uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Ahmad Jamal Shadid
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M. (Computer Science)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Efficient Load Balancing Techniques for Real-Time RTI-Based Large-scale Distributed Simulation Systems

TITRE DE LA THÈSE / TITLE OF THESIS

Azzedine 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

Dorina Petriu

Voicu Groza

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

EFFICIENT LOAD BALANCING TECHNIQUES FOR REAL-TIME RTI-BASED LARGE-SCALE DISTRIBUTED SIMULATION SYSTEMS

by

AHMAD JAMAL SHADID

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

December 2007

Your file Votre référence
ISBN: 978-0-494-50923-4
Our file Notre référence
ISBN: 978-0-494-50923-4

Abstract

The Real-time extension of the High Level Architecture (HLA) is very essential and useful for many large-scale distributed simulation systems. Most previous attempts to design the Real-Time Run Time Infrastructure (RT-RTI) have enabled the usage of supported scheduling and prioritization services from underlying Real-time operating systems (RTOSs) augmented by communication QoS mechanisms. In this thesis, we wish to build on this functionality by proposing an algorithm that differentiates services processing within the RTI itself by incorporating resources' load balancing mechanisms with several scheduling and allocation policies. We focus our efforts on making the RTIs internal operations organized and well suited to the tasks and services it will be providing throughout the lifetime of HLA-compliant federations.

The load-balancing mechanisms incorporated within the proposed system have been designed to build an efficient management scheme of the available processing resources. The proposed design have been heavily verified and tested from all related perspectives under the constraints of the Discrete Event System Specification (DEVS), especially the (RT-DEVS). This formalized system design approach have propagated a totally-new RT-RTI design that solves most of the shortcomings demonstrated and shown by previous designs.

We have proved through our analytical performance evaluation and simulation experiments that our proposed real-time RTI framework exhibits a better performance in all terms, especially in the number of tasks served within deadlines when compared with existing RT-RTI frameworks. Our simulation results from the design model of the RT-DEVS formalism have verified the experimental results of the real implementation of the proposed RT-RTI design. In addition, these simulation models are capable of predicting key factors that may affect the performance of the RT-RTI core system, and could be used as guidance in searching for the optimal design.

Table of Contents

Page

Abstract ii

Table of Contents iii

List of Tables v

List of Figures vi

List of Acronyms vii

List of Publications viii

Acknowledgements ix

Chapter

1 Introduction 1

1.1 Real-Time Run-Time Infrastructure Motivation 1

1.2 Real-Time Run-Time Infrastructure Objective 2

2 High Level Architecture 5

2.1 High Level Architecture Framework 5

2.2 Components of the HLA Framework 7

2.2.1 Object Model Template 7

2.2.2 Federate Interface Specification 10

2.2.3 HLA Federate and Federation Rules 11

3 Discrete Event System Specification 15

3.1 Discrete Event System Specification Overview 15

3.2 DEVS and RT-RTI Based Systems 21

4 Previous And Related Works 24

4.1 Real-Time RTI Based Systems 24

4.2 Real-Time DEVS Based Systems 27

5 Proposed Real-time RTI Design and Implementation 30

5.1 Overview of RT-RTI Design 30
5.2 RTI-Core Design Approaches 31
5.2.1 Static Load Balancing Strategy 33
5.2.2 Dynamic Load Balancing Strategy 34
5.3 Description of RT-RTI Components 36
5.3.1 Dispatching and Allocation Mechanisms 37
5.3.2 Threads Management 40
5.3.3 Fixed Priority Global Scheduling 42
6 RT-RTI Formalization and Verification using RT-DEVS 44
6.1 Formalization Motivation 45
6.2 Design Requirements and Verification 46
7 Performance Evaluation of RT-RTI 52
7.1 Real Implementation Experimental Results 53
7.1.1 Analytical Performance Evaluation 53
7.1.2 Simulation Experiments 57
7.2 RT-DEVS Formalism Experimental Results 60
7.2.1 Dynamic Thread Pool Management in a Modeled RT-RTI System . 61
7.2.2 Load Balance of Services in the Modeled RT-RTI System 63
8 Conclusion and Future Work 66
8.1 Summary of Contributions 66
8.2 Future Work 67
References 68

List of Tables

4.1 **Differences between RT-RTI Designs** 29

7.1 **Deadlines Satisfaction Percentage of RT-RTI Designs** 64

List of Figures

3.1 Relations between DEVS Objects [7] 17

5.1 High-level RTI view 31
5.2 General Design of RT-RTI 33
5.3 Recursive Dynamic LB Procedure 40
5.4 Thread Pool with Lanes 42

6.1 RT-RTI DEVSJAVA Formalism Representation 47

7.1 1_Queue_Model service rate (μ) 55
7.2 6_Queues_Model service rate (μ) 56
7.3 RT-RTI Designs Comparison 58
7.4 Simulation Time Overhead 59
7.5 Effects of Dynamic Thread Pool Management 62

List of Acronyms

HLA High Level Architecture

RTI Run-Time Infrastructure

RT-RTI Real-Time Run-Time Infrastructure

DEVS Discrete Event System Specification

RT-DEVS Real-Time Discrete Event System Specification

RTOS Real-Time Operating System

GPOS General Purpose Operating System

OMT Object Model Template

SOM Simulation Object Model

FOM Federation Object Model

DDM Data Distribution Management

TM Time Management

IntServ Integrated Service

DiffServ Differentiated Service

MPLS Multiple Protocol Label Switching

VRTP Virtual Reality Transfer Protocol

SQ Service Queue

List of Publications

The following publications are relevant to the topic of this thesis and have been authored by Ahmad Shadid.

Conferences and Workshops:

1. 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.

2. Azzedine Boukerche, Ahmad Shadid, Ming Zhang, "A Formal Approach to RT-RTI Design Using Real Time DEVS," 6th IEEE International Workshop on Haptic, Audio and Visual Environments and Games (HAVE 2007) , pp.84-89, 2007

3. Azzedine Boukerche, Abdulaziz Al Hamidi, Luqman Ahmad, Ahmad Shadid, R. Pazzi, "A Novel Design of a Virtual Hospital over HLA/RTI", Under Review.

Journals:

1. Azzedine Boukerche, Ming Zhang, Ahmad Shadid, Verification of a Novel RT-RTI Design Using Real Time DEVS, Submitted to IEEE Transactions on Parallel and Distributed Systems, July, 2007.

In addition, our paper titled "Efficient Load Balancing Schemes for Large-Scale Real-Time HLA/RTI Based Distributed Simulations" obtained the Best Paper Award in the 11th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2007) held in Crete Island, Greece (October, 2007).

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. Luqman Ahmad, Mr. Raed Jarrar, Mr. Samer Samara, Mr. Elie El Ajaltouni, M. Haifa Maamar, Mr. Yungfeng Gu, Mr. Richard Pazzi and Dr. Ming Zhang. 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

MOTHER, FATHER and BROTHERS A.K.R.

with love and appreciation

Chapter 1

Introduction

In this chapter, we provide an introduction to the topic we discuss throughout this thesis, that is the Real-Time extension to the Run Time Infrastructure (RT-RTI). As will be seen in the following chapters, RTI is the federate interface specification of the High Level Architecture HLA, and thus, being able to exactly understand this standard from all its perspectives and the relations formed towards its RTI component is an ideal base from which the following proposed ideas can proceed. In addition, we don't only provide the description of our idea from a detailed view but we also verify our designing approach through a design formalization process using the Discrete Event System Specification formalism (DEVS) and its corresponding real-time extension (RT-DEVS).

Hereunder, we are discussing the important need and motivation for the RT-RTI. It is a vital issue to be discussed indeed, since we believe that our referenced framework extension is very critical and needed in many areas of simulation systems especially that the Modeling and Simulation (M&S) community is expanding to tackle all kinds of computing areas. We also show and describe the sight through which our proposed idea and scheme can be seen and judged from a general overview. Our specific design details and differences are described in detail in following chapters as this thesis proceeds.

1.1 Real-Time Run-Time Infrastructure Motivation

The HLA-RTI framework has become a standard for simulation systems in all different types of applications since its release in 2000. Nowadays, many simulation systems need their services to be provided in real-time, including e-learning applications, analytical simu-

lations, military training, and distributed virtual environments. These kinds of applications and many others ask for the real-time extension of the RTI in order to have their services provided in a more reliable and a predictable fashion. Real-Time services are not the only crucial part in our proposed extension, instead, it is the reliability and flexibility this system demonstrates when it becomes real-time. This is the general sense that differentiates the core part from the other positive side effects of an extension to such a huge standard.

The view through which we look at the real-time support in our proposed RT-RTI system, is that we have many tasks, each of which is originated from a specific federate and has a pre-defined deadline. This deadline is the deciding factor that will guide us through the prioritization process of this specific task and thereafter processing it. This means that inevitably, we need to have a reliable prediction of the worst-case scenario that may arise at any point in time and to how to deal with it efficiently and effectively without the un-liked overhead. In addition, users require a system that is sufficiently reliable in terms of guaranteed response times. It is worth mentioning that real-time systems in general are systems that deliver results within deadlines, neither faster nor slower, and that is the definition we start from in defining the extension of the real-time RTI we discuss in this thesis.

1.2 Real-Time Run-Time Infrastructure Objective

We have to clarify that what we are proposing in this RT-RTI design is majorally dependent on the way we balance the work and manage the resources available in an efficient and a just manner. It is obvious that we deal with distributed systems either geographically or logically, and the issue that arise here is the network specifications. Therefore, from one side, we assume that the Network QoS is supported and the delay that could arise is bounded. In addition, from another side, we are dealing with simulation systems, and this dictates that the time management issues and synchronization concerns are of vital importance. Therefore, we assume that the synchronization between local simulation times and

the wall-clock time will be applied within the time-step (time-stepped simulation systems) Δt period after processing the tasks received and generating responses.

All the different services that are provided by the RTI must have an important effect on the way we see the required real-time support. It was not truly the case on the previous RT-RTI design efforts. In our proposed design and verification, all of the facts these services impose on the RTI design, even all the special cases are considered. So that, the proposed design is an overall real-time RTI structure that can be used with any system having similar characteristics.

We know that real-time RTI has been investigated and the usage of the scheduling capabilities supported by both Real-Time Operating Systems (RTOS) and General Purpose Operating Systems (GPOS) with the communication network QoS parameters has been put heavily on use, all without taking the real relationships between the federates' tasks and how these relations affect the available resources. When it comes to resources, then it becomes clear that managing the resources in an efficient manner is an absolute need for these tasks to be processed correctly within their deadlines and under the umbrella of other constraints. We have found that there are many possibilities for these relationships to be investigated in a sense that not only the already tested and well-known algorithms of the RTOS and networks QoS mechanisms can be useful. These relationships have the potential to be really studied and utilized in a wide horizon as will be seen throughout this thesis.

Having the desired model of real-time RTI is an absolute case that incorporates the communication models with processing schemes all leveraged under the same performance measure. Those different parts should all be worked out in a way that does not only have the real-time RTI purpose accomplished, but also having it accomplished with the minimum cost and overhead.

