer group; as a result, they have all the functionalities of the global peer group service but with added capabilities to handle cluster specified tasks. The core services architecture in the Core Distributed Simulation Engine is hierarchical and can provide multi-level simulation entities control and tasks coordination. [1] presents here also a short description of each key service:

A. Simulation Entities Registration Service (SERS): This service runs on a JXTA Rendezvous Peer Node. If it runs as a local peer group service, it handles the static registration of participating simulation entities within a cluster as well as deals with the dynamic registration of a new simulation entity asking to join or leave the local cluster. If it runs as a global peer group service, it handles the static and dynamic registration of local heads in a similar way.

B. Simulation Time Management Service (STMS): This service is the most important service in the framework that manages the time advances of each simulator running on peer nodes. The local STMS manages the time advances of all simulators within the local cluster, and coordinates with the global STMS to synchronize time with other nodes in other clusters. In other words, the time advance of each individual simulator

is controlled by both the local and global STMS.

C. On-Line Network Analyzing Service (OLNAS): This service is used to analyze run-time network message passing latency to assist in building a most favorable load balancing mechanism. This service observes the workload on each peer node, and studies the present computing and communication workload distribution to decide how to equilibrium the overall workload in a local cluster and then, in the overall cluster of clusters. The roles of the local and global OLNAS are much related to the local and global STMS mentioned above.

D. Message Passing Protocol Control Service (MPPCS): This service is being utilized to dynamically decide and apply the message passing protocol with respect to the application specifications, which is particularly valuable for real time distributed simulations that required making use of the best available network transport protocol to meet the time constraints of the applications. Likewise, the global MPPCS applies the message passing protocol for inter-cluster communication, whereas local MPPCCS decide the message passing protocol for the peer nodes within a local cluster.

E. Run-Time Model Repartitioning Service (RTMRPS): This service is used to run-time simulation reconfiguration. It communicates with the On-Line Network Analyzing Service to dynamically determine on a repartition plan, and also communicates with the Simulation Time Management Service to pause and resume the simulation time advance. Likewise, global RTMRPS determines the inter-cluster based repartition plan, whereas the local RTMRPS takes care of the repartitioning of the simulation entities within a local cluster.

Chapter 3

Related Work

Dynamic load balancing for distributed simulations has received much interest due to its potential performance gain. Many algorithms [5]-[22] have been developed to tackle the problem for distributed simulations running on different workstations, clusters and the grid . The authors in [5] propose an algorithm that uses two central metrics: The processor advance time (PAT) and the cluster (of LPs) advance time (CAT) which are the amount of wall clock time required by a processor and a cluster respectively to advance one unit of simulated time in the absence of rollback. The load balancing algorithm first sorts the usable set of processors based on their PAT values and then traverses the sorted set, migrating (clusters of) LPs from processors with large PAT values to those with lower PAT values so that the maximum difference between the PAT values in any two processors is minimized. The clusters to migrate are decided by CAT which defines the amount of computation required to advance a cluster one unit of simulated time in the absence of rollback. Moreover, the authors in [6] extends the algorithm presented in [5] to take communication latencies into account. It defines two additional metrics to be considered for the algorithm's migration decision, the Processor-to-Processor Communication and Cluster-to-Processor Communication that indicate the total number of messages exchanged between two processors (between any LPs) and between a cluster and a processor (any LP on that processor) respectively within a time interval. Most of the research in [9] - [11] uses the grid services and the HLA / RTI for distributed simulations creating a middleware between the HLA and the grid services. The authors in [13] developed a new dynamic partitioning algorithm based on performance estimates whose calculation is completely platform independent. Moreover, aging functions were defined to increase the robustness of the algorithm

and a set of parameters allowed user configuration to meet specific scenarios. However, their host capacity estimation implicitly assumed that event execution has a uniform cost. The authors in [12] perform dynamic load balancing in two steps. First, a central controller collects information about the pending load distribution and the variance of the CPU usage of the participating hosts. Then, it determines the amount of load to transfer between hosts. In a second step, the hosts decide which simulation objects to move according to different policies. Among others, a communication-based policy may be selected. However, none of their different policies defined to decide on which simulation object to move consider the combined computation and communication implications. The authors in [14] proposes a dynamic load balancing algorithm based on a defined cost model. It showed using different simulation models that to achieve consistent performance imbalances, as well as lookahead between processors need to be considered. Furthermore, the authors in [15] proposed a dynamic load balancing algorithm which assumes no knowledge about the workload parameters based on the CPU-queue length. Two variations of the algorithm were designed, the centralized and a multi-level methods. The centralized approach makes use of a central processor to collect and disseminated the loads for each processor while the hierarchical method use a central approach at level one and a distributed strategy at level two, where processors communicate and share LPs' information only with their immediate neighbors. It is worth mentioning that a new metric for measuring the workload is defined in [16] for both optimistic and conservative simulations. In their approach, an architecture for load distribution services over a multi-threaded HLA / RTI is described, and these services aimed to improve the overall simulation performance. The authors in [7] and [8] introduces agents for achieving load balancing over the grid. In [7] the authors uses a combination of intelligent agents and multi-agent approach. The agents are used as representatives of grid resources and utilize predictive application performance data and iterative heuristic algorithms to engineer load balancing across multiple hosts. In [8] the agents are responsible for sensing the context of the Grid, and serve as the scheduler for dynamic load adjustment. The authors in [19] described a dynamic load balancing algorithm

consisting of three components: (1) load sensor (2) load evaluator and (3) load adapter. The load sensor computes the Virtual Time Progress(VTP) for each simulation process. The load evaluator decides whether to launch process migration or not, depending on ratio between the actual and the predicted VTP. Process migration is then controlled by the load adapter. The algorithm aims at reducing the lag between the VTPs of the processors for a better simulation performance. Burdorf and Marti [20] presents initially a static balancer that assigns objects to processors according to the load, which is gathered by running a pre-simulation. During the simulation, the balancer records the smallest simulation time of all objects on each machine and seeks to minimize the variance between the objects' simulation times. The work in [21] studies both static partitioning and dynamic load balancing strategies. The static partitioning strategy attempts to minimize load imbalances. The dynamic load balancing scheme works together with the static partitioning strategy and tries to schedule LPs dynamically to threads. The authors in [22] compare the performance of three metrics for use with a token-based dynamic load balancing algorithm. The three defined metrics were processor utilization (PU), processor advance simulation rate (PASR), and a combination of these metrics. We refer to [17][18] for another approach to load balancing through interest management.

We conclude our related work section by outlining the differences between other dynamic load balancing schemes for distributed simulations and ours. We argue that a dynamic load balancing scheme should optimize both computation and communication workloads. The performance approximation metrics we used are completely platform independent and are based on the simulation entities' activity with no priori knowledge needed. As a matter of fact, none of the algorithms we have surveyed provides an adaptive load differentiation mechanism based on the simulation entities' dynamic behavior. Therefore, in our approach, "priorities" are introduced to the different computation and communication load-balancing phases and are dynamically modified to meet the requirements of the distributed simulation. Indeed, our proposed scheme coupled with the system architecture ensures a scalable hierarchical design: a local intra-cluster load balancing scheme and its

Table 3.1: Differences between Dynamic Load Balancing Designs

	Balancing Computation and Communication	Platform Independent	Adaptive Load Balancing	Hierarchical Scalable Design	No Pre-Defined Performance Metrics
Proposed Design	Yes	Yes	Yes	Yes	Yes
Fujimoto et al.[5]	No	Yes	No	No	No
Jiang et al.[6]	Yes	Yes	No	No	No
Cao et al.[7]	No	Yes	No	Yes	Yes
Wang et al.[8]	No	Yes	No	Yes	Yes
Wilson et al.[12]	No	Yes	No	No	No
Patrick et al.[13]	Yes	Yes	No	No	No
Low et al.[14]	Yes	Yes	No	No	Yes
Boukerche et al.[15]	No	Yes	No	Yes	No
Tan et al.[16]	No	Yes	No	No	Yes
Gan et al.[21]	No	Yes	No	No	Yes
El-Khatib et al.[22]	No	Yes	No	No	Yes

extension into a global inter-cluster load balancing scheme. Moreover, our algorithm is JXTA service based, and such modular design provides independence among the different simulation services favoring any future service or architecture extendibility. These characteristics will be clear after the description of the proposed design in Chapter 4. Table 3.1 summarizes the differences between some dynamic load balancing schemes and ours.

Chapter 4

Our Proposed Dynamic Load Balancing Algorithm: Design and Implementation

In this section we present our dynamic load balancing scheme in detail. We first outline our system architecture on which our load balancing scheme is designed. We then present and discuss our intra-cluster load balancing algorithm.

4.1 System Architecture

Our architecture is illustrated in Figure 4.1, in which each individual cluster is formed by a number of peer nodes. Every node has one or more simulation entities running with a monitoring agent. Each cluster has two local head nodes to run the key services (time management service, online network analyzing service, run-time model repartitioning service) used by our scheme and a global monitoring agent. The local head nodes are responsible for the load balancing of the intra-cluster distributed simulation. Fault tolerance is realized through the duplicate local head [1]. All the services and monitoring agents are introduced in both local and global levels. Two global head nodes located at one of the participating clusters run the global key services (global time management service, global on-line network analyzing service, global runtime model repartitioning service). The global head nodes combine with the local head nodes to form the global virtual cluster. The global head nodes are responsible for the overall load balancing of the inter-cluster distributed

simulation.