The organization of this thesis is as follows: Chapter 2 provides a well structured introduction to the High Level Architecture HLA that is very needed to build upon in the following chapters. In addition, Chapter 3 builds a clear base for understanding the Discrete Event System Specification (DEVS), that will also be needed when it comes to the

verification of our proposed design among different alternatives. Afterwards, in Chapter 4, we review the literature of the other Real-Time RTI (RT-RTI) designs and efforts, as well as reviewing the previous RT-DEVS designs' verification efforts and experimentation. In Chapter 5, we go a step further and propose our RT-RTI design specifications, all with the schemas and algorithms for solving the issues overseen as a problem definition. While in Chapter 6, we describe the route we followed for our design verification using the RT-DEVS formalism in order to make sure that the most optimum and efficient design out of many other alternatives have been built. Chapter 7 shows our experimental frames and results to prove the design's efficiency and usefulness from the real implementation point of view as well as the RT-DEVS formalism. We conclude the thesis with some future work suggestion in Chapter 8.

Chapter 2

High Level Architecture

In this chapter we discuss the High Level Architecture (HLA) with all related issues from the demonstrated need for this standard, its components; the Object Model Template (OMT), Federate Interface Specification and the Federate/Federation Rules. In this chapter, we are trying to show the different HLA perspectives needed to be known before tackling the issue of the RT-RTI and also, it builds a basic and a reliable base of knowledge for the High Level Architecture in general.

2.1 High Level Architecture Framework

In fact, simulations are abstractions of the real world components that need to be simulated, and no one simulation can solve all of the functional needs for the modeling and simulation community without an overall scheme and paradigm. It is anticipated that technology advances will allow for new and different modeling and simulation (M&S) implementations within a specific framework in which all (or most of the) different functional and procedural details are described and detailed. This standard is the High Level Architecture (HLA) that was developed and standardized by the Department of Defence (DoD) in 2000. The components of the HLA standard are interrelated to each other and need to be considered as complete set in which any change in one is likely to have an impact on the others. As such, the HLA is an integrated approach that has been developed to provide a common architecture for simulation.

The HLA provides a general framework within which simulation developers can structure and describe their simulation applications. Flexibility, interoperability and reusability

are the aim of the HLA. In particular, the HLA addresses two key issues with the large-scale distributed interactive simulation systems, promoting interoperability between simulation systems and aiding the reuse of models in different contexts. The way through which these issues have been realized was through 2 major components of the HLA (will be described in detail in following sections). The first is the Object Model Template (OMT), which forms a documentation standard describing the data used by a particular model from all different perspectives, this is basically realizing the the reusability issue presented by the HLA. The second component, the Federate Interface Specification, describes a generic communications interface that allows simulation models to be connected and coordinated, thus, addressing the interoperability issue. It is very critical to see the HLA as an architecture, not a software. However, the use of the Run Time Infrastructure (RTI) software is required to support operations of a federation execution in the exchange protocol of information between different simulation systems. The RTI software provides a set of services, as defined by the Federate Interface Specification, used by federates to coordinate operations and data exchange during a runtime execution.

As described in the IEEE Standard 1516.1 [19], in defining an overriding architecture like the HLA, one of the major issues is to have a description and an explanation of all the concerns raised within the M&S community, however, generic issues must be addressed because of the different ways challenges will be solved in different implementations following the same HLA standard. When doing so, it is essential that such an architecture encompass both differing computing environments and differing classes of simulations.

As mentioned before, HLA can be seen as an architecture for creating software simulations out of component simulations in a large-scale distributed interactive environment. From one side, It provides a general scheme within which the simulation developers can structure and describe their simulation applications in an efficient and precise manner. On the other hand, it provides a very specific scheme for the interaction and communication tunnels through which the different federates will be interacting with each other.

As it is well known, HLA has defined and structured the important properties of

reusability and interoperability in the world of distributed simulations. Many complex simulations have individual simulation components that have been created under several types of platforms. Often, some components of these systems may need to be used elsewhere. However, when this individual simulation component is inserted into another simulation system that doesn't follow the HLA framework, the integration phase becomes very difficult and complex. In some cases, it is easier to rebuild that simulation component completely than to adapt an existing one due to this costly and complex overhead, this example fully justifies the need for such an overwhelming overall framework.

2.2 Components of the HLA Framework

HLA has three major parts: Rules, Federate Interface Specification, and the Object Model Template (OMT). All of these components outline a strong collaboration with each other in organizing any large-scale distributed interactive simulation system.

2.2.1 Object Model Template

We start describing the Object Model Template (OMT) component of HLA; because its items are used to describe other major HLA components. The OMT specification defines the format and syntax (but not content) of HLA object models [18]. It prescribes the format and syntax for recording information about objects and interactions in the simulation system. It fosters its documentation upon different levels of abstraction independently, allowing flexibility in the exchange of information and components interaction. The object model template (OMT) provides a common framework for the communication between HLA simulations.

As mentioned before, OMT facilitates the interoperability among simulations and reuse of simulation components.its importance comes from 3 basic reasons as mentioned in [18]. The first one is the that it provides a commonly understood mechanism for specifying the exchange of data and general coordination among members of a federation. In addition,

it provides a common, standardized mechanism for describing the capabilities of potential federation members. Lastly, it facilitates the design and application of common tool sets for development of HLA object models.

The OMT consists of the 2 major models on 2 levels; the federate and the federation. They are described as follows [19], [18]:

Federation Object Model (FOM) The FOM describes the shared object, attributes and interactions for the whole federation. During development of an HLA federation, it is critical that all federation members achieve a common understanding as to the nature or character of all required communications among participating federates. The primary purpose of an HLA FOM is to provide a specification for data exchange among federates in a common, standardized format. The content of this data includes an enumeration of all object and interaction classes pertinent to the federation and a specification of the attributes or parameters that characterize these classes. Taken together, the individual components of an HLA FOM establish the .information model contract. that is necessary (but not sufficient) to achieve interoperability among the federates [19], [18].

Simulation Object Model (SOM) A SOM describes the shared object, attributes and interactions used for a single federate. A critical step in the formation of a federation is the process of determining the composition of individual federates to best meet the overall objective. An HLA SOM is a specification of the types of information that an individual federate could provide to HLA federations and the information that an individual federate could receive from other federates in HLA federations. The standard format in which SOMs are expressed facilitates determination of the suitability of federates for participation in a federation. The HLA OMT formats described in this document are generally applicable to either FOMs or SOMs. Thus, SOMs are also characterized by their objects, attributes, interactions, and parameters. The primary benefit from the common utilization of the OMT formats for FOMs

and SOMs is that it provides a common frame of reference for describing object models in the HLA community. In some cases, this commonality may even allow SOM components to be integrated as .piece parts. in a FOM, facilitating FOM construction.

Although, HLA Object Model Template (OMT) is the standardized documentation structure for HLA object models, Federation Object Models (FOMs) and Simulation Object Models (SOMs) do not completely correspond to common definitions of object models in the well known object-oriented (OO) analysis and design (OOAD) techniques. In the OOAD literature, an object model is described as an abstraction of a system developed for the purpose of fully understanding the system. To achieve this understanding, most OO techniques recommend defining several views of the system. On the other hand, HLA object models intended scope of the system description is much narrower, focusing specifically on requirements and capabilities for federate information exchange [18]. For example, in the SOM, the intent is to describe the public global interface of the federate in terms of an identified set of supported HLA object classes and interaction classes. It provides a more detailed vision of how the federate should be functioning and interacting within itself and the other external components. For FOMs, the intent is to describe information exchange that happens during a federation execution.

In addition, the difference between HLA and OOAD principles and concepts appear at the individual object definition level. The OOAD literature defines objects as encapsulations of data and methods (functions). However, in the HLA terminology, the object definition depends on identifying characteristics (attributes), values of which are exchanged between federates during a federation execution. Also, OO objects interact via message passing, in which one OO object invokes an operation provided by another OO object. On the other hand, HLA objects do not directly interact. It is the federates that interact, via HLA services, by updating instance attribute values or sending interactions.

2.2.2 Federate Interface Specification

Another critical and vital component of the HLA is the Federate Interface Specification that defines the Run-Time Infrastructure (RTI); it is the actual and formal implementation of it. It is very important to know that the RTI is a software that conforms to the specification but is not itself a part of it. It is not only the part that defines functional interfaces by which different services provided by the RTI will be requested but it will also provide those services; that are necessary to support HLA-compliant simulation systems. Besides that, it should identify the call back functions on the other way around from itself back to the federate.

As well known, HLA is an integrated architecture that has been developed to provide a common architecture for M&S. The HLA requires that inter-federate communications use a standard application programmer's interface (API) which is the RTI. This specification defines the standard services and interfaces to be used by the federates in order to support efficient information exchange when participating in a distributed federation execution. Additionally, the capability for reuse of individual federates that adhere to these standard services and interfaces is increased. RTI simply provides a specification for the HLA functional interfaces between federates and the RTI. The RTI provides services to federates in a way that is analogous to how a distributed operating system provides services to applications [17]. These interfaces are arranged into seven basic service groups. They describe the interface between the federates and the RTI, and the software services provided by the RTI for use by HLA federates. these services are all described thoroughly in [17]:

1. Federation management
2. Declaration management
3. Object management
4. Ownership management
5. Time management

6. Data distribution management

The Run-Time Infrastructure (RTI) acts as a core provider of services that could be requested by simulation federates at any time during the simulation (from the initialization of a federation until the resignation of the last federate). Therefore, as this thesis tackles the issue of the Real-Time RTI, we believe that having this core RTI provider organized -in terms of how to deal with requests- as much as possible, with taking all possibilities and cases into consideration would be the ultimate purpose of having it real-timed, yet, we will not have the related efforts described in Chapter 4 towards this purpose under-utilized.

2.2.3 HLA Federate and Federation Rules

The last component of the HLA framework is the Rules governing the federate and federation levels. They govern the whole picture of building and running large-scale distributed interactive simulation systems. The Federate and Federation rules can be seen as a constitution; not only to describe the responsibilities of federates, but also to ensure their proper interaction under the umbrella of a specific federation. All of the 10 rules (five for federates and five for federations) are acting as a satisfaction guarantee when it comes to how should these federations and federates comply with the interoperability and usability pledges. Herein the different rules are mentioned in sequence with a brief description as in [19].

- ***Federation rules are:***

  Federations shall have a HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).

  All data to be exchanged in accordance within the HLA compliant federations shall be documented in a FOM. A FOM shall document the agreement among federates on data to be exchanged using the HLA services during federation execution and the minimal set of conditions of the data exchange (e.g., updates to be sent when changes

exceed a certain value) [19]. As such, a FOM is an essential element in defining a federation. The HLA does not prescribe which data are included in a FOM (this is the responsibility of the federation user and developer).

In a federation, all representation of objects in the FOM shall be in the federates, not in the Run-Time infrastructure (RTI).