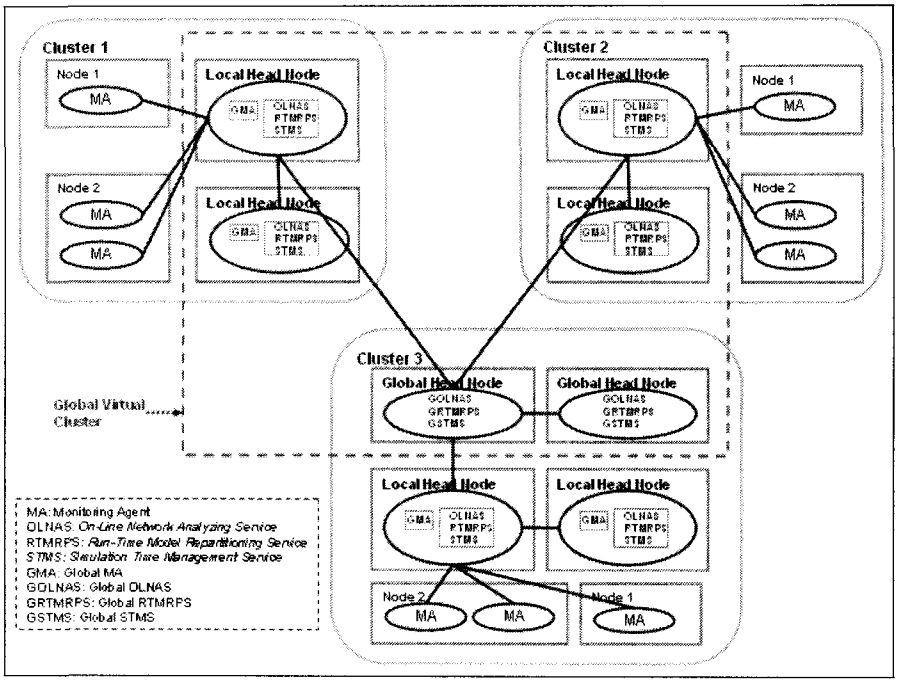

Figure 4.1: **High Level System Architecture**

4.2 Structure of the Dynamic Load Balancing Algorithm

Our load balancing algorithm assumes that recent "past behavior" reflects the near future one. Therefore, the monitoring agents introduced will monitor the simulation entities current activity within a certain interval. Based on the analysis of the observed data, a simulation reconfiguration will be decided. The on-line network analyzing service (OLNAS) gathers the monitoring agents observed data, analyzes the intra-cluster computing and communication workload distribution among the simulation entities. The run-time model repartitioning service (RTMRPS) is the key to run-time simulation reconfiguration.

It handles the repartitioning of the simulation entities within the cluster. The OLNAS triggers the load balancing by sending a request to all the monitoring agents to collect their observed data upon the elapse of a waiting time interval Δ. The monitoring agents will send their monitored data to the OLNAS, reset their monitoring mechanism and stop their observation policies. Upon receiving the data from the monitoring agents the OLNAS will perform a thorough analysis and forward its analysis results to the RTMRPS. The RTMRPS does the migration decision based on its analysis. Before starting the migration the RTMRPS will communicate with the STMS to pause the simulation during the migration. When the migration is done the RTMRPS will notify the STMS to resume the simulation and the OLNAS to trigger the monitoring agents back to work. Our load balancing algorithm is limited to one migration per session due to the migration overhead on the whole simulation and we aim to find the migration that will give us the best possible result. Figure 4.2 illustrates the load balancing sequence diagram. In the following sections, we will describe in detail the three major components of our load balancing algorithm; the monitoring agent, on-line network analyzing service and run-time model repartitioning service. Following a similar methodology in [13] of computation and communication load balancing, every component has two phases that will be described; a computation phase and a communication phase. Although these phases are described separately they are interdependent and the final outcome of our load balancing algorithm takes both phases into consideration.

4.2.1 Monitoring Agent

4.2.1.1 Computation Monitoring Mechanism

The computation workload monitoring mechanism is based on the simulation entity's processing activity. We need to mention here that our simulation entities are implemented as multi-threaded components. The processing activity can be approximated by measuring the CPU usage of the simulation entity. To obtain the time spent on the

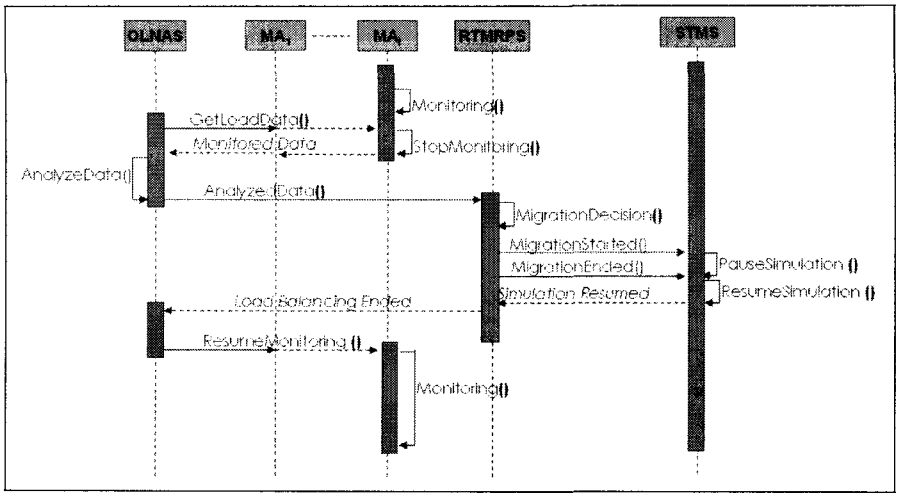

Figure 4.2: **Load Balancing Sequence Diagram**

CPU by a specific simulation entity thread, we use ThreadMXBean interface within Java's java.lang.management package. Therefore, the thread identifier and CPU processing time can be collected. We use the method getThreadCpuTime() to obtain a good approximation of the observed CPU computation processing time (Φ). To avoid too abrupt variations we introduce a smoothing mechanism that balances the previous simulation entity load value with the current simulation load value. The smoothed simulation entity load value ($Load_{SE}$) is calculated in the following way:

$$Load_{SE_i} = (1 - \alpha).Load_{SE_{i-1}} + \alpha.\Phi_i \tag{4.1}$$

Where:

i is the current load balancing phase

$0.1 < \alpha < 1$

$\alpha = \frac{Load_i}{Load_{i-1}}$ if $Load_i < Load_{i-1}$

$\alpha = \frac{Load_{i-1}}{Load_i}$ if $Load_i > Load_{i-1}$

The greater α is the smaller the value between the past and current load is. Moreover, a

node workload ($Load_N$) is defined as the sum of all its simulation entities workload:

$$Load_{N_i} = \sum_{SE \in N_i} Load_{SE} \tag{4.2}$$

4.2.1.2 Communication Monitoring Mechanism

The communication workload monitoring mechanism is based on introducing communication log tables. Every message initiated by the simulation entity updates the communication log table. The table entries store the message length in bytes as well as message's destination. Analyzing the table gives a good approximation of the simulation entity's communication behavior. Basically, it clearly indicates whether the simulation entity communicates more with foreign hosts rather than with its local host. A simulation entity having more foreign communication is considered to have low host binding while a simulation entity with more local communication is considered to have high host binding.

4.2.1.3 Monitoring Agent Output

Upon receiving the request from the OLNAS for the observed data, the monitoring agent first calculates the workload of the simulation entity as defined in Equation 4.1. Then it responds to the OLNAS by sending information about the simulation entity (simulation entity id, host name and the size of objects controlled by the simulation entity), the communication log table and the calculated workload.

4.2.2 On-line Network Analyzing Service

4.2.2.1 Computation Analysis

The OLNAS computation analysis aims at:

- Classifying the nodes as overloaded, balanced and under-loaded

- Identifying highest workload generating simulation entities within each overloaded node

For the node classification, it calculates every node load as defined in Equation 4.2, the average μ of all the nodes load and their respective standard deviation δ. According to Chebyshev's inequality [41], for nearly all random distributions, about 68 % of the values are within 1 standard deviation of the mean. Consequently, the node classification is based on the following:

- Overloaded nodes are those whose load is greater $\mu + \delta$
- Under-loaded nodes are nodes whose load is less than μ - δ
- Balanced nodes has a load equal or in between μ - δ and $\mu + \delta$

Then, the OLNAS identifies the simulation entities with highest workload within the set of overloaded nodes.

4.2.2.2 Communication Analysis

The OLNAS communication analysis aims at:

- Identifying simulation entities with high foreign communication

The communication analysis is carried out on the communication log table by calculating the difference between the foreign calls and local calls. Simulation entities having high foreign communication are said to have low host binding while those with high local communication are said to have high host binding. The analysis aims at locating simulation entities that have low host binding and will be possible migration candidates. Nevertheless, a simulation entity may be experiencing a slightly higher foreign communication than local one and its migration will have a negative impact on the simulation performance. We, therefore, consider simulation entities with low host binding those who have foreign calls

double or more than local calls.It is worth mentioning that our selection for the *double* parameter has no major impact on our load balancing algorithm behavior and performance since what matters is to have significantly more foreign calls than local calls. Eventually, simulation entities that have foreign calls less than double their local calls will be considered as high host binding.

4.2.2.3 On-line Network Analyzing Service Output

At the end of the analysis, the OLNAS will have 4 possible outcomes; two possible migration candidates a computation and a communication one, one possible computation migration candidates, one possible communication migration candidates or none. If no possible migration candidates are found then our load balancing algorithm will stop here. If a possible set of migration candidates is found then it will be sent to the RTMRPS with all the analyzed data.

4.2.3 Run-Time Model Repartitioning Service

4.2.3.1 Computation Migration Decision

The computation migration decision will be initiated if a possible set of computation migration candidates has been found by the OLNAS. It considers the identified possible computation migration candidates by the OLNAS which are basically the high workload generating simulation entities within the overloaded nodes. As we have mentioned earlier our purpose is to get down to one migration which will basically yield the highest gain. Consequently, we will run a first set of conditions on the possible migration candidates and if more than one possible migration candidate is left we will run another set of conditions. The first set of conditions is the following:

- The simulation entity has more foreign communication than local
- The simulation entity migration will not cause the overloading of the destination node

The latter condition can be easily calculated by adding the simulation entity workload to the destination node's workload and by checking whether the destination node becomes overloaded. If more than one possible computation migration candidate satisfies the first set of conditions, we have to further reduce our selection by considering a second set of conditions:

- The simulation entity has high communication with one of the under-loaded nodes
- The simulation entity migration will induce the least migration latency(predicted by the size of objects it has)

Consequently, the set of possible computation migration candidates that passes the two set of conditions will be referred to as the computation migration candidates set.

4.2.3.2 Communication Migration Decision

The communication migration decision will be initiated if a possible set of communication migration candidates has been found by the OLNAS. It considers the identified possible communication migration candidates by the OLNAS which are basically the simulation entities with high foreign communication. With a similar methodology and argument for the computation migration decision we run two set of conditions on the possible communication migration candidates. The first set of conditions is the following:

- The simulation entity migration will not cause the overloading of the destination node
- The simulation entity migration will keep its host node within a balanced state

If more than one possible migration candidate meets the first set of conditions, we try to further reduce our selection by running a second condition:

- The simulation entity migration will induce the least migration latency

Consequently, the set of possible communication migration candidates that passes the two set of conditions will be referred to as the communication migration candidates set.