In the HLA, the responsibility for maintaining the values of HLA object instance attributes shall take place in the joined federate. In an HLA federation, all joined federate-associated instance attributes shall be owned by federates, not by the RTI. However, the RTI may own instance attributes associated with the federation Management Object Model. The RTI may use data about instance attributes and interactions to support RTI services (e.g., Declaration Management), but these data are merely used by the RTI, not changed.

During a federation execution, all exchange of FOM data among federates shall occur via the RTI.

The HLA federate interface specification [17] specifies a set of interfaces to services in the RTI to support coordinated exchange of instance attribute values and interactions in accordance with a federation's FOM. Under the HLA, intercommunication of FOM data among joined federates participating in a given federation execution shall be executed by the exchange of data via the RTI services. Based on the FOM, joined federates shall identify to the RTI what information they will provide and require, along with instance attribute and interaction data corresponding to the changing state of object instances in the joined federate. The RTI shall then provide the coordination, synchronization, and data exchange among the joined federates to permit a coherent execution of the federation.

During a federation execution, federates shall interact with the run-time infrastructure (RTI) in accordance with the HLA interface specification.

The HLA provides a specification for a standard interface between the federate ap-

plication and the RTI. Joined federates shall use this standard interface to access RTI services [17]. The specification shall define how federate applications interact with the infrastructure. However, because the interface and the RTI will be used for a wide variety of applications requiring data exchange of diverse characteristics, the interface specification says nothing about the specific federate data to be exchanged over the interface.

During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.

The HLA allows for different joined federates to own different attributes of the same object instance (e.g., a simulation of an aircraft might own the location of the airborne sensor, whereas a sensor system model might own other instance attributes of the sensor). To ensure data coherency across the federation, at most, one joined federate may own any given instance attribute of an object instance at any given time. Joined federates may request that the ownership of instance attributes be acquired or divested, dynamically, during federation execution. Thus, ownership can be transferred, dynamically during execution, from one joined federate to another.

- ***Federate rules are:***

  Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT).

  The HLA SOM shall include those object classes, class attributes, and interaction classes of the federate that can be made public in a federation. The HLA does not prescribe which data are included in the SOM; this shall be the responsibility of the federate developer.

  Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.

  The HLA allows for joined federates to make internal object representations and inter-

actions available for external use as part of federation executions. These capabilities for external interaction shall be documented in the SOM for the federate. If documented in the SOM, these federate capabilities shall include the obligation to export updated values of instance attributes that are calculated internally in the federate and the obligation to be able to exercise interactions represented externally (i.e., by other federates in a federation).

Federates shall be able to transfer and/or accept ownership of an attribute dynamically during a federation execution, as specified in their SOM.

The HLA allows ownership of instance attributes of an object instance to be transferred dynamically during a federation execution. The instance attributes of a federate that can be either owned or reflected, and whose ownership can be dynamically acquired or divested during execution, shall be documented in the SOM for that federate.

Federates shall be able to vary the conditions under which they provide updates of attributes of objects, as specified in their SOM.

The HLA permits federates to own (i.e., provides the privilege to produce updated values for) instance attributes of object instances represented in the federate and to then make those values available to other federates through the RTI. Different federations may specify different conditions under which instance attributes will be updated [e.g., at some specified rate, or when the amount of change in value exceeds a specified threshold (such as altitude changes of more than 1000 ft, etc.)]. The conditions applicable to the update of specific instance attributes owned by a federate shall be documented in the SOM for that federate.

Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

Federation designers will identify their time management approach as part of their implementation design. Federates adhere to the chosen time management approach.

Chapter 3

Discrete Event System Specification

In this chapter we discuss the Discrete Event System Specification formalism (DEVS), with the conventions surrounding this formal modeling and simulation language. In addition, we demonstrate the relation between the DEVS and the RT-RTI from the application perspective of DEVS methodologies upon the proposed design, and achieving outstanding experimental results when combining both in the verification of the implemented proposed ideas. The DEVS formalism supported the results of the real implementation shown in following chapters, and therefore, it becomes more reliable that the design we are showing is a verified and a well-tested one. On the other hand, the usage of the DEVS formalism helped in pinpointing the potential shortcomings of the proposed design that could not be shown when running experiments on the real implementation itself. Therefore, the DEVS formalism is a very helpful tool that can be used in building any discrete event simulation system or extension.

3.1 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 for-

malism 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 [7].

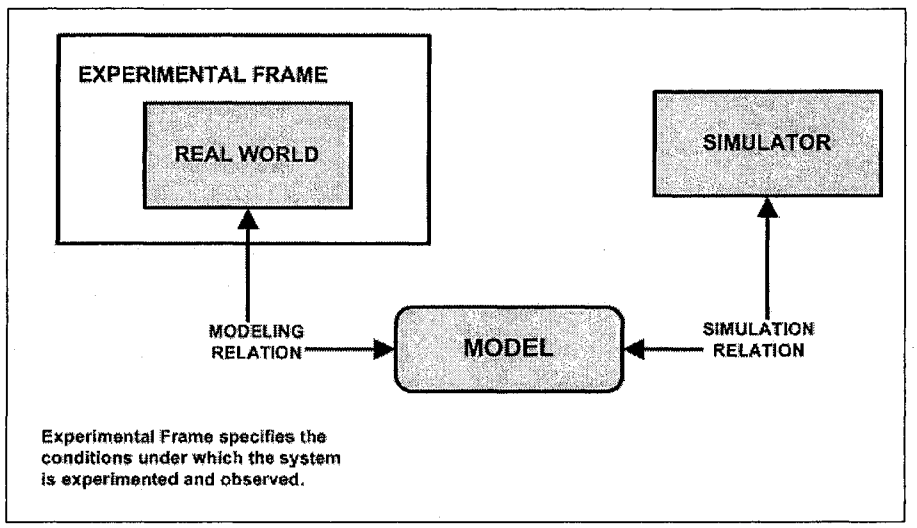

Figure 3.1: **Relations between DEVS Objects [7]**

As in Figure 3.1, the basic components of the DEVS formalism described in the previous list and their relations 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.

The term *"formalism"* we use refers to the way we express the structure of any model in a mathematical language. The discrete event formalism focuses on the changes of variable values and generates time segments that are piecewise constant. Thus an event is a change in a variable value, which occurs instantaneously. In essence the formalism defines how to generate new values for variables and the times the new values should take effect. An important aspect of the formalism is that the time intervals between event occurrences are variable (in contrast to discrete time where the time step is a fixed number).

In the DEVS formalism, there are two points that need to be described, the first one is the *basic models* from which larger ones are built, and the second one shows how these models are connected together in a hierarchical fashion to form more complicated and larger models. To specify modular discrete event models, we need to adopt a different view than other described by traditional simulation languages. As with modular specification in general, we must view a model as possessing input and output ports through which all interaction with the environment is performed. In addition, the inputs and outputs are the deciding factors through which transition in states either locally or globally is actuated. In the discrete event case, events determine values appearing on such ports. More specifically, when external events, arising outside the model, are received on its input ports, the model description must determine how it responds to them including the current state of the model from which the transition can be performed. Also, internal events arising within the model change its local state, as well as manifesting themselves as events on the output ports to be transmitted to other model components.

Any basic model either used to present something simple or to build a more complex hierarchal model contains the following information:

- The set of input ports through which external events are received
- The set of output ports through which external events are sent
- The set of state variables and parameters: two state variables are usually present, phase and sigma (in the absence of external events the system stays in the current phase for the time given by sigma)
- The time advance function which controls the timing of internal transitions when the sigma state variable is present, this function just returns the value of sigma.
- The internal transition function which specifies to which next state the system will transit after the time given by the time advance function has elapsed.
- The external transition function which specifies how the system changes state when an input is received the effect is to place the system in a new phase and sigma thus scheduling it for a next internal transition; the next state is computed on the basis of the present state, the input port and value of the external event, and the time that has elapsed in the current state.
- The confluent transition function which is applied when an input is received at the same time that an internal transition is to occur the default definition simply applies the internal transition function before applying the external transition function to the resulting state.
- The output function which generates an external output just before an internal transition takes place. Coupled Models Basic models may be coupled in the DEVS formalism to form a coupled model. A coupled model tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction.

A coupled model contains the following information:

- The set of components
- The set of input ports through which external events are received
- The set of output ports through which external events are sent

These components can be synthesized together to create hierarchical models having external input and output ports.

The coupling specification consisting of:

- The external input coupling which connects the input ports of the coupled to model to one or more of the input ports of the components this directs inputs received by the coupled model to designated component models.
- The external output coupling which connects output ports of components to output ports of the coupled model- thus when an output is generated by a component it may be sent to a designated output port of the coupled model and thus be transmitted externally.
- The internal coupling which connects output ports of components to input ports of other components- when an input is generated by a component it may be sent to the input ports of designated components (in addition to being sent to an output port of the coupled model).

DEVS is basically a mathematical formalism used to describe real-world system behaviors in an abstract and rigorous manner. The models that are created using DEVS are all conceptual models of systems either intended to be built or ready to use. In the first case, the models are needed to find the best suitable design approach with the most efficient architecture alternatives. In the second case where the system is already-built and ready to use, it is a critical phase because of the verification and testing requirements in all large-scale and reliable systems.

DEVS has defined its standard as a well-known discrete event modeling and simulation methodology as described before. Compared with Non-DEVS traditional modeling and simulation methodologies, DEVS defines a strict and concrete modeling and simulation framework that supports fully object oriented modeling and simulation. The critical importance of DEVS is realized by the experimental frames that could be applied within the modeling and simulation frameworks. Furthermore, DEVS has been proved to be effective not only for discrete event models but also for continuous spatial and hybrid models.

With the help of modern object oriented language such as C++ and Java, the frameworks for modeling and simulation based on DEVS have reached their mature stages and have been applied in many real-world applications in order to find the best combination of design and architectural alternatives. With the increased demand for high-performance and large-scale simulation frameworks, parallel and distributed simulations are called on to support various scientific and engineering studies, including technical (e.g., standards conformance), system level (focus on a single natural or engineered system) and operational (focus on multiple systems, such as families of systems or system of systems). The objectives of such studies may include testing of correctness of system behavior/function, evaluation of measures of performance, and evaluation of measures of effectiveness and key performance parameters.

3.2 DEVS and RT-RTI Based Systems

Indeed, the proposed RT-RTI is a complex software framework that needs to be designed to meet the strictly required time constraints. As we know, standard structural based designs generally involve these steps: requirement definition, architectural and detailed design, coding, and testing [20]. This design method is not well suited for real-time systems because it identifies the timing problems mostly at the testing stage (the last stage in the design circle) or sometimes even after deployment [9].

In a real-time system, time constraints are critical and it is crucial for any timing

problem to be discovered and corrected in the earlier stages of the design circle. Therefore, the traditional design approach may not be well suited for designing such a real time constrained system. Model based design, on the other hand, is a modern methodology that helps to solve the complex dynamic system design problems and issues quickly and effectively. Contrary to traditional structural based system design methods, the model based method uses simulation models to represent real-world systems, which can be pure software, hardware, or embedded systems.

Such models are then simulated under designated conditions that represent the real-world scenarios. The closer these scenarios are to the real world, the better the experimental results tend to be. One of the key advantages of such a design methodology is that critical system design problems can be identified at earlier stages before the costly overhead of realizing the implementation. This methodology also supports design reusability by using a model repository. Another advantage is that the design validation can be done before its realization.

Model based design has been used by many researchers for solving complex design problems. For instance, Schulz and Rozenblit [28] have proposed a novel co-design approach that uses a formal specification language to describe embedded systems. In addition, Hu [29] has proposed a model based methodology to be applied in designing dynamic distributed real-time systems with a particular focus on model continuity. Indeed, model based designs involve using a formal system specification language such as UML-RT [24], Timed Automata [27], DEVS, etc. In general, UML-RT is very suitable for the high-level formal description of a proposed system (software, hardware, embedded system), while Timed Automata and DEVS can be applied for precisely modeling the system components and their interactions, and are therefore ideal for the design and evaluation of real-time dynamic software or embedded systems.

As we know, a RT-RTI system generally needs the participating simulation components to be deterministic, while DEVS formalism is ideal for expressing the deterministic model behaviors to satisfy the requirements of the RT-RTI system because it is able to do low

level system specification in every detail for components behaviors and their interactions in run-time. RT-RTI designers commonly have to face crucial design questions, such as "How could the service components be grouped in order to obtain the best efficiency?" or "Will distributing RTI service components and thread pools over a network be helpful in improving the overall performance?". With the help of a simulation-based modeling formal design approach, such questions can be answered clearly and directly by applying different experimental frames to the design model.

In the following chapters, after the related work literature review, we will propose a novel design approach for a RT-RTI system augmented by an additional formalization using a real-time formal language. This language is Real-Time DEVS (RT-DEVS), a formalism extended from standard DEVS to be used for expressing the real-time static and dynamic system. We follow our design approach according to the formal method and verify our real implementation results by simulating our design model system. We also manipulate the design models to identify some of the key system parameters that could significantly affect the core RT-RTI performance.

Chapter 4

Previous And Related Works

In this chapter, we are concentrating on 2 different aspects of the related literature to the topic being tackled in this thesis. The first one is the Real-Time RTI previous efforts that have worked towards adding this functionality to the HLA standard and to its interface specification. While the second one looks at the verification efforts through the use RT-DEVS formalism. Both sides are not only intended to be demonstrating the previous efforts by themselves, however, they are also pinpointing the differences that granted our designs their unique characteristics.

4.1 Real-Time RTI Based Systems

A lot of work and effort has been invested in enabling real-time functionalities within the Run-time Infrastructure (RTI) [14], [23], [4], [8]. Therefore, in this section, we present a complete picture of the current efforts towards realizing RT-RTI from the core parts of these efforts up until the different general techniques and algorithms in networking and operating systems they applied, to realize their specific design approaches and proposed ideas.

Most of these efforts have concentrated on certain major issues. Frequently, these issues concern the guaranteed network performance through QoS schemes [14], [23], [4] and the scheduling services provided by the underlying operating systems, all with the use of multi-threading asynchronous processing [14], [4], [8]. In other words, they generally focused on the network QoS from the external perspective of a specific federate while guaranteeing the capabilities and resources to be offered within the local perspective of that specific

federate, to complete the job in conformance with the HLA standard including its federate interface specification (RTI). And because each one of the simulation federates will be a specific instance of that design, then there will be no foreseen shortcoming at all. Other efforts aimed towards adjusting and optimizing specific services provided by the RTI [23], [4], such as Data Distribution Management (DDM) [4], and Time Management (TM) [23]. In addition, others have designed a completely new special purpose transmission protocol that extends the HLA standard with the real-time extension [8].

With respect to the utilization of Operating Systems' (OSs) specific characteristics in scheduling and allocation, bounding the response time and requiring all capabilities to meet deadlines have been the most important objectives. The scheduling services provided by OSs are pivotal in processing the different tasks, each with an upcoming deadline. These scheduling services try to improve the predictability factor of the systems. The OSs are not only responsible for scheduling the different tasks with different priorities to meet deadlines, but they are also responsible for allocating different system resources to these tasks during the processing phase. Nevertheless, the consistency of the system should not be affected by multiple accesses at the same point in time. All of these utilizations and optimizations of the system resources are meant to satisfy the time constraints of different tasks. For these OSs, different papers have proposed using preemptive priority scheduling with a global fixed priority scheme [14], [4]. Preemptive scheduling grants processing power to the highest priority task at all times. The global fixed priority scheme assigns all kinds of tasks, some of which may be in the interface specification of HLA with a specific fixed priority that will never change dynamically during run time (as an HLA priority). Thus, there is a scheme for mapping these HLA priorities to OS priorities.

Regarding the guaranteed network performance and its QoS parameters, it is clear that both are essential for time critical applications, especially due to the distributed nature of simulation systems at hand. At this point, there should be guarantees for the end-to-end delay and throughput between the different components. This means that a specific network performance should be dictated with a high assurance rate. According to what

have been proposed in [14], the network QoS approach needed to enable RT-RTI should be both fine-grained and scalable. The three different major QoS approaches referenced in [14] are the Integrated Service (IntServ), Differentiated Service (DiffServ), and Multiple Protocol Label Switching (MPLS). Accordingly, the combination of those approaches would form a hybrid approach to rendering them on different network entities. In this thesis, investigating network specifications is out of scope since our focus is on organizing the RTI with the assumption that the Network QoS is already supported.

In addition, multi-threaded processing paradigms have been proposed to have the highest utilization of systems resources. They are one of the mechanisms used for parallel processing in multi-processor platforms, and are very helpful since they are known to enhance the system performance. These paradigms are also essential to guaranteeing the scalability of systems with more flexibility between different processors.

The authors in [23] proposed a hard real-time HLA-RTI with time-bounded services. These time bounds were visible to simulation developers. They assumed that deadlines should be available to applications. The purpose was to have an environment with deterministic relationships between real-time events in order to reduce spatial and temporal anomalies, while also reducing the overall RTI overhead. Additionally, the authors in [8] proposed a Virtual Reality Transfer Protocol (VRTP) that was designed to meet the Real-time requirements in a distributed environment platform.

We conclude the differences between all Real-time RTI designs in Table 1. This table shows how all performance parameters are taken into account. The three major unique characteristics of our design are Services Processing Differentiation, Automatic Performance Adaptation, and Scalability. For scalability, our design is more flexible in handling more interacting objects (as will be seen in Chapter 7). These characteristics will be clear after the description of the proposed design in Chapter 5.

As we we have seen, none of these approaches take into account the real relationships between the federates' tasks, which are in fact important factors in designing a better RT-RTI system. It is also a key concern for a RT-RTI design to have the capability to predict

the worst-case scenario to help us build a dependable real time distributed simulation system.

4.2 Real-Time DEVS Based Systems

DEVS formalism [6] 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 worth mentioning at this stage that the operation of the discrete event modeled systems is basically represented by a chronological set of events that change the system's state. So, 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.

Furthermore, the extension of DEVS to its real time representative opens up a new area for the specification and verification of real time system design. Real Time DEVS (RT-DEVS) formalism aims to solve the discrete event based system problems incurred from real time, and is able to specify real time distributed systems as DEVS models. The RT-DEVS formalism is defined by Hong [15]. This extended DEVS formalism is as follows(i.e. an atomic RT-DEVS):

Real Time Atomic Model (RTAM) $= <X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta, A, \psi>$

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) | 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,

Y^b: is a set of bags over elements in Y,

ta: time advance function (advances in real-time),

A: set of activities with the constraints,

ψ: S $\Longrightarrow$ A: an activity mapping function.

We can easily see the difference between this new formalism and the standard atomic DEVS in that an activity set A and its associated mapping function ψ are added to the existing atomic DEVS to provide the real time interaction capabilities for DEVS models in their environments. Thus, different model components in a real time system can be encapsulated uniformly by this new formalism. In the following sections, we will see the powerfulness of RT-DEVS in its capability to effectively express and verify our proposed RT-RTI design system.

Table 4.1: **Differences between RT-RTI Designs**

	Proposed Design	Boukerche et. al. [4]	Mclean et. al. [23]	Zhao et. al. [14]	Bruzman et. al.[8]
Services Processing Differentiation	Yes	No	No	No	No
Scalable Design	Yes	No	No	No	No
Auto. Performance Adaptation	Yes	No	No	No	No
Static & Dynamic Load Balancing	Yes	No	No	No	No
ADAPTIVE Multi-threading	Yes	No	No	No	No
Global Scheduling Service	Yes	Yes	No	No	No
Optimized RTI services	No	No	Yes	No	No
Special Protocol	No	No	No	No	Yes
Network QoS	Yes	Yes	Yes	Yes	No

Chapter 5

Proposed Real-time RTI Design and Implementation

In this chapter, we discuss our proposed design from all different perspectives while elaborating all paradigms and strategies used in the load balancing of processing resources and services-dependent handling schemes. We demonstrate the load balancing (LB) schemes introduced as well as showing the different general issues that will be used by our LB schemes like the threads management and the fixed priority issue.

5.1 Overview of RT-RTI Design

After investigating the RTI from all perspectives, we found that neither the RTI nor any previous RT-RTI design has an overall organization paradigm for tasks and available processing resources. This presents a huge shortcoming in the coherence and flexibility between the RTI and its RT support. The overall picture of the RTI components in a distributed simulation system is shown in Figure 5.1.

It is clear that this system should be distributed by nature; however, it is also important for the distributed components to have a framework that formalizes their communication and interaction, which is the RTI in our case of distributed simulation systems.

As mentioned earlier, most of the previous efforts toward realizing the RT-RTI depend on the scheduling and allocation policies provided by the OSs with the usage of simple multi-threaded processing schemes [4]. This makes these efforts primitive and elementary in supporting QoS differentiation when dealing with the tasks of different services within the

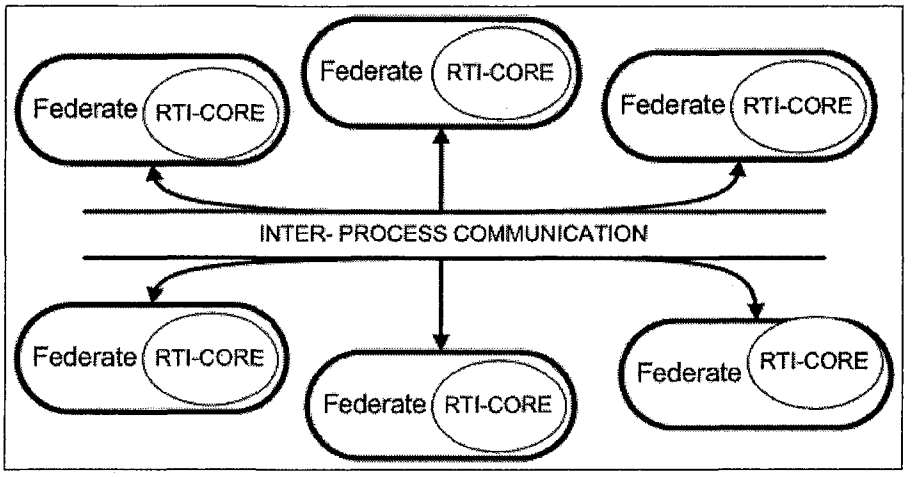

Figure 5.1: **High-level RTI view**

RTI. We believe that having different associations between these tasks and the processing resources available, with some deeper studies of their functionalities at different intervals throughout the federations' lifetime, will be a key issue for flexibility and high RT assurance.