4.2.3.3 Run-Time Model Repartitioning Service Output

By running the computation and communication migration decisions the RTMRPS is left with a computation migration candidate set and a communication migration candidate set. There are four possible combination cardinalities for these two sets which we need to carefully consider. Based on each combination a different execution path and set of procedures will be executed by the RTMRPS. The four possible combination cardinalities are:

1. The cardinality of the computation migration candidate set and the communication migration candidate set are both greater than or equal to 1

The set with the higher priority will be selected and if it has more than one migration candidate a random simulation entity will be picked. The priority scheme will be explained in detail in Section 4.3.

2. The cardinality of the computation migration candidate set is greater than or equal to 1 while the cardinality of the communication migration candidate set is 0

A simulation entity will be randomly picked from the computation migration candidate set.

3. The cardinality of the communication migration candidate set is greater than or equal to 1 while the cardinality of the computation migration candidate set is 0

A simulation entity will be randomly picked from the communication migration candidate set.

4. The cardinality of the computation migration candidate set and the communication migration candidate set are both equal to 0

Reaching such a case means that we have a load imbalance locally however any local migration will not resolve the imbalance. Therefore, in such a case we trigger our global

load balancing to help us resolve the local imbalances.

Finally, when the RTMRPS migration decision is settled on a specific simulation entity, the migration process can be initiated. Before starting the migration, the RTMRPS communicates with the STMS to pause the simulation during the migration. When the migration is done the RTMRPS notifies the STMS to resume the simulation and the OLNAS to trigger the monitoring agents back to work. The basic concepts of the local intra-cluster load balancing algorithm are outlined in Algorithm 1.

4.3 Dynamic Load Balancing Algorithm Adaptation

The worst-case scenarios for our load balancing algorithm are with a well-balanced simulation or with a simulation that is experiencing only one kind of imbalance either a computation or a communication one. Running our load balancing algorithm with such scenarios will induce an overhead that is not needed and will potentially degrade the simulation performance. To resolve this issue we add to our design two adaptation mechanisms that can detect such scenarios and adjust accordingly to our load balancing algorithm procedure and monitoring policies. Our first adaptation mechanism is for the OLNAS waiting time interval (Δ). Δ is initially set to a small value. If a migration decision is reached it is halved and if none is reached than it is doubled.It is worth mentioning that our selection for the *double* and *half* parameters has no major impact on our load balancing algorithm behavior and performance since what matters is to have a significant decrease of the waiting time interval when there are frequent load imbalances and a significant increase of the waiting time interval when there are no load imbalances. As a result, with a well-balanced simulation our load balancing algorithm waiting time interval will keep on increasing and eventually reducing its unnecessary overhead. Our second adaptation mechanism considers prioritizing the computation and communication imbalances. The RTMRPS besides doing the migration decision ensures an adaptive load differentiation mechanism based on the simulation dynamic behavior. Priorities are introduced to the different computation

Algorithm 1 Local Intra-Cluster Load Balancing Algorithm

Δ: OLNAS waiting time interval
SE_{ijk}: i^{th} simulation entity group of j^{th} node of k^{th} cluster
MA_{ijk}: i^{th} monitoring agent of j^{th} node of k^{th} cluster
N_{ij}: i^{th} node of j^{th} cluster
$Load(SE_{ijk})$: load of i^{th} simulation entity group of j^{th} node of k^{th} cluster
$Load(N_{ij})$: load of i^{th} node of j^{th} cluster
μ: average of simulation entity loads
σ: standard deviation of simulation entity loads
ON: set of overloaded nodes
BN: set of balanced nodes
UN: set of under-loaded nodes
$PPMC$: set of possible computation migration candidates of simulation entities
$PCMC$: set of possible communication migration candidates of simulation entities
PMC: set of computation migration candidates of simulation entities
CMC: set of communication migration candidates of simulation entities
BEGIN
ON $\leftarrow \emptyset$, BN $\leftarrow \emptyset$, UN $\leftarrow \emptyset$, PPMC $\leftarrow \emptyset$, PCMC $\leftarrow \emptyset$, PMC $\leftarrow \emptyset$, CMC $\leftarrow \emptyset$
wait (Δ)
for *all* MA_{ijk} **do**
  Send monitored data to OLNAS
end for
OLNAS data analysis:
Calculate the load of all nodes, Calculate μ and σ
for *all* N_{ij} **do**
  if $N_{ij} > \mu + \sigma$ **then**
    ON $\leftarrow$ ON $\cup \{N_{ij}\}$
  else if $N_{ij} < \mu - \sigma$ **then**
    UN $\leftarrow$ UN $\cup \{N_{ij}\}$
  else
    BN $\leftarrow$ BN $\cup \{N_{ij}\}$
  end if
end for
for *every node in ON* **do**
  Identify SE_{ijk} with highest workload
  PPMC $\leftarrow$ PPMC $\cup \{SE_{ijk}\}$
end for
for *all* SE_{ijk} **do**
  Identify SE_{ijk} with highest foreign communication
  PCMC $\leftarrow$ PCMC $\cup \{SE_{ijk}\}$
end for
Send analyzed data to RTMRPS
RTMRPS migration decision:
for *every simulation entity in PPMC* **do**
  If migration of SE_{ijk} is possible,
  PMC $\leftarrow$ PMC $\cup \{SE_{ijk}\}$
end for
for *every simulation entity in PCMC* **do**
  If migration of SE_{ijk} is possible,
  CMC $\leftarrow$ CMC $\cup \{SE_{ijk}\}$
end for
if $card(PMC) \geq 1$ AND $card(CMC) \geq 1$ **then**
  Prioritize
end if
if $card(PMC) \geq 1$ AND $card(CMC) = 0$ **then**
  pick randomly a simulation entity from PMC
end if
if $card(PMC) = 0$ AND $card(CMC) \geq 1$ **then**
  pick randomly a simulation entity from CMC
end if
if $card(PMC) = 0$ AND $card(CMC) = 0$ **then**
  trigger global inter-cluster load balancing
end if
Initiate migration
END

and communication workload and are dynamically modified to meet the requirements of our simulation. The argument here is that a simulation at run-time can experience either computation or communication imbalances for a long period of time. Consequently, during this period, it is preferable to give either one a higher priority over the other. Furthermore, switching off the policies to detect one of the imbalances within this period can reduce unnecessary overhead. The mechanism is implemented by defining two counters. In our implementation, the computation load balancing phase is given a higher priority at the beginning. If the RTMRPS migration decision is settled on a computation migration candidate then its respective counter will be incremented while the other decremented and vice versa. The trailing phase policies are switched off for the next load balancing phase. For instance, assuming that the communication phase is switched off, the monitoring agents will stop updating their communication log tables after being notified by the OLNAS. Furthermore, the OLNAS communication data analysis and the communication RTMRPS migration decision will be skipped. It is worth mentioning that, during the switching off for the communication phase, the RTMRPS computation migration decision will use the last saved communication behavior of the simulation entities. A similar approach is applied when the computation phase is switched off. The advantages of our adaptive scheme will be evaluated in Chapter 5.

4.4 Migration Process

JXTA [23]-[26] provides a virtual network, which can be used for designing highly dynamic and self-aware heterogeneous virtual networking infrastructure. Such a virtual distributed network aims to support run-time evolution of participating computing entities through publish/subscribe based peer discovery protocol, as well as pipe based message passing mechanisms.

Consequently, in our simulation, simulation entities can join a running simulation and the simulation structure evolves dynamically through dynamic simulation entities regis-

tration and dynamic pipes creation. The migration is initiated by the RTMRPS. The RTMRPS signals the STMS to pause the simulation while migration is under progress. When the simulation is paused, all the simulation entities save their current state. A copy of the simulation entity is created with the same simulation state and it is set to dynamically lookup the STMS. When it discovers the STMS it requests for registration. After the registration is granted by the STMS, the simulation entity creates pipes and can join the simulation. Moreover, the RTMRPS releases all the resources held by the original simulation entity. After the migration is done, the RTMRPS signals the STMS to resume the simulation.

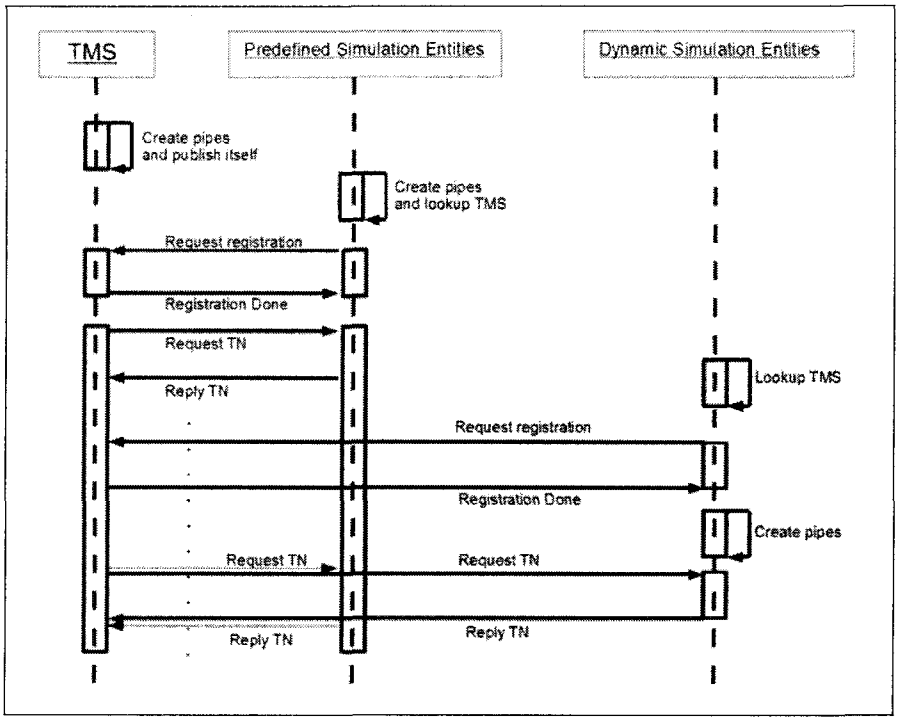

Figure 4.3: **Dynamic Simulation Entity Creation and Registration**

Chapter 5

Performance Evaluation of our Proposed Dynamic Load Balancing Algorithm

After assembling all the different components of the design together, a well-seen improvement became comprehensible. We implemented all the sets of rules and procedures already mentioned, and the results were as promising as we predicted. From what was mentioned earlier, our goal is to have a dynamic load balancing to decrease our simulation execution time. However, an overhead has been noticed. The overhead that was noticed and is supported by all of our experiments can be easily justified as the price of a dynamic load balancing scheme. We can never have dynamic load balancing without incurring a little overhead, since this overhead increases our overall simulation performance. From the results of our experiments, the simulation execution time of a simulation with high computation and communication imbalances has been enhanced significantly. However, an overhead has been noticed in terms of the time of the simulation execution time. The monitoring agents, data analysis, migration decision and the actual migration are the sources of the overhead. The experiments ran on a cluster of nodes at the PARADISE Research Laboratory in the University of Ottawa. The nodes' product type is eserver xSeries 336. Each node has two Intel Xeon processors at 3.4 GHz each, with 2 GB DIMM DDR synchronous RAM. Various experiments were performed using 8 and 32 nodes: One node is used to run the local head which controls the simulation and runs the dynamic load balancing scheme while the others are used by the remaining simulation entities. Interaction between the

simulation entities is done through the inter-process communication channels available.