In this section, we present our redesigning process. It is composed of two phases. Both of these phases perfect each other by making the pivotal functionality of the *RTI Core -the part that handles and processes incoming tasks from the different federates-* more organized by dividing processing resources efficiently to achieve the ultimate purpose of a RT-RTI. The second phase can involve more than one option, and we discuss these options in detail.

5.2 RTI-Core Design Approaches

The motivation behind redesigning the RT-RTI was to answer critical questions like, why a consistent advancement in services provided by the RTI is not currently guaranteed by any RT-RTI framework, why the RTI waits until the dispatching phase to know what to do with tasks in different queues, and why should all threads be combined?

The first phase is to confirm separating tasks (or requests) coming to the RTI Core from different federates, each for a specific purpose under a specific category of service. The

separation process is based on the service under which any of these tasks is categorized. We know that certain practical implementations of RTI have divided service queues [23], [4] but this is not always the case for the overall standard of RTI interface specification in [17]. Therefore, the RTI can receive tasks for Data Distribution Management (DDM), Time Management (TM), Federation Management (FM), or any other service. Each one of these tasks will be appended to its respective queue, if and only if the RTI cannot handle it immediately. In other words, having six different services supported by the RTI means that we should have six different queues. Also, It is worth mentioning that the queues used in the design are priority queues, i.e. they have a priority field for each of the entries, and therefore, hand the highest priority task to the processing thread once dispatched. Figure 5.2 shows the different service queues with customized pools assigned to them individually.

In addition, we not only separate the service queues, but also divide the thread pool seen in previous designs into six different pools, each of which is assigned to a specific service queue. These pools are all of the same size except for one. This unique thread pool is larger in terms of the number of threads it has and the resources it can utilize. This initial phase of separation for both the queues and thread pools not only helps in achieving a real-time scheduled scheme, as stated before, but also allows for the possibility of building a well-structured RTI with real-time as one of its characteristics.

The second phase of the redesigning process is the real differentiation in processing tasks. We apply a load balancing strategy for the purpose of assigning the most critical service (i.e. the service queue with the highest level of pressure more than others) at any single point in time larger amount of resources. This way, the proposed design ensures that the most critical service queue receives the resources it needs in order to be predictable and meet deadlines. The largest thread pool is assigned to the needy service queue in either the static way or depending on the closest dynamic relative deadline or priority of tasks within. Meanwhile, the other service queues that have normal thread pools are still producing competent and valuable results within their deadlines. This phase is divided

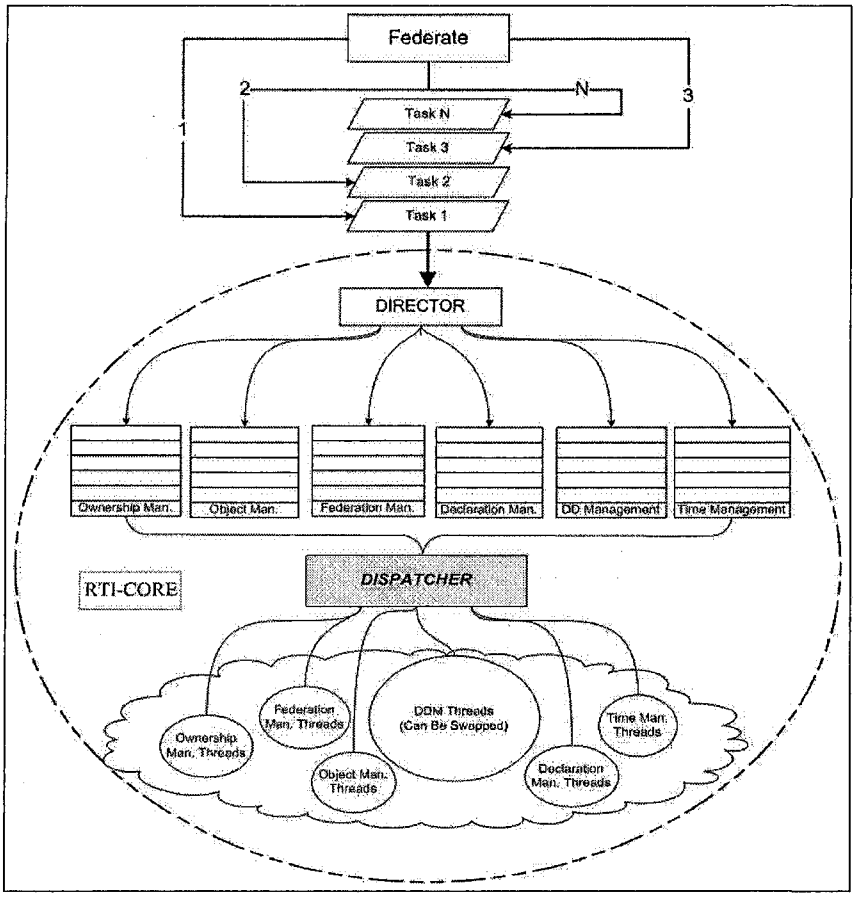

Figure 5.2: **General Design of RT-RTI**

into two different kinds of Load Balancing: Static and Dynamic.

5.2.1 Static Load Balancing Strategy

This strategy depends on the simulation time intervals at which any of the service queues dominate the processing phase. We determine when each one of these services will be at its *Prime Time*. The prime time of a specific service means that this service is the most required service by different simulation federates at a given point in time. In other words, at a specific simulation time interval or wall clock time interval [T, T + Δ T], the prime

time service is the service that will get tasks from different federation components, and the probability of a task being requested for this service is higher than the probability of another service task.

When determining the prime time of a service, we can process it faster than others. In other words, as previously stated, we dedicate more resources to that service queue by assigning it the larger thread pool. The prime time of a service queue signifies that it is very important at a certain point in time, more important than the other services in terms of the pressure it is experiencing.

This load balancing strategy is static because it depends on the number of tasks present in each of the service queues at any point in simulation time. It does not depend on any dynamic parameter, as we will see in the Dynamic Load balancing strategy in Section 5.2.2. We will show the implementation details of both strategies in Section 5.3.1.

This step is important in that it provides a deeper knowledge of the randomness of the simulation systems. This way, we can easily satisfy the most important service queue at a specific time interval because of its importance and the pressure of its tasks.

For example, at the beginning of any simulation system the federation management service is the most requested service, simply because we do not have any joined federates yet. That is, federates are still joining the federation. It is also evident that we should have less pressure on the DDM, DM, and TM service queues during this time interval. Afterwards, when we proceed further and further with the simulation, the federation management service will receive less attention and fewer requests since the joined federates are now interacting with each other through different supported services such as DDM, TM, etc.

5.2.2 Dynamic Load Balancing Strategy

In this section, we describe the dynamic way through which we balance the processing resources between different service queues. We periodically calculate the average deadline of the tasks in all service queues and, depending on the result of the queue with the closest deadline (or highest priority), we assign it the larger thread pool.

We calculate the average deadline of each of the service queues. It is not necessary to have the average deadline of a specific queue as the only deciding factor since we can average the relative priority of all tasks in a queue. However, we would then have to assign the largest pool to the highest relative priority service queue instead of the closest average deadline queue.

These values or parameters can be calculated through recursion, which is the optimum solution for solving such an overlapping problem. We start with one of the tasks and continue the process through the others until we find the pre-assigned base case, as will be seen in Section 5.3.1.

These functions are not rigid, but can be modified to be able to satisfy a greater number of deadlines all over the RT-RTI design. Any function with a low complexity factor and a high deadline satisfaction percentage will be accepted.

The purpose of this overall design is to have the resources management criterion built upon a load balancing strategy. We consider the usage and waste factors of the processing resources, and can see that the usage factor of processing threads in the previous RT-RTI design was lower than what it should have been [4]. The waste factor of resources in previous designs was very high, which contradicted the the purpose of having an efficient real-time RTI system.

It is important to see real-time systems as systems that are highly dependent on resources with load balancing. Thus, it is also important to see that the more efficient and effective our resources management criterion is, the more the real-time system will be realized and gratified.

In addition, the predictable behavior of the RT-RTI with respect to the resources usage factor is also a key element, and thus the division of the thread pool is necessary as well. It cannot be assumed that one service is at its prime time while the other service queues are task-free. Thus, we should have a division of resources among a dynamic system of queues, with the prime time service queue having the largest resource thread pool while the other queues have normal pools to serve them gradually. This is described in detail in

Section 5.3.2.

In our proposed scheme, we are not overriding the different efforts toward having a real-time RTI. Both the scheduling policies to organize the processing of different tasks through priorities and the thread pool mechanisms for concurrent processing, as mentioned in the discussion of previous related work in Chapter 4, are suitable approaches that can be inherited in this proposed design. We utilize every possible part of these concepts, as it is useful for our modified real-time RTI design to process different up and down call tasks exchanged both ways in a predictable fashion.

5.3 Description of RT-RTI Components

As we can see in Figure 5.2, the proposed design is composed of some additional handlers at different layers. In this section, we will describe them to clarify any misunderstanding about their objectives.

The distributed components are communicating with each other via inter-process communication channels. Each one of these have a real-time capable RTI component designed in a way that allows it to serve tasks coming from its own federate or from any of the other remote federate in a call back fashion, as shown in Figure 5.2.

Regarding the modified layout of the task that is sent by federates; the task must have a small header that is informative to accelerate the process of handling it. The header that is added to all tasks is a two-part header. One part declares the priority of the task explicitly. We will describe the fixed priority scheme in Section 5.3.3. In addition, the other part of the header is a small section that defines the service to which this task belongs. It is going to be in a range from 1 to 6, with each number associated with one corresponding service. The header part defining the task's service is called the Service ID (SID). This small section helps in directing tasks to their corresponding service queue. The other important component is called the Director. This component acts as the reception desk that receives customers and directs them to officers according to their need. The director is located

inside the RTI core. It is there to receive the different tasks arriving from local federate or from any remote federate.

It is obvious that the director should be able to access the headers of received tasks easily. This access to the header is intended for two purposes. Firstly, the director can see which service queue this specific task belongs to, and easily forward it there directly without the overhead of extracting the whole task. Secondly, when the director sees this value and decides to send it to a specific queue, there is no need to have this value appended to the header of the task any longer, because all of the tasks in the same queue have the same SID value. In addition, the subsequent accesses to the task are at the service queue level and they, by default, are priority queues. They should be able to access the other section of the header (the priority of the task) easily in order to decide which task is the next to be processed. Thus, all of the information regarding headers and priority schemes should be defined and inserted into each federate in an HLA-compliant simulation system. This should be modified and updated in the HLA standard framework.