The initial static partitioning assigns the local head to one node and distributes the simulation entities evenly on the remaining nodes. As a future work a more intelligent initial static partitioning will be the responsibility of the model static partitioning and deployment service [1]. In order to provide trustworthy results, each plotted point in our graphs represents the average of 33 runs. Thus, the confidence intervals were calculated using the z-distribution at a confidence level of 95%. Our experiments ran with a centralized conservative time management [34]. We simulated a tank battle model. We had a number of tanks running in the battle field. The tanks are divided into several teams. A team of tanks can fire other tanks in different teams. A tank will fire to its enemy tanks when they are moving into its shooting range. The simulation will end when the simulation loops are over or when there is only one team left. Our simulation entities has the responsibility of simulating a number of tanks moving around in the battle field. The range of tanks for each of the simulation entities was from 2 to 10. Our simulation entity will simulate the tanks in a same team and a team can consist of several simulation entities. For every simulation loop the simulation entity requested time advance from STMS, received interactions from other simulation entities, computed its simulated tanks locations and sent interactions to other simulation entities. The simulation ran for 200 simulation loops. The computation load of our simulation is generated by the computation of the simulated tanks locations. We implanted different hills with different steepness in the battle space. The computation differs whether the tank's location is on a flat surface is on a hill and how steep is the hill. As for the communication load, it is generated by the simulation entity location updates and the tank's fighting.

Two general experiment setups for five different test case groups were devised and we will refer to them as general experiment setup 1 and 2. Both general experiment setups use the experimental configurations and environment already presented. However, the general experiment setup 1 has 35 simulation entities running on 8 nodes. The OLNAS waiting time interval Δ was initially set to 0.5 seconds. The general experiment setup 2 has a

range of simulation entities from 60 to 200 running on 32 nodes. The OLNAS waiting time interval Δ was initially set to 5 seconds.

To evaluate our approach, experiments were clustered in five test-case groups. The five test-case groups use the general experimental setups outlined here however some of them may have some modifications. The first group highlighted the benefits of dynamic load balancing on the simulation performance. The second group studied the effect of balancing both computation and communication for different simulation scenarios. In the third test-case group, we evaluated our load balancing algorithm overhead. The fourth test case group studied our load balancing algorithm adaptation mechanism and the fifth compared our load balancing algorithm to the load balancing algorithm presented in [13]. The first three test case groups use the general experiment setup 1 while the last two use general experiment setup 2.

5.1 Dynamic Load Balancing Benefits

This test case group shows how dynamic load balancing can benefit our simulation performance. The percentage of computation and communication loads are increased for different experimental runs. The computation percentage load increase is accomplished by increasing the hills steepness while the communication percentage load increase is accomplished by increasing the tanks shooting area range. The results are shown in Figure 5.1. For less than a 30% load increase our dynamic load balancing exhibited more overhead up to 9.3% than actual performance gain. However, as we increased the computation and communication generated load by 100% we achieved a 24.8% decrease in the simulation execution time.

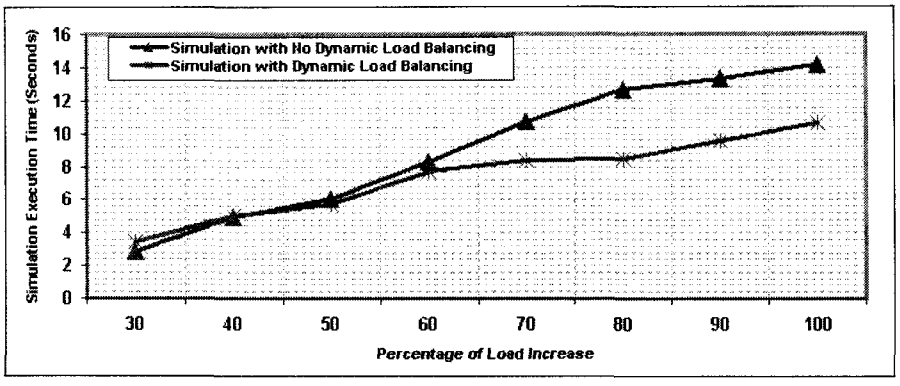

Figure 5.1: **Comparison of a Simulation with and without Dynamic Load Balancing for an Increasing Percentage of Load**

5.2 Effect of Computation and Communication Load Balancing

In a second test-case group of experiments, we studied the effect of balancing both computation and communication for different simulations. A lot of existing algorithms takes only either computation or communication imbalances into consideration but not both. However, our algorithm functions best when both imbalances are present which the case in most simulations is. The simulation ran with and without dynamic load balancing for three different scenarios. Our load balancing adaptation mechanism was only limited to the waiting time interval. The percentage of computation and communication loads are increased using the same methodology presented in the first experiment. For the first scenario, the simulation ran with computation events only, in the second scenario with communication events only and in the third one with both computation and communication events. To generate only computation events we disabled the tanks' firing ability while to generate only communication events we set an all flat battle space. Figure 5.2, Figure 5.3 and Figure 5.4 illustrates the results of the three scenarios respectively. The dynamic load balancing overhead outweighs its gain for a small percentage of load since not much

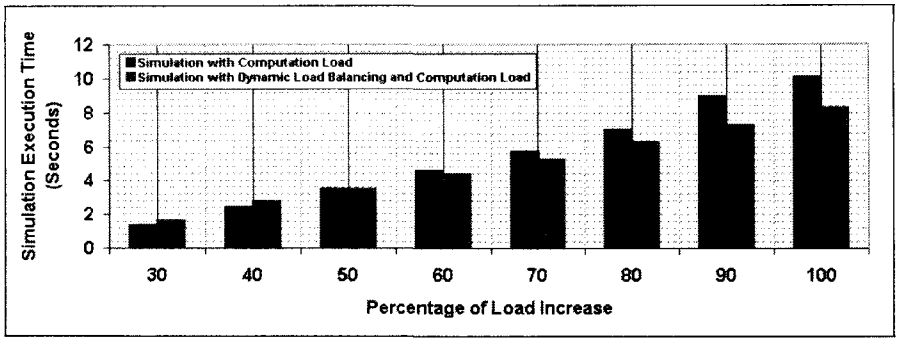

Figure 5.2: **Simulation Comparison with and without Dynamic Load Balancing for Computation Load**

computation and communication load is generated. For instance, for a 30% load increase the overhead for the first scenario is 14%, the second scenario is 12.3% while the third is 9.3%. As we can observe, our algorithm produces more overhead for a simulation with only one kind of imbalance. Nevertheless, as the percentage of load increases to 100 the gain for the first scenario becomes 17.9%, the second scenario 15.2% while the third becomes 24.8%. The higher gain in the computation part as compared to the communication one has no significance here and is caused only by initial configurations. The preceding results verify our argument that our proposed dynamic load balancing performs best when both sources of imbalances are present which the case in most simulations is. Furthermore, our algorithm scales well for an increasing percentage of computation and communication load.

5.3 Evaluation of the Dynamic Load Balancing Algorithm Overhead

We also investigated the overhead sources of our load balancing algorithm: The monitoring agents, data analysis, migration decision and the actual migration. We ran three different scenarios of our simulation in order to isolate the overhead generated from the monitoring policies, data analysis and migration decision from that generated by the mi-

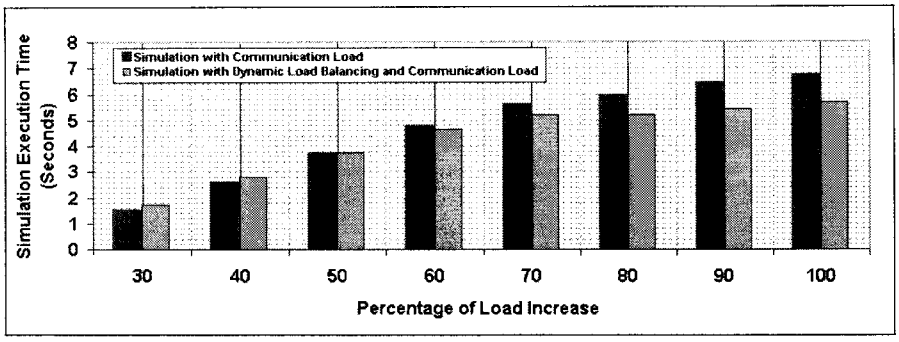

Figure 5.3: **Simulation Comparison with and without Dynamic Load Balancing for Communication Load**

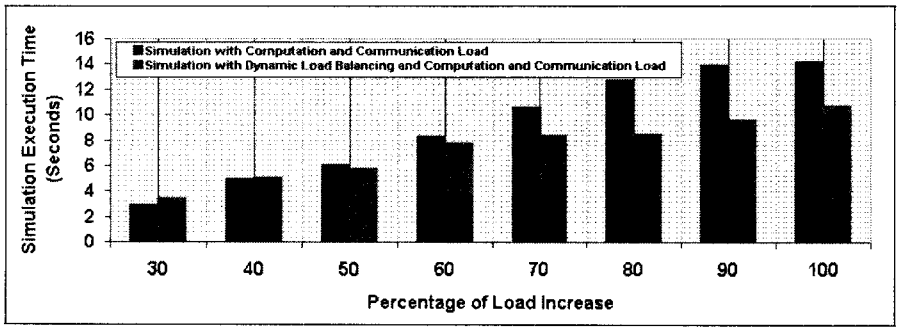

Figure 5.4: **Simulation Comparison with and without Dynamic Load Balancing for Computation and Communication Load**

gration. In the first scenario, we ran the simulation with no dynamic load balancing. In the second, we added the dynamic load balancing algorithm but disabled the migration procedure, while in the third scenario; the migration was enabled and was limited to one migration. Table 5.1 summarizes our results; the overhead from the migration (3.6%) outweighs the overhead from the monitoring policies (0.3%). This validates our preceding argument of the one migration per session restriction.

5.4 Evaluation of our Load Balancing Algorithm Adaptation Mechanism

In this fourth test-case group we evaluate our algorithm adaptation mechanism and specifically our prioritization scheme that aims at reducing unnecessary overhead. An increasing number of simulation entities are used and are set to generate only one type of load imbalance which is a computation load in our case. We ran our simulation with the dynamic load balancing once with the adaptation mechanism functioning and another without it. As shown in figure 5.5, a decrease in the simulation run-time was noticed when using the adaptation mechanism. This is due to the fact that our algorithm considers both computation and communication load as sources of imbalances and by integrating our adaptive prioritization scheme we were able to get rid of the unnecessary overhead generated by the communication monitoring, analysis and migration decision. Our performance gain reached up to 9.8% for 200 simulation entities.

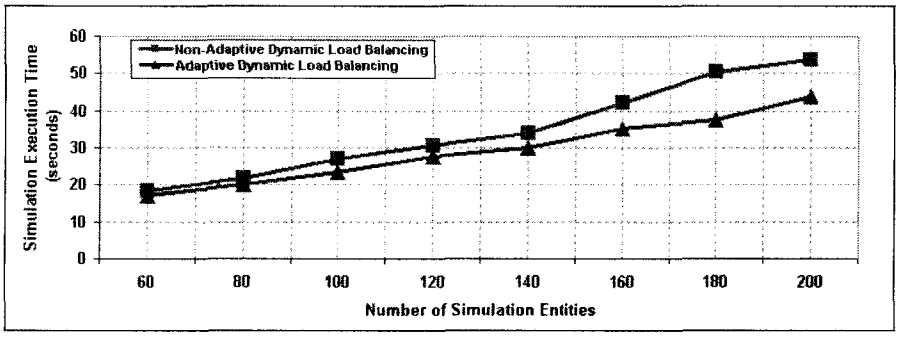

Figure 5.5: **Comparison of a Simulation with and without Adaptive Dynamic Load Balancing**

Table 5.1: Detailed Overhead of our Algorithm

	Scenario 1	Scenario 2	Scenario 3
Simulation Execution Time (Seconds)	2.819	2.828	2.933
Total Overhead (%)			3.9
Monitoring, Data Analysis, Migration Decision Overhead (%)			0.3
Overhead per Simulation Entity (%)			0.01
Actual Migration Overhead (%)			3.6

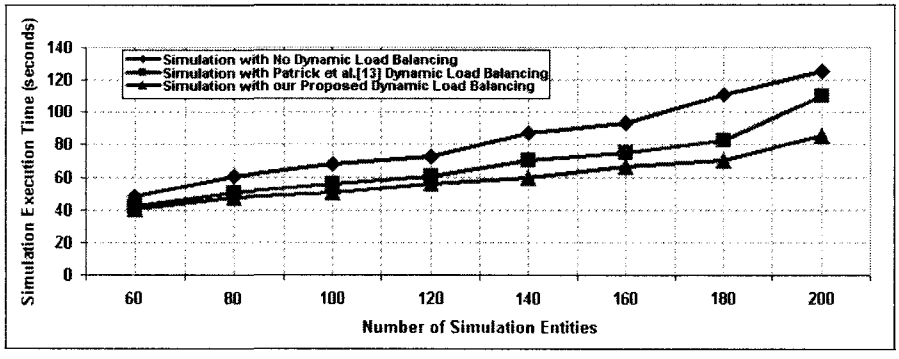

Figure 5.6: **Comparison of our Proposed Load Balancing Algorithm to the Load Balancing Algorithm in [13]**

5.5 Comparison of Load Balancing Algorithms

Finally, in this test-case group we compared our load balancing algorithm to the dynamic load balancing algorithm proposed in [13] and to a simulation running with no dynamic load balancing. As shown in figure 5.6, our algorithm had a significant improvement over the algorithm presented in [13] and over the simulation with no dynamic load balancing. The improvement noticed for 60 simulation entities was around 4.5% in comparison to [13] and around 16.6% in comparison to the simulation with no dynamic load balancing. Our algorithm scaled well and the improvement reached up to 21.9% in comparison to [13] and around 31.4% in comparison to the simulation with no dynamic load balancing for 200 simulation entities. The preceding results demonstrates our algo-

rithm scalability with an increasing number of simulation entities. Moreover, most of our improvement is due to our low migration latency and our restriction to one migration per load balancing phase as opposed to many migrations and movements per a load balancing phase suggested in [13].

The data and observations from our simulation experiments indicate that the benefits from the dynamic load balancing are significant when well configured. These experiments justified what has been claimed in the previous sections about the trustworthiness of our proposed dynamic load balancing scheme.

Chapter 6

Dynamic Load Balancing Algorithm Inter-Cluster Extension

The global inter-cluster load balancing scheme is an extension of the local intra-cluster load balancing scheme. However, few modifications need to be introduced in order to handle new challenges such as heterogeneity and scalability. The inter-cluster load balancing algorithm can be triggered in two different cases, either periodically or upon the failure of the intra-cluster balancing algorithm. The waiting time interval (Π) of the global inter-cluster algorithm will be greater than the largest participating cluster waiting time interval (Δ). This will ensure that enough effort has been made to balance the clusters locally before any global costly intervention. Moreover, in the intra-cluster load balancing scheme, the use of the average and standard deviation to identify the overloaded, balanced and under-loaded nodes is well justified since the nodes within one cluster are homogeneous. However, in a grid environment, most probably we have heterogeneous clusters. Therefore our algorithm needs to be slightly modified to meet this requirement. A benchmark test is necessary to identify the capacity of every cluster in terms of its nodes workload defined in Equation 4.2.

$$Cap_{C_i} = \sum_{N \in C_i} Load_N \tag{6.1}$$

The benchmark test can be executed on one node and the capacity can be deduced by multiplying the node's capacity to the number of nodes in the cluster. The capacity of the node will be calculated by executing a sequence of interactions and computations while

monitoring the node load using well known commands for different platforms (such as top command for Linux).

The cluster workload is calculated by the GOLNAS by summing the simulation entity group workloads provided by their global monitoring agents. The clusters classification is done by first normalizing the current cluster load by dividing it by its respective capacity. Consequently, the average, standard deviation and classification will be conducted on the normalized loads. Our global load balancing scheme will incur a large overhead compared to the local one. An inter-cluster simulation entity group migration can lead to significant network latency as opposed to a single intra-cluster simulation entity migration. Also, a benchmark test execution is needed to identify each cluster's capacity. Consequently, we configure our algorithm to provide the user the flexibility of switching off the global load balancing scheme. The basic concepts of the global inter-cluster algorithm are outlined in Algorithm 2. In the following chapter, we formalize and validate our inter-cluster dynamic load balancing using DEVS and realize it in DEVSJAVA.

Algorithm 2 Global Inter-Cluster Load Balancing Algorithm

Π: GOLNAS Triggering interval
GSE_{ij}: i^{th} simulation entity group of j^{th} cluster
GMA_{ij}: i^{th} global monitoring agent of j^{th} cluster
C_i: i^{th} cluster
$Nor(C_i)$: Normalized load of i^{th} cluster
μ: average of normalized cluster loads
σ: standard deviation of normalized cluster loads
OC: set of overloaded clusters
BC: set of balanced clusters
UC: set of under-loaded clusters
$GPPMC$: set of possible computation migration candidates of simulation entity groups
$GPCMC$: set of possible communication migration candidates of simulation entity groups
$GPMC$: set of computation migration candidates of simulation entity groups
$GCMC$: set of communication migration candidates of simulation entity groups
BEGIN
OC $\leftarrow \emptyset$, BC $\leftarrow \emptyset$, UC $\leftarrow \emptyset$, GPPMC $\leftarrow \emptyset$, GPCMC $\leftarrow \emptyset$, GPMC $\leftarrow \emptyset$, GCMC $\leftarrow \emptyset$
wait (Π) or upon intra-cluster load balancing failure
for *all* GMA_{ij} **do**
  Send monitored data to GOLNAS
end for
GOLNAS data analysis:
Calculate current load of all clusters, Normalize the loads by dividing
by the cluster capacity, Calculate μ and σ
for *all* C_i **do**
  if $Nor(C_i) > \mu + \sigma$ **then**
    OC $\leftarrow$ OC $\cup \{C_i\}$
  else if $Nor(C_i) < \mu - \sigma$ **then**
    UC $\leftarrow$ UC $\cup \{C_i\}$
  else
    BC $\leftarrow$ BC $\cup \{C_i\}$
  end if
end for
for *every cluster in OC* **do**
  Identify GSE_{ij} with highest workload
  GPPMC $\leftarrow$ GPPMC $\cup \{GSE_{ij}\}$
end for
for *all* GSE_{ij} **do**
  Identify GSE_{ij} with high foreign communication
  GPCMC $\leftarrow$ GPCMC $\cup \{GSE_{ij}\}$
end for
GRTMRPS migration decision:
for *every simulation entity group in GPPMC* **do**
  If migration of GSE_{ij} is possible,
  GPMC $\leftarrow$ GPMC $\cup \{GSE_{ij}\}$
end for
for *every simulation entity group in GPCMC* **do**
  If migration of GSE_{ij} is possible,
  GCMC $\leftarrow$ GCMC $\cup \{GSE_{ij}\}$
end for
if $card(GPMC) \geq 1$ *AND* $card(GCMC) \geq 1$ **then**
  Prioritize
end if
if $card(GPMC) \geq 1$ *AND* $card(GCMC) = 0$ **then**
  pick randomly a simulation entity group from GPMC
end if
if $card(GPMC) = 0$ *AND* $card(GCMC) \geq 1$ **then**
  pick randomly a simulation entity group from GCMC
end if
if $card(GPMC) = 0$ *AND* $card(GCMC) = 0$ **then**
  return
end if
Initiate migration
END

Chapter 7

Dynamic Load Balancing Algorithm Validation and Verification using DEVS

In this chapter, we we discuss the Discrete Event System Specification (DEVS), with the conventions surrounding this modeling and simulation language. Then, we validate our inter-cluster dynamic load balancing algorithm using DEVS and realize it in DEVSJAVA.