5.3.1 Dispatching and Allocation Mechanisms

The Dispatcher component that is mentioned in Figure 5.2 is very critical to our design. This component hands a processing thread to a specific task depending on its priority. We have to discuss two different points; the first is concerned with the regular, simple case of having one task in any queue with a fixed priority waiting to be processed. This task should be handed to its specific thread pool. Therefore, the dispatcher's job for each of the service queues is to find the highest priority task in a queue and hand it to a thread from its pool for processing.

The other important job of the dispatcher is the application of the load balancing strategies mentioned above. There are two different schemes for applying it: one for the static and the other for the dynamic strategy. These schemes decide which service deserves the largest pool (in Figure 5.2, the DDM service queue has the largest pool, and it can be swapped to other service queues).

Regarding the static load balancing scheme, the dispatcher has a parameter at each of the different service queues. Each of these queues has this parameter as a counter for the tasks inside that queue at any point in time. Thus, when a task enters the queue, that queue's counter will be incremented by one, while, when any task is handed out to be processed, the counter will be decremented by one. Therefore, this counter has the actual, exact number of tasks inside its queue.

The counters for each of the queues must be visible to the dispatcher. The dispatcher makes use of this information to assign the large thread pool to one of the queues (the most in need). Thus, when the value of any of the counters reaches a specific, pre-defined threshold, the dispatcher is triggered to assign the large thread pool to this queue instantly and directly.

This threshold value should be adaptive and cannot be constant for all kinds of systems. It is the developer's responsibility to define the specific threshold for his/her application, as it will vary from one system to another.

We can assume a case in which we have two (or more) different queues exceeding the threshold limit. In this scenario, the large thread pool will be assigned to the queue that has the most tasks according to the counter, while the other will have a regular thread pool assigned to it. When the counter of the queue with the large thread pool falls below the threshold, the large pool will then be assigned to the other queue, or to any queue that has a number of tasks greater than the threshold limit or greater than the other queues' counters.

Using this technique, we can ensure that the RTI queues of different services are processed at least in part at their respective prime times. The prime times of the queues are not constant, however: they change regularly, and thus the proposed technique is the most effective for interchanging the large pool between different queues based upon their need.

Regarding the dynamic load balancing, as previously stated, the design will depend on a recursive procedure that works in a special manner as shown in Figure 5.3. One of the greatest advantages of recursion is that it allows the allocation of additional automatic

objects at each function call. This property plays an important role in our design.

Now, it is clear that our purpose is to find out which service queue must use the largest thread pool dynamically: we will depend on a parameter that is the average deadline or average priority of each one of the service queues and, based on that value, we will decide to give the large pool to a specific queue. This algorithm for calculating the average deadline or average priority must be applied periodically, or upon request. This is done by having a global parameter that acts as a sensor to trigger the application of this recursive procedure. We assume this global sensor to be the maximum number of tasks overall the queues. It uses the parameters implemented in the static load-balancing scheme, and is their over all maximum. This way we know that, at any point in time, one of the service queues will already have the larger pool. In addition, when we find that the global sensor is exceeding a specific threshold (which is different from the one defined on the static scheme), this triggers the recursive dynamic balancing scheme to be applied and calculated immediately.

Therefore, the goal now is to calculate the value of the average deadline or priority for each of the queues. We propose a recursive procedure. Its base case would be that all tasks have gone under calculation at each of the queues.

As shown in Figure 5.3, the dynamic load balancing procedure will take each of the task queues as an input. It will fetch the deadline (or priority) of the first task and put it in the average calculator; then, a recursive call to the same function will be made with an input of the same queue with the already calculated task unseen.This process will continue until one of two states is reached. Either all of the tasks in that queue have gone under calculation (the base case) or the value of the average deadline for this service queue is thus far closer than any other service queue deadline. This suggests that we should have a component that continuously checks the values of the average deadlines or (priorities) during the run-time of this recursive procedure and decides instantly which service queue should get the larger pool either according to the closer average deadline or the maximum average priority.

This way, we reduce the overhead that may be caused by this procedure at run time.

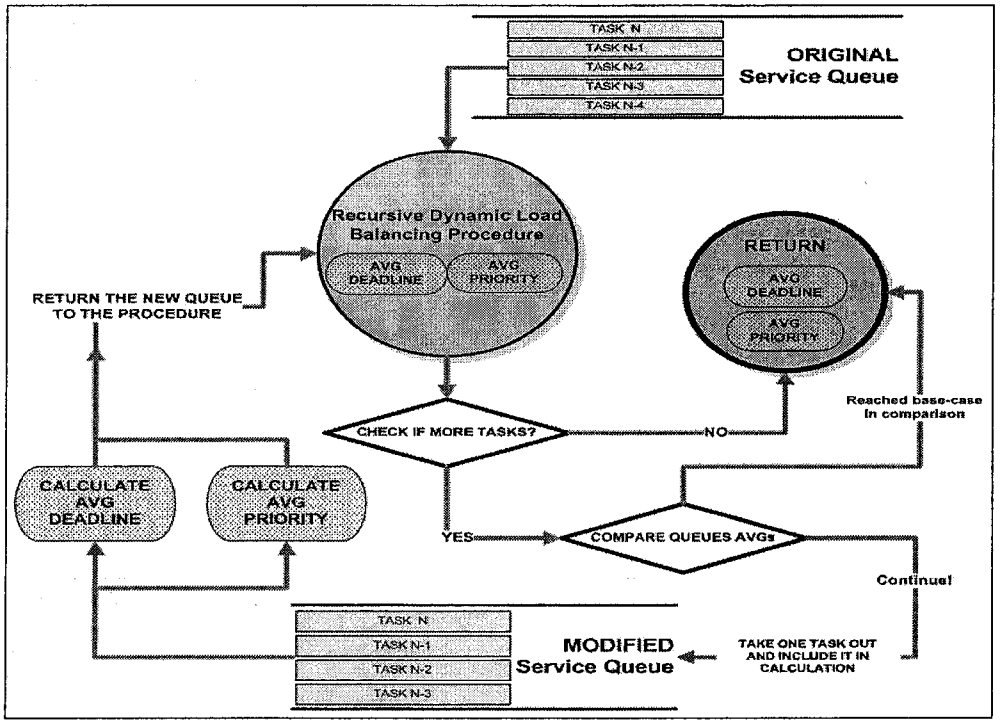

Figure 5.3: **Recursive Dynamic LB Procedure**

We can stop when it becomes clear that we do not need to continue because we have found out which queue should get the large pool.

5.3.2 Threads Management

It is well known that having strict control over the scheduling and execution of processor resources is essential for many fixed priority real-time applications. Predictability should be the regulating relation between the RTI cores and the scattered federates everywhere. In this section, we will explain the characteristics and features that these pools should inherit in order to satisfy the purpose of our proposed design.

It is very important to note that two kinds of threads are needed from the general overview of this design. There is a separate I/O layer of threads that are shared by all service queues. This layer takes care of the tasks arriving and coming out of these queues. This

layer is completely separate from the other kind of threads that are used for application-level task processing.

The thread pools are all of the application-level processing threads and, as mentioned previously, these thread pools are all identical in size except for the single one that has more threads and more resources. In addition, this pool has a few special capabilities that the other pools do not have, including the dynamic property. All the other pools have a static, fixed number of threads assigned at their creation time; the numbers of these threads cannot change after the initial assignment. However, the larger pool is a dynamic one that, while it also has a predefined number of threads working at run time (like the static kind), it can create new threads on-demand to handle new tasks when these tasks arrive and all threads are busy. The maximum number of dynamic threads is also predefined at the time of creation.

Another capability of the thread pools is Thread Borrowing. This allows a pool to borrow free threads from other pools when it reaches the bound of its static and dynamic threads. Thread borrowing is allowed between the normal identical pools. However, it cannot happen between the normal pools and the large pool in a scenario where the normal pool is the requester; it can only take place the other way around.

To allow this, we have included a flag at each of the thread pools except for the large pool. This flag has two values: it either "has free threads", represented as 1, or "no free threads", represented as 0. This flag takes into consideration two issues: the number of tasks waiting in its corresponding queue, and how many free threads it has at that point in time. When this flag is 1, it means that other pools can borrow threads, while 0 means that borrowing is not allowed. Therefore, when a pool needs to borrow a thread, it must check the flags of all thread pools and find the one(s) that can lend it one or more threads.

The last useful capability of the large thread pool is the Lanes property described in Figure 5.4. This large pool has lanes of threads inside. Lanes are subsets of threads, each with a specific priority that is different from any other lane. Therefore, when a task is of a high priority, it should receive a thread with a comparative priority from a lane in the

pool. It is important to know that there is a crucial consequence to having priority schemes for HLA/RTI from one side, and different OSs priorities from the other. For example, the HLA priority scheme that is defined locally at the RTI is independent from the OS priority scheme defined for threads and processes. Consequently, this forces the designer of the real-time RTI distributed simulation systems to have a mapping scheme between the HLA priority and the underlying OS as mentioned in [4], and between different OSs. It is worth mentioning that the thread-borrowing scheme is applied between the different lanes of the large pool as well.

The reason for all these dynamic optimizations is that we need to draw the maximum capabilities out of the processing power we have to serve tasks within deadlines. The point is that each of the queues will ask for the processing power of threads at different times; when each of the queues is at the peak of its processing requests rate (prime time), it should be given the maximum capabilities, and that is the true reason for implementing all of these dynamic functionalities.

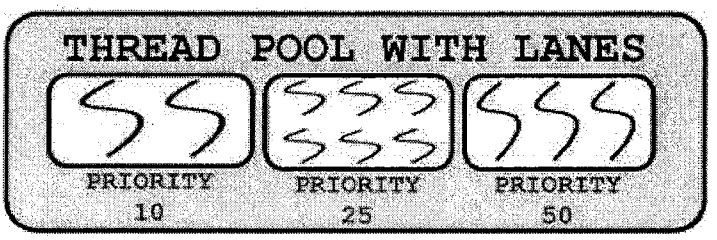

Figure 5.4: **Thread Pool with Lanes**

5.3.3 Fixed Priority Global Scheduling

In this section, we discuss the tasks themselves in terms of how these tasks the way these tasks are assigned priorities and retrieved for processing. It is clear that, at any point, the task with the highest priority will be given the processing resources it needs. Therefore, the priority of these tasks becomes the real specific description of when to process a task. Because of this fact, the fixed priority scheme has been selected.

The fixed priority scheme -as the name implies- is useful in real-time distributed systems because of the heterogeneity of different OSs used everywhere. These different OSs may have different prioritizing schemes at their level, which will definitely cause a conflict between the system divisions and components. This suggests that we should have a fixed HLA priority scheme to overcome this problem. Thus, in this case, a mapping scheme between the OS priority level and the HLA priority level should be provided as suggested in [4].

The global scheduling service is also important in the case of remote requests at foreign components. Unless this scheduling service is global, it is not simple to have remote tasks be served by foreign components because of the priorities locality problem.

This global fixed priority scheme will assign a specific priority to each of the different kinds of tasks. This priority is fixed and static; it will never change after its assignment.