7.1 Discrete Event System Specification Overview

The Discrete Event System Specification (DEVS) is a framework for modeling and simulation. It provides the means of specifying a mathematical object called a system. Basically, a system has a time base, inputs, states, outputs, and functions for determining next states and outputs given current states and inputs. Discrete event systems represent certain constellations of such parameters just as continuous systems do. For example, the inputs in discrete event systems occur at arbitrarily spaced moments, while those in continuous systems are piecewise continuous functions of time. The insight provided by the DEVS formalism is in the simple way that it characterizes how discrete event simulation languages specify discrete event system parameters. Having this abstraction, it is possible to design new simulation languages with sound semantics that are easier to understand. Indeed, the DEVSJAVA environment we used is an implementation of the DEVS formalism in Java, which enables the modeler to specify models directly in its terms.

When building a DEVS model of any system, three basic components of that model should be present to fully describe the functional aspect of the DEVS formalism in that system. They all work together under the same constitution of DEVS in order to complete the results overseen in the correct manner. These components generally are the Model, Simulator and the Experimental frame. None of them can be seen as a separate component since each one of them performs a specific job that the other will either complete or verify. The description of these components is as follows:

Model It is a set of instructions for receiving data (as an input) and consequently generating data (as an output) corresponding to that observable in the real system being modeled. In other words, it is the part that mocks the behavior of a real system in terms of input, output and intenal/external behaviors. The structure of the model is its set of instructions. The behavior of the model is the set of all possible data that can be generated by executing the model instructions as specified by the real-world component's behavior noticed at all stages. The more detailed this model gets, the more accurate the experimental results will get.

Simulator It exercises the model's instructions to actually generate its behavior and making sure that the real component's behavior is very close (if not exactly the same) to the model built over DEVS. Therefore, not only building the models is a critical issue, however, building a simulator will realize the issues represented as the "model".

Experimental frame This is the most critical part of the DEVS formalism as it actuates the model's behavior to get out some data for experimental purposes. It captures how the modelers objectives impact on model construction, experimentation and validation. For example, in DEVSJAVA (the version used in the modeling of our RT-RTI), the experimental frames are formulated as model objects in the same manner as the models of primary interest. In this way, model/experimental frame pairs can form coupled model objects with the same properties as other objects of this kind. This

uniformed treatment yields key benefits in terms of modularity and system entity structure representation as described in [33].

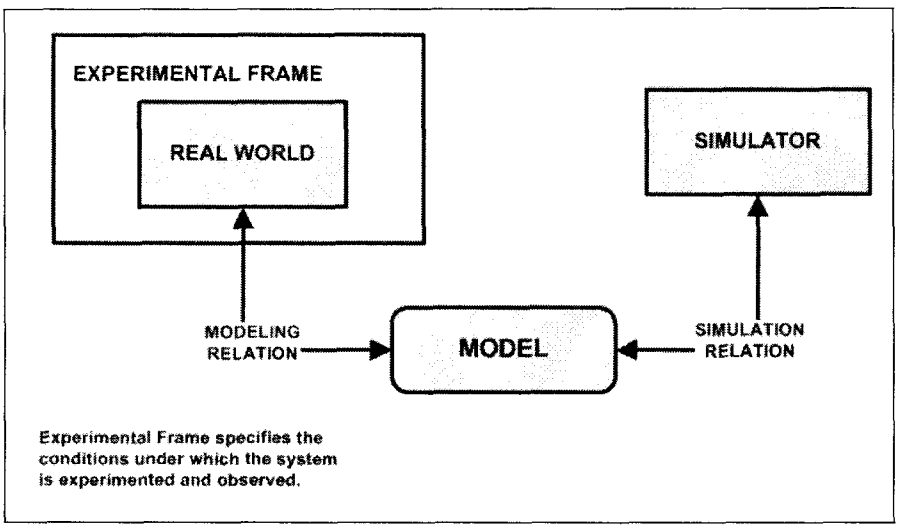

Figure 7.1: **Relations between DEVS Objects [33]**

As in Figure 7.1, the basic components of the DEVS formalism described in the previous list and their relatious are referenced. The relations between these basic objects are two relations and they are described hereunder:

Modeling Relation Linking real system and model, defines how well the model represents the system or entity being modeled. In other words, how close the model represents the real system. In general terms, a model can be considered valid if the data generated by the model agrees with the data produced by the real system in an experimental frame of interest and that is the parameter that defines the agreeability with the modeling relation detailed status.

Simulation Relation Linking model and simulator, represents how accurately the simulator is able to carry out the instructions of the model. The basic items of data

produced by a system or model are time segments. These time segments are mappings from intervals defined over a specified time base to values in the ranges of one or more variables. The variables can either be observed or measured.

7.2 DEVS Validation and Verification Methodology

DEVS formalism [27] is one of the most important components of the modeling and simulation theory. It provides a conceptual framework and an associated computational approach for solving methodological problems in Modeling and Simulation (M&S) communities. This computational approach is based on the mathematical theory of systems including the hierarchy of system specifications and specification morphisms. It manipulates a framework of elements to determine their logical relationships. As a formal system specification language, DEVS formalism and its associated modeling and simulation environment is a very useful tool for helping with complex system design and verification. DEVS is one of the best tools for low-level system specifications due to its ability to represent real world components in a very detailed manner through its discrete event favor, which is generally required for designing large-scale distributed systems. It is worth mentioning that the operation of the discrete event modeled systems is basically represented by a chronological set of events that change the system's state. Thus, for any discrete-event system, the event that occurs at a specific point in time changes the state of the system. While in continuous event systems, time is a continuous function and events occur at any point in time with no specific expectations for events timing, and here appears the difference clearly in the time function.

The DEVS formalism is as follows(i.e. an atomic DEVS):

$$\textbf{\textit{Atomic Model (AM)}} = \langle \boldsymbol{X},\ \boldsymbol{S},\ \boldsymbol{Y},\ \delta_{int},\ \delta_{ext},\ \lambda,\ \boldsymbol{ta} \rangle$$

Where:

X: set of external input events,

S: set of sequential states,

Y: set of outputs,

δ_{int}: S $\Longrightarrow$ S: internal transition function,

δ_{ext}: Q * $X^b \Longrightarrow$ S: external transition function

Where,

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$

X^b is a set of bags over elements in X,

λ: S $\Longrightarrow$ Y^b: output function,

ta: time advance function,

7.3 System Design using DEVS

As we know, the design of software or embedded systems is traditionally non-formalized, which often results in low performance and error-prone system architecture. It is worth mentioning that such non-formalized design approaches are still being used by modern researchers. Some researchers may argue that it is not necessary to use complex formal languages to describe their designs because they believe they have enough experience. However, questions such as: "How can you verify that you have reached an optimal system design?" and "Is your designed system scalable? If so, can we verify the efficiency of this design?" are definitely difficult to answer. From another point of view, it is actually impractical to use a non- formalized design for large-scale and dynamic systems since these systems generally require stage-by-stage realizations, verifications and a final integration. Design formalization becomes a necessity for designing large-scale and complex dynamic systems, especially real-time distributed systems, where time constraints need to be considered carefully at a very early stage. One of the key advantages of formalized design approach, such as using DEVS, is that critical system design problems can be identified at earlier stages before the costly overhead of realizing the implementation. Such an approach also supports design reusability by using a model repository. Another advantage is that the

design validation can be done before its realization. DEVS model-based design has been used by many researchers for solving complex design problems. For instance, Schulz and Rozenblit [35] have proposed a novel co-design approach that uses a formal specification language to describe embedded systems. In addition, Hu [36] has proposed a DEVS model-based methodology to be applied in designing dynamic distributed real-time systems with a particular focus on model continuity.

7.4 Proposed Design Validation and Verification using DEVS

In this section, we validate and verify our design using DEVS, and then realize the design in the DEVSJAVA environment. It is worth mentioning here that the design model components presented here are all reusable ones that can also be used to construct alternative design systems.

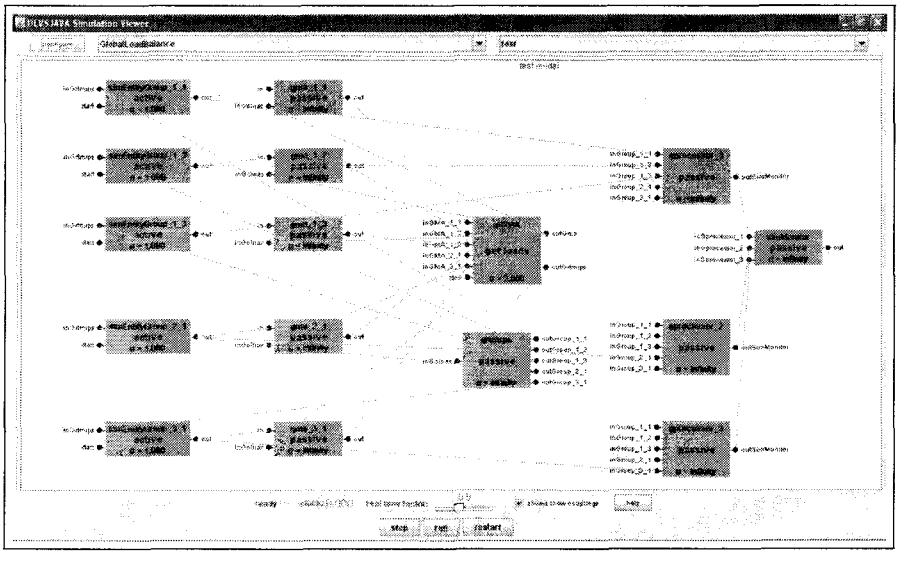

Figure 7.2: **Proposed Design Realized in DEVSJAVA**

As shown in Figure 7.2 , the design model is composed of the proposed dynamic load

balancing design. The model is composed of *simEntityGroup* (simulation entity group), *gma* (global monitoring agent), *golnas* (global online network analyzing service), *grtmrps* (global run time model repartitioning service), *gprocessor* (global processor), and *simMonitor* (simulation monitor). Each of these model components is described using DEVS formalism and then realized as DEVSJAVA atomic models. In Figure 7.2, the illustrated model is composed of 3 clusters. However, our implementation can handle any number of clusters with any number of simulation entity groups. *simEntityGroup_1_1*, *simEntityGroup_1_2* and *simEntityGroup_1_3* form cluster 1 while *simEntityGroup_2_1* forms cluster 2 and *simEntityGroup_3_1* forms cluster 3. Each *simEntityGroup* has its respective *gma* and each cluster has its respective *gprocessor*. The *golnas* and *grtmrps* are responsible for the dynamic load balancing while the *simMonitor* monitors the simulation and signals its end.

In the following paragraphs, we present each design model component in detail, and several key model components are presented using DEVS formalism as representatives.

1. *simEntityGroup* model: The *simEntityGroup* generates events within every simulation loop. It outputs to its respective *gprocessor* and *gma*. The generated events can be either computation or communication loads.

2. *gma* model: The *gma* model calculates and stores the load of the *simEntityGroup* generated events. It stores the load of the generated events and upon the *golnas* periodic request sends the saved load to the *golnas*.

3. *gprocessor* model: The *gprocessor* model is responsible for processing the events generated by all the *simEntityGroup* belonging to the same cluster. The computation events processing by the *gprocessor* represents the computation processing done by the different cluster processors while the communication events processing represents the latency generated by the message passing over the network. Upon processing all the events it signals the *simMonitor* that it has finished. The gprocessor has a predefined capacity for processing

the *simEntityGroup* generated events which represents the cluster's capacity.

4. *golnas* model: It is the key control unit for triggering the dynamic load balancing. The *golnas* sends out a request to each *gma* upon the elapse of its waiting time interval (Π) which is calculated as described in Algorithm 2. After getting the load generated by each *simEntityGroup*, it then classifies the clusters as overloaded, under-loaded and balanced and identifies within the overloaded cluster the highest computation load generating simulation entity group. Moreover, it analyzes the simulation entity group communication behavior whether it has high or low host binding. Finally, it sends to the *grtmrps* the simulation entity groups which are candidates for migration.

5. *grtmrps* model: The *grtmrps* model decides whether a simulation entity migration is possible. If a migration is possible it sends a message to the selected *simEntityGroup* in order to migrate to the specified cluster. This model has a configurable time delay for outputting its result to the selected *simEntityGroup* model. The time delay in here represents the actual migration latency which can be approximated by having the clusters bandwidth connections and the simulation entity groups size.

6. *simMonitor* model: The *simMonitor* model monitors the simulation and upon receiving messages from all the *gprocessor* of the clusters participating in the simulation signals the end of the simulation.

As we can see from the above descriptions of the model components, they have some common characteristics. We therefore select only two key models to describe in DEVS formalism as representatives.

The *gma* model is described with DEVS as

X={computation load, communication load, fetch load}
S={passive, active, outLoad}
Y={monitored load}

δ_{int}\{passive, active, outLoad\} = \{passive\}

δ_{ext}\{computation load,\{passive,0<e<infinite\}\} = \{active\}

δ_{ext}\{communication load,\{passive,0<e<infinite\}\} = \{active\}

δ_{ext}\{fetch load,\{passive,0<e<infinite\}\} = \{outLoad\}

λ\{outLoad\} = \{monitored load\}

ta(outLoad) = 0 ; ta(active) = 0 ; ta(passive) = infinite

The *golnas* model is described with DEVS as

X=\{monitored load\}

S=\{passive, fetchLoad, analyzeData\}

Y=\{fetch load, migration candidates\}

δ_{int}\{passive, fetchLoad, analyzeData\} = \{passive\}

δ_{ext}\{monitored load,\{passive,0<e<infinite\}\}=\{analyzeData\}

λ\{fetchLoad\} = \{fetch load\}

λ\{analyzeData\} = \{migration candidates\}

ta(fetchLoad) = 0 ; ta(analyzeData) = 0 ; ta(passive) = infinite

In the following section, we will present some precisely designed simulation experiments that aim to test and verify the advantages of the proposed dynamic load balancing design when compared with a design with no load balancing. Our particular focus is on looking at two key characteristics of our design; The importance of balancing both computation and communication loads, and the effect of run-time model repartitioning through dynamic migration on the performance of the simulation execution time.

7.4.1 Design Verification through Simulation Experimental Frames

In this section, we follow the experimental frame concepts commonly used in DEVS based modeling and simulation. We analyze our design using previously defined DEVS models, and we focus our experiments on the effects of dynamic migration and the simul-

Table 7.1: Settings for the DEVSJAVA Experiments

Simulation Loops	200
Waiting Time Interval (Δ) for Cluster 1	3
Waiting Time Interval (Δ) for Cluster 2	2
Waiting Time Interval (Δ) for Cluster 3	4
Processing Capacity of Cluster 1	0.3
Processing Capacity of Cluster 2	0.6
Processing Capacity of Cluster 3	0.2
Initial Generated Computation Events Processing Time Random Range	0.1 - 0.3
Initial Generated Local Communication Events Message Latency Random Range	0.1 - 0.2
Initial Generated Foreign Communication Events Message Latency Random Range	0.2 - 0.3
Estimated Migration Latency	5

taneous computation and communication load balancing. For the following simulation experiments, the design models are simulated in a DEVSJAVA environment. Our simulation experiments are designed to verify the key aspects of the proposed inter-cluster algorithm. Therefore, the following simulation experiments presented will focus on two of the most important characteristics of our new design: run time model repartitioning through dynamic migration and the effect of balancing both computation and communication loads.

It should be noted that, for the following simulation experiments, all the models are set to depict the same behavior as illustrated in Algorithm 2. Table 7.1 summarizes the parameter settings used for our experiments. Moreover, the linkages between the *simEntityGroup* and *gprocessor* can be altered dynamically according to the load balancing requirements, and such dynamic linkage changes are realized in DEVSJAVA by using a technique called variable structure. This technique has the ability to change the model system structure and couplings during simulation run-time.

7.4.2 Run-Time Model Repartitioning in the Modeled Design

In this simulation experiment, we study how dynamic migration improves the overall simulation performance. The simulation is set to run with an increasing percentage of load by increasing the computation and communication events processing time random range generated by the *simEntityGroup*. The dynamic migration is realized in DEVS models by a request from the *grtmrps* model to a *simEntityGroup* model. Consequently the coupling between the *simEntityGroup* model and the *gprocessor* model are changed accordingly.

In this experiment, we ran two different scenarios; In the first one, the simulation is set to run without dynamic load balancing while in the second with dynamic load balancing. As shown in Figure 7.3, the simulation running with dynamic load balancing and run-time model repartitioning through dynamic migration had a significant improvement on the simulation performance. The improvement on the simulation execution time varied between 14.5% with a 20% load increase up to 33.1% for a 100% load increase. This result verifies and validates the effectiveness and worthiness of our proposed inter-cluster algorithm. However, it is worth mentioning that the preceding results are based on our initial configurations and estimated migration latency. Therefore, on a real implementation the migration overhead should not count more than the predicted gain.

7.4.3 Effect of computation and communication load balancing in the Modeled system

In this simulation experiment, we use the same experimental setup and parameters used in the earlier experiment. However, we focus on studying another key characteristic of our algorithm which is the effect of computation and communication load balancing. A lot of existing algorithms takes only either computation or communication imbalances into consideration but not both. However, our algorithm functions best when both imbalances are present which is the case in most simulations. We simulated three different scenarios; A

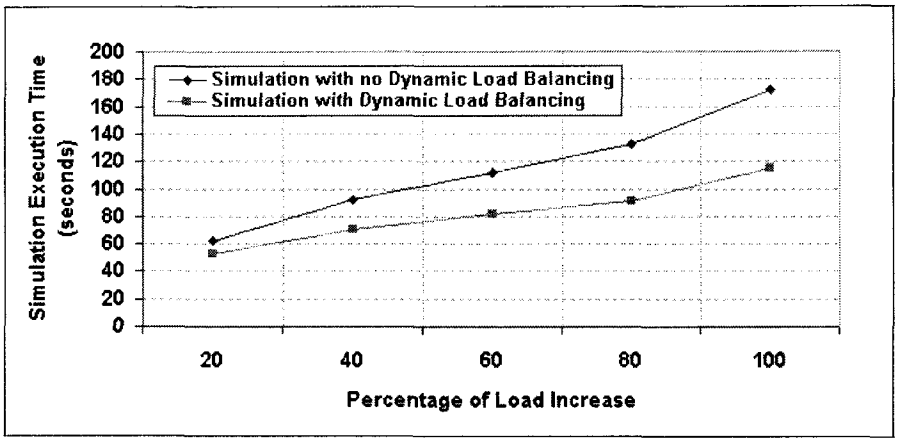

Figure 7.3: **Simulation Comparison with and without Dynamic Load Balancing for an Increasing Percentage of Load using DEVS**

scenario where the *simEntityGroup* model generates only computation events another where it generates only communication events and a last one where we have both computation and communication events. As shown in Figure 7.4, the gain in performance is at most when the simulation encounters both communication and computation load imbalances. The average gain in terms of execution time where only computation events were generated was around 18.6%, where only communication events were generated was around 14.1% and where both loads were generated was around 26%. The higher gain in the computation part as compared to the communication one has no significance here and is caused only by initial configurations.

This experimental result further verifies our proposed design in terms of the importance of considering both computation and communication load imbalances. Therefore, the computation and communication load-balancing capability is an important characteristic and advantage for our proposed design.