Each federate, when deciding to send a down-call task to the RTI, should refer to the priority scheme it has and assign a fixed priority value to a specific task. After that, it should append this priority value to this task at its location in the header and send it normally to the RTI Core through the pre-mentioned director that directs it to its corresponding queue.

When retrieving tasks from the dispatcher, the Earliest Deadline First (EDF) algorithm is used with a non-preemptive processing policy [29]. This is the best combination for our design, because the overhead could be very effective in the case of a preemptive processing scheme, while very little is gained in this specific case.

Chapter 6

RT-RTI Formalization and Verification using RT-DEVS

In this chapter, we discuss the formalization stages of our RT-RTI design described in Chapter 5. As we know, the design of any software or embedded system is traditionally un-formalized, which often results in low performance and error-prone system architecture that can cause a lot of drawbacks along the life cycle of any system. It is worth mentioning that such un-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, some questions are definitely difficult to answer like: "How can we verify that we have the most optimal system design?", and "Does the designed system have the scalability property within? If so, can we verify the efficiency of this design?".

From another point of view, it is actually impractical to put an un-formalized design in large-scale and dynamic environments since these systems generally require stage-by-stage realizations, verifications, and a final integration. In addition, these systems has to be maintainable as long as they are put in use. Therefore, 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. Besides, the large-scale, distributed and interactive systems have to be very competent in their technologies and in making the maximum potential use of it.

We focus our efforts on illustrating the proposed RT-RTI design by stating the problem clearly, defining the design requirements, and describing the design and all of its compo-

nents, techniques, and strategies. In addition, we use the simulation-based modeling approach (a formal approach for verifying the RT-RTI system) to build the RT-RTI model. In particular, we use RT-DEVS as a formal description language for representing the proposed RT-RTI system. We have realized our design approach in a formalized way along with its real implementation as a RT-RTI system. Our design model components were built using RT-DEVS and represent the real software components in a RT-RTI system. We simulated the design model in a real time fashion under crucial experimental frames to verify the key advantages of our design at all levels.

6.1 Formalization Motivation

Real-Time (RT) RTI is the fundamental core of HLA enabled real-time distributed simulation systems. In particular, the efficiency of processing tasks initiated from joined federates plays a key role in designing an optimized RT-RTI core infrastructure. In the original standard RT-RTI design, all tasks initiated from different federates are handled by service queues, which are then attached to a designated thread pool for tasks processing. We would like to argue that a better RTI core design might be achieved when the core service component in the original RT-RTI is re-designed.

As mentioned above, the standard RTI design is based on simple service queues and a thread pool mechanism [4], while the core of the framework uses scheduling and allocation policies. Furthermore, there is no organization of services inside the RTI framework, which makes it not flexible or scalable. There is also no consideration of load-balancing and the other dynamic factors involved. Therefore, for a real time distributed simulation application, such a RTI framework may not able to handle the time-constrained requirements of a distributed real time system.

6.2 Design Requirements and Verification

Our re-designing process for the standard RTI is based on satisfying the following requirements:

1. Providing a high-performance and adaptive techniques for operating the real-time RTI.

2. Improving the RTI core services' efficiency through better management and utilization schemes for the computing resources used at all points in the simulation time.

3. Applying load-balancing strategies at all layers to the federation structures.

The purpose of the overall design is to create better resources management criteria built on suitable load balancing strategies. The usage factor of the original RT-RTI design is low; therefore, improving the resource management is the key to an optimized RT-RTI design.

In this section, we formalize our design using RT-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.

As shown in Figure 6.1, the proposed RT-RTI design model is composed of "taskGen", "director", "service queue" (such as "DDM", "ObjM" and etc.), "dispatcher", "thread pool", and "task monitor". Each of these model components is described using RT-DEVS formalism and then realized as DEVSJAVA atomic models. We can see that the "dispatcher" model is the core component that was made very simple in previous designs. In the following paragraphs, we present each design model component in detail, and several key model components are presented using RT-DEVS formalism as representatives.

- "taskGen" model:

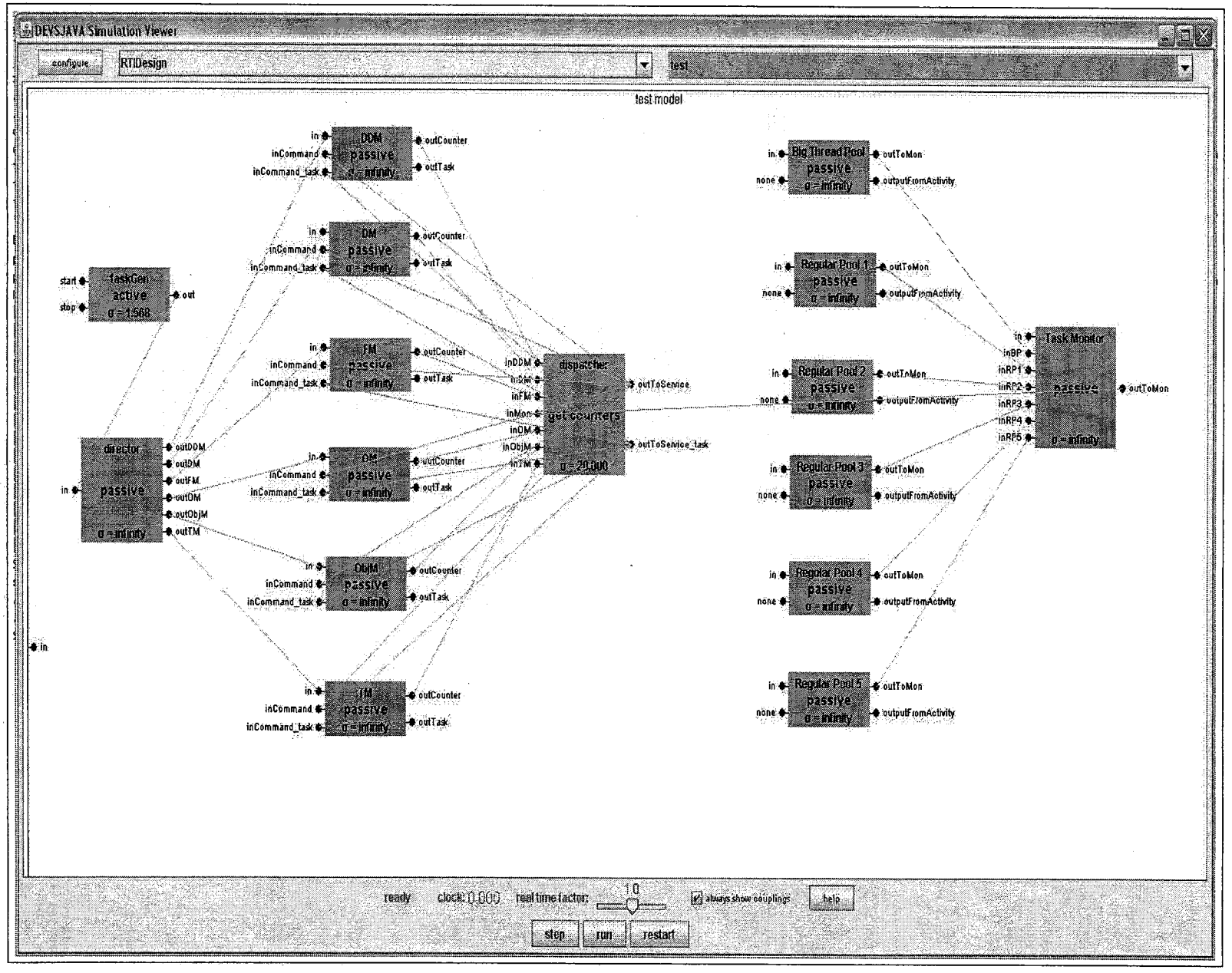

Figure 6.1: **RT-RTI DEVSJAVA Formalism Representation**

The "taskGen" is an atomic DEVS model for generating random tasks, which outputs tasks to the underlying RT-RTI model system at random intervals. The output tasks all have a time stamp for entering the underlying RT-RTI system, a deadline (also a random double value), and an ID (a random integer ranging from 1 to 6). The purpose of generating tasks in this way is to conform to the random nature of generating tasks from federates when a real HLA/RTI system is in action. In other words, the tasks that arrive at any federate in a real HLA/RTI simulation system are either sent from another federate (global) or from the federate itself (local). Therefore, we are making the concept simplified in the way to generate these local and global tasks in such a model. Also, since we are making the concept clear and simple, the ID given to any of these tasks shows the service this task belongs to, assuming that these tasks will be more towards WHAT? not HOW?. The randomly generated tasks are then forwarded to the "director" through the output port of the "taskGen" model.

- "director" model:

  The "director" model is also an atomic model in DEVS that serves as the first stage task handler. It reads the task ID and then sends the tasks to different "service queues" according to their IDs. This model has a zero time delay for outputting tasks to the "service queue" models. This shows our assumption that these tasks will be in their respective service queues immediately after the generation phase, no delay is introduced within the director model. Moreover, as we mentioned before, we are tackling the issue of balancing the processing resources and assuming that the network used within this system is a given delay-bounded network with no specifications introduced.

- "service queue" model:

  This is an atomic DEVS model that is implemented as a priority queue. It orders the incoming tasks according to the deadline of each one, where tasks with shorter deadline come before tasks with longer deadlines. The "service queue" has input

ports for the "director" and "dispatcher", as well as output ports for the "thread pool".

- "thread pool" model:

  This model is implemented as an atomic DEVS, which takes each input task and runs it using a thread. The "thread pool" starts each task activity in real time and outputs the result to the "task monitor" when all tasks are finished in the pool for a certain time interval.

- "task monitor" model:

  This model monitors all the tasks running on all the thread pools, and then sends messages to the "dispatcher" to trigger the next cycle for dispatching tasks from the "service queue" to the "thread pool".

- The "dispatcher" model:

  It is the key tasks control unit for dispatching tasks from the "service queue" to the "thread pools". The "dispatcher" periodically sends out a request to each "service queue" to get its size or average priority on both load balancing strategies Static and Dynamic described in Chapter 5, and then determines how each "service queue" is connected to each of "thread pools". As an example, at a specific point in simulation time, the dispatcher will send a specific job to all different service queues. This job asks each one of the queues to tell about their status depending on the scheme used (static or dynamic). When all queues send their specific statuses back to the dispatcher, it (the dispatcher) starts assigning the thread pools to the respective service queues including the biggest one to the most in need service queue. At the end of the dispatching phase, the most in-need "service queue" is assigned the "bigger thread pool" while other "service queues" are connected to "regular thread pools" each.