In this section, we have demonstrated the powerfulness of the simulation based modeling design approach for the proposed dynamic load balancing algorithm. The DEVS model simulated in DEVSJAVA provides the key design parameters for predicting the crucial

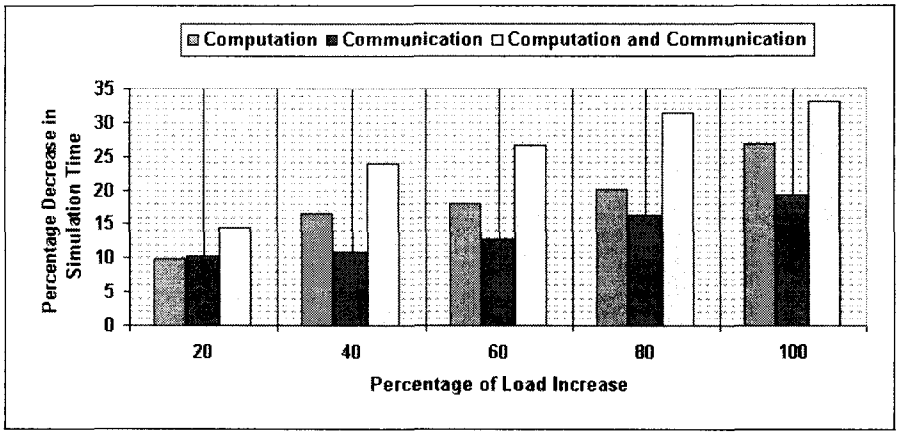

Figure 7.4: **Percentage Decrease in Simulation Time with Different Load Imbalances and an Increasing Percentage of Load using DEVS**

performance characteristic of the real system. It is also necessary to mention that the simulation experiments used here only examine some of the key interests for our design, and further more experiments can be done by changing the experimental frames. The purpose of the above simulation experiments is to verify and validate our design through simulation based modeling using DEVS.

Chapter 8

Conclusion and Future Work

In this thesis, we proposed an adaptive and efficient dynamic load balancing scheme. It is generally applicable and configurable to different simulation requirements and adapts to meet the simulation dynamic behavior. The JXTA service based modular implementation favors our hierarchical scalable design and any future service or architecture extendibility. We verified our proposed dynamic load-balancing scheme by a series of simulation experiments, and our scheme demonstrated a better performance in terms of the simulation execution time. Furthermore, we extended our proposed algorithm to an inter-cluster algorithm and presented a DEVS based formal language approach for our design.

As a future work we plan to implement our global load balancing scheme using a set of clusters at the PARADISE Lab.

References

[1] A. Boukerche and M. Zhang. Towards Peer-to-Peer Based Distributed Simulations on a Grid Infrastructure. In Proceedings of the 41st Annual Simulation Symposium (anss-41 2008), pages 212-219, April 2008.

[2] IEEE Standard. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules. 1516 2000, September 2000.

[3] IEEE Standard. Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Federation Interface Specification. 1516.1-2000, April 2000.

[4] IEEE Standard. Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification. 1516.2-2000, April 2000.

[5] C. D. Carothers, Richard Fujimoto. Background Execution of Time Warp Programs. Workshop on Parallel and Distributed Simulation 1996: 12-19

[6] M. Jiang, R. Anane, G. Theodoropoulos. Load balancing in distributed simulations on the grid. 2004 IEEE International Conference on Systems, Man and Cybernetics, Volume 4, 10-13 Oct. 2004 Page(s):3232 - 3238

[7] J. Cao, D. P. Spooner, S. A. Jarvis, G. R. Nudd. Grid load balancing using intelligent agents. Future Generation Comp. Syst. 21 (1): 135-149 (2005)

[8] J. Wang, Q. Yuan Wu, D. Zheng, Y. Jia. Agent based Load Balancing Model for Service based Grid Applications. International Conference on Computational Intelligence and Security, 2006 Volume 1, pages: 486 - 491, 2006

[9] W. Cai, S. J. Turner, and H. Zhao. A load management system for running hla-based distributed simulations over the grid. In DS-RT '02: Proceedings of the Sixth IEEE

International Workshop on Distributed Simulation and Real-Time Applications, page 7, Washington, DC, USA, 2002. IEEE Computer Society.

[10] K. Zajac, M. Bubak, M. Malawski. Towards a Grid Management System for HLA-based Interactive Simulations. In Proceedings of The Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications, Delft, The Netherlands, pp. 4-11, 2003.

[11] Z. Yuan, W. Cai, and M. Yoke H. Low. A framework for executing parallel simulation using rti. In DS-RT '03: Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications, page 12, Washington, DC, USA, 2003. IEEE Computer Society.

[12] L. F. Wilson and W. Shen. Experiments in load migration and dynamic load balancing in speedes. In Proceedings of the 1998 Winter Simulation Conference, pages 483-490. IEEE Computer Society, 1998.

[13] P. Patrick, Tobias H., P. Martini. A Flexible Dynamic Partitioning Algorithm for Optimistic Distributed Simulation. In Proceedings of the 21st Workshop on Parallel and Distributed Simulation (PADS'07), pages 219-228. IEEE Computer Society, 2007.

[14] M. Y. H. Low. Dynamic load-balancing for BSP time warp. In Proceedings of the 35th Annual Simulation Symposium (SS'02), pages 267-274. IEEE Computer Society, 2002.

[15] A. Boukerche and S. K. Das. Dynamic load balancing strategies for conservative parallel simulations. In Proceedings of the 11th Workshop on Parallel and Distributed Simulation (PADS'97), pages 32-37. IEEE Computer Society, 1997.

[16] G.Tan, and K.C.Lim. Load Distribution Services in HLA. In proceedings of 8th IEEE Distributed Simulation and Real-time Applications, Budapest, Hungary, October 2004, pp 133-141.

[17] G. Theodoropoulos and B. Logan. An approach to interest management and dynamic load balancing in distributed simulation. In Proceedings of the 2001 European Simulation Interoperability Workshop, pages 565-571, July 2001.

[18] Morse, K. L. (1996). Interest management in large scale distributed simulations. Technical Report 96-27, Department of Information and Computer Science, University of California, Irvine.

[19] R. Schlagenhaft, M. Ruhwandl, C. Sporrer, and H. Bauer. Dynamic load balancing of a multi-cluster simulation on a network of workstations. In proceedings of 9th Workshop on Parallel and Distributed Simulation. Society for Computer Simulation, 1995, pp. 175-180, Society for Computer Simulation.

[20] C. Burdorf and J. Marti. Load Balancing Strategies for Time Warp on Multi-User Workstations. The Computer Journal, 36(2): 168-176, 1993.

[21] B. P. Gan, Y. H. Low, S. Jain, S. J. Turner, W. Cai, W. J. Hsu, S. Y. Huang. Load balancing for conservative simulation on shared memory multiprocessor systems. In proceedings of the 14th Workshop on Parallel and Distributed Simulation, 28-31 May 2000,pp. 139-146

[22] K. El-Khatib and C. Tropper. On metrics for the dynamic load balancing of optimistic simulations. In proceedings of the 32nd Hawaii International Conference on System Sciences. IEEE Computer Society, 1999.

[23] JXSE 2.5 Programmers Guide:JXTA Concepts, https://jxta.dev.java.net/.

[24] JXTA Protocols Specification, http://jxtaspec.dev.java.net.

[25] E. Halepovic, R. Deters. The costs of using JXTA. Proceedings of the Third International Conference on Peer-to- Peer Computing, 2003. (P2P 2003),Pages 160-167, 2003.

[26] G. Antoniu, M. Jan, D. A. Noblet. Enabling the P2P JXTA Platform for High-Performance Networking Grid Infrastructures. In proceedings of the first Intl. Conf. on High Performance Computing and Communications (HPCC'05), Sorrento, Italy, Lect. Notes in Comp. Science, Springer-Verlag, September 2005, no3726, p. 429440.

[27] B. P. Zeigler, T.G. Kim, and H. Praehofer. Theory of Modeling and Simulation. 2 ed. New York, NY: Academic Press, 2000

[28] M. Eklf, M. Sparf, F. Moradi, R. Ayani. Peer-to-Peer-Based Resource Management in Support of HLA-Based Distributed Simulations. SIMULATION, Vol. 80, Issue 4-5, April-May 2004, 181-190.

[29] A. E. Saddik and A. Dufour. Peer-to-Peer Suitability for Collaborative Multiplayer Games. In proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT03), Page 101, 2003.

[30] D. C. Parker. S. A. Collins, and D. C. Cleary. E. Ireland, Building Near Real-Time P-2-P Applications with JXTA. 2004 IEEE International Symposium on Cluster Computing and the Grid, Page 338-345, 2004.

[31] C. Qu, W. Nejdl. Interacting the Edutella/JXTA Peer-to-Peer Network with Web Services. In proceedings of the 2004 International Symposium on Applications and the Internet (SAINT04), Page 67-73, 2004.

[32] G. Antoniu, P. Hatcher, M. Jan, D.A. Noblet. Performance Evaluation of JXTA Communication Layers. 5th International Workshop on Global and Peer-to-Peer Computing (GP2PC 05), Cardiff, UK (2005) Held in conjunction with the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005), Page 251-258, 2005.

[33] B. Zeigler H. Sarjoughian. Introduction to DEVS Modeling & Simulation with JAVATM: Developing Component-based Simulation Models. Arizona Center for Integrative Modeling and Simulation, Arizona State University, March, 2003

[34] Azzedine Boukerche, Ming Zhang. Time Management Service in JXTA Based Distributed Simulation. In proceedings of 12^{th} IEEE International Symposium on Distributed Simulation and Real Time Application, pp. 167-173, 2008.

[35] Schulz, S., J.W. Rozenblit, M. Mrva and K. Buchenrieder. Model-Based Codesign. IEEE Computer, 31(8): 6067, 1998.

[36] Hu, X. and B.P. Zeigler. 2005. Model Continuity in the Design of Dynamic Distributed Real-Time Systems. IEEE Transactions on Systems,Man And Cybernetics Part A: Systems And Humans, 35(6): 867878.

[37] R. M. Fujimoto. Parallel simulation: parallel and distributed simulation systems. In Proceedings of the 2001 Winter Simulation Conference, pages 147157. ACM, 2001.

[38] Forster I. And Kesselman C. (Eds.), The Grid Blueprint for a new Computing Infrastructure, Morgan Kauhan, 1999

[39] I. Foster, C. Kesselman and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In International J. Supercomputer Applications vol. 15, no. 3, 2001.

[40] I. Foster et al. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. tech. report, Glous Project; http://www.globus.org/research/papers/ogsa.pdf (current June 2002).

[41] Donald Knuth. The Art of Computer Programming. 3rd edition, volume 1, 1997, p.98

[42] Azzedine Boukerche, Ahmad Shadid, Ming Zhang, "Efficient Load Balancing Schemes for Large-Scale Real-Time HLA/RTI Based Distributed Simulations," 11th

IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'07), pp. 103-112, 2007.Donald Knuth. The Art of Computer Programming. 3rd edition, volume 1, 1997, p.98

[43] Ming Zhang, B.P. Zeigler, P. Hammonds, "DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies", International Test & Evaluation Association (ITEA) Journal of Test and Evaluation, March/April 2006, Volume 27, Number 1, Page 49-60.