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

The "service queue" atomic model is described with RT-DEVS as:

X={"input tasks", "output counter", "output task"}

S={"passive", "outCounter", "outTask"}

Y={"counter", "task"}

δ_{int}\{"passive", "outCounter", "outTask"\} = {"passive"};

δ_{ext}\{"output counter",{"passive", 0 < e < infinite}\} = {"outCounter"};

δ_{ext}\{"output task",{"passive", 0 < e < infinite}\} = {"outTask"};

λ\{"outCounter"\} = {"counter"};

λ\{"outTask"\} = {"task"};

ta("outCounter") = 0 ; ta("outTask") = 0 ;

A = {};

ψ = {};

"Thread Pool" described with RT-DEVS:

X = {"input tasks"}

Y = {"message"}

S = {"passive", "busy"}

δ_{int}\{"passive", "busy"\} = {"passive"};

δ_{ext}\{"input tasks",{"passive", 0 < e < infinite}\} = {"busy"};

λ\{"busy"\} = {"message"};

ta("busy") = constant value ; ta("passive") = infinite;

A = {"create and run task thread"};

ψ\{"busy"\} = {"create and run task thread"};

As we can see from the above description of models in the RT-DEVS formalism, each model component has strictly defined "states", input and output "events", as well as how the model responds to internal and external events. Therefore, the overall system behaviors

can be modeled in whatever resolution we need. Indeed, we are trying to make it representing -closely- the behaviors of our proposed RT-RTI system. Thus, the simulation of such model system is able to predict critical parameters of the proposed RT-RTI system, and can therefore be used to verify our RT-RTI design. In addition, the more closer this model gets to the real implementation of the RT-RTI, the more critical and non-critical parameters will be discovered and adjusted to any other available design alternative. The model continuity issue presented in [29] suggests that any model can -at some point in time- be extended to be the full system only if we go further into details when building the coupled model.

Model continuity refers to the ability of transition -as much as possible- for a model specification through the stages of a development process. It shows how a modeling and simulation environment, based on the discrete event system specification formalism DEVS, can support model continuity in the design of dynamic distributed real-time systems. In [29] the authors restricted such continuity to the models that implement the systems real-time control and dynamic reconfiguration. The proposed methodology supported systematic modeling of dynamic systems and adopts simulation-based tests for distributed real-time software. If we would apply the model continuity concept over our proposed RT-RTI design, then we have to design control models of a dynamic distributed RT-RTI system, analyze, and test them by the simulation methods, all to smoothly apply the transition from simulation to distributed execution. We would suggest the application of the model continuity concept to the RT-RTI system as a core future work improvement.

In the following Chapter, we will present some precisely designed simulation experiments that aim to test and verify the advantages of the proposed new RT-RTI design when compared with the standard RT-RTI design. Our particular focus is on looking at how dynamic thread pool management can benefit the performance of the RT-RTI system in terms of the served tasks that meet their deadlines. We also examine carefully the load-balance of the served tasks for different services provided by a RT-RTI system.

Chapter 7

Performance Evaluation of RT-RTI

In this chapter, we present our experiments and discuss the results we have obtained from 2 different perspectives. The first is the real implementation experiments that evaluate the performance of our proposed RT-RTI described above in Chapter 5. It shows the design proposed in action with all different parameters and algorithms put in use within a real infantry training simulation system. While the other presents the verification results of our RT-DEVS formalism criteria described in Chapter 6. The real implementation experiments are of two different types, the first of which is an analytical performance evaluation and comparison of the queuing models mentioned in [14], [4] with ours. This demonstrates the proof-of-concept for the queueing model we have chosen in our implementation algorithms. This evaluation experiment randomly assigns values to the queuing system parameters depending on distribution probabilities. This is because everything is random in nature and there is no way to anticipate how the federates of a simulation system may act or react to any stimuli. For this experiment, we used Microsoft Excel to build a model of the queuing systems used previously. In addition, we compared it with the proposed queuing model used in this paper. We showed analytically and mathematically that the proposed queuing model is better than previous models [14], [4] by using the conventions and theories of Queuing Systems described in [12]. The second experiment is the simulation of the whole system, with all specifications and design paradigms mentioned before under examination. The results are very promising, as will be seen in following sections.

7.1 Real Implementation Experimental Results

In this section, we provide a thorough description of our real implementation results obtained after building the RT-RTI proposed design in Chapter 5. It is divided into 2 subsections; an Analytical performance evaluation and a Simulation experiments of an infantry training operation.

7.1.1 Analytical Performance Evaluation

In this section, we prove analytically that by having tasks inserted into different service queues and their processing threads in separated pools, the system performance will be enhanced. We used formulas from the Queuing Theory field [12] built upon a Microsoft Excel simulation spreadsheet. These formulas describe precisely what are the different cases and schemes that can be used in such a RT-RTI design. The simulation generates random numbers for each one of the simulation trials (50'000 Trials) depending on the probability distribution chosen.

In fact, the type of queues used by the previous RT-RTI designs is the same as the type of queues we propose to use in our RT-RTI design. These queues are M/M/n queues. The first M represents the probability distribution of tasks' arrival into the internal RTI and is categorized under the *poisson distribution* probability function. The second M in the queue description means that the probability distribution of service times for these tasks is exponentially distributed. The last n defines how many servers (threads) there are to process the tasks waiting in the input queue.

In a poisson stream, successive tasks arrive after intervals whose lengths of time are distributed independently and exponentially. It is important to know that tasks arriving in a poisson order have an exponential inter-arrival times as proved in [12]. The difference between the exponential and the poisson distribution is that the exponential one is continuous while the poisson is discrete. The exponential distribution is often used to model the time between independent events (tasks) that happen at a constant average rate(λ).

The simulation experiment is a visualization of the different cases that could appear while having a large number of non-constant parameters with no specific pattern to expect them. A huge number of trails with those variables changing at every trial, all to see what can be inferred about the results and the rates of different kinds. These simulation results are compared between the previous scheme [4] and our proposed scheme. After those comparisons, it became obvious and clear that this scheme is much more useful that the other.

The simulation software we used tries to take the assumptions we provide upon the probability distributions. After a large number of trials based on those assumptions and correlations, the results are provided with respect to the forecasts and decisions dictated. These assumptions are built upon the equation of the Poisson distribution in Equation 7.1

$$f(k,\lambda) = \frac{e^{-\lambda}.\lambda^{k}}{k!} \tag{7.1}$$

Where k is the probability that is given by the whole function, λ is a positive real number, equal to the expected number of occurrences of a task arrival in a given interval (mean arrival rate). Poisson distribution is a discrete probability function that is applicable when entities arrive at random. These entities should be independent from each other; it is exactly the case in our tasks' arrival to the RT-RTI queuing system.

In fact, if we can obtain proof that the mean service rate (μ) of the proposed queuing system μ_{new} is greater than the mean service rate of the previous design μ_{old} ($\mu_{new} > \mu_{old}$), then we can claim that it is better. It is important to remember that simulation experiments depend on assumptions and a large number of trials. The results acquired are all mentioned, to assure that the proposed queuing model is better than the previous designs' models in the interface specification standard [17] and in [14], [4].

Therefore, two models were built to run these experiments. The first model is the One_Queue_Model while the second is the Six_Queues_Model. The first represents the old design while the second one represents our proposed design with the division of the input queues with a thread pool dedicated to each one of them. The experiment used a Monte-

Carlo simulation with an initial random seed of 999 and a precision control confidence level of 95%. Furthermore, we had about 168 assumptions including the queuing models with the multiple thread pools for each of the queues on all models. Those assumptions are defining the probability distribution functions working together in forming the queuing model. These probability distribution functions are either poisson distribution functions in discrete event representations or exponential distribution in continuous event representations.

After building the models, we calculated the mean arrival rate (λ) and the mean service rate (μ) of the two models. To ensure the correctness of the results for comparison purposes, the mean arrival rate in both models was fixed, which means that (λ_{new}=λ_{old}). This is to ensure that any difference in the mean service rate is correct with no calculation flaws. The mean service rate was calculated by dividing the number of tasks served at all times by the number of time units simulated.The simulation ran about 50'000 trials with the assumptions and correlations previously described. The results are shown in Figure 7.1 and Figure 7.2.

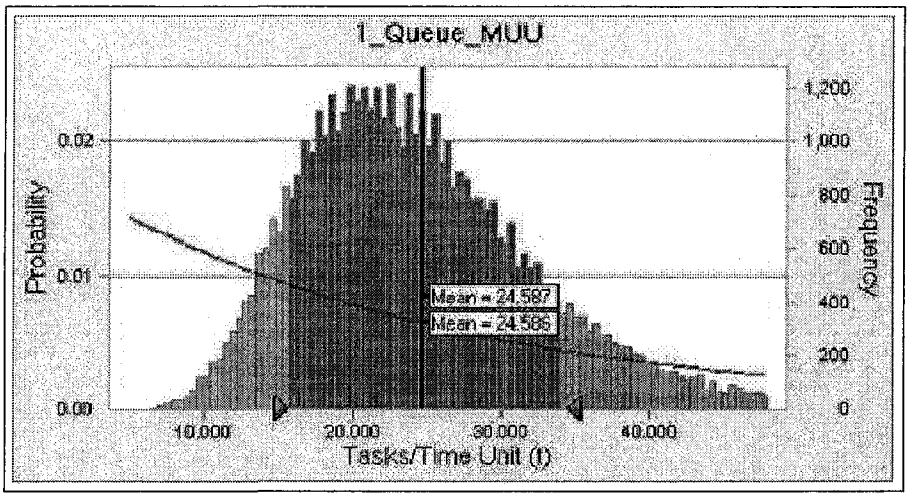

Figure 7.1: **1_Queue_Model service rate (μ)**

In these figures, we can see that after running all trials, the mean service rate of the One_Queue_Model (μ_{old}) is 24.580 Tasks/Time Unit (t) served within their pre-dictated

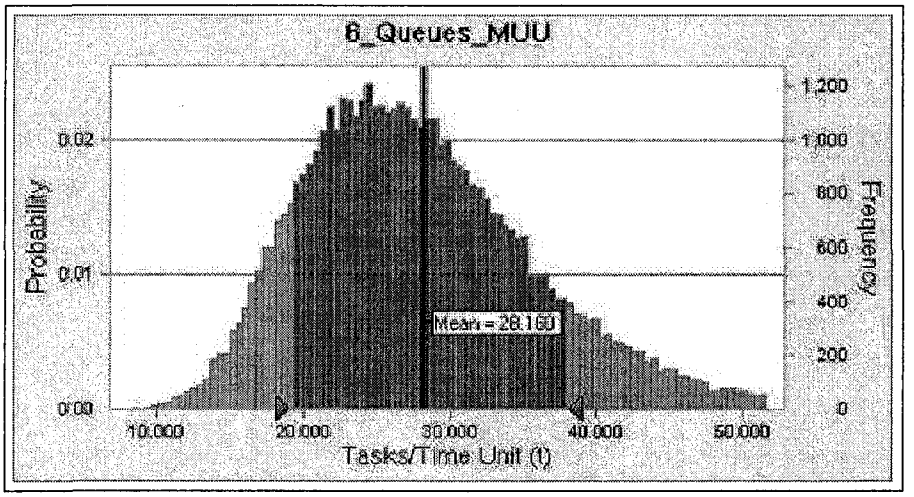

Figure 7.2: **6_Queues_Model service rate (μ)**

deadlines, while in the other Six_Queues_Model, the mean service rate(μ_{new}) is 28.160 Tasks/Time Unit (t). The difference is at least about 3.5 4 Tasks/Time Unit (t) served within their deadlines, or approximately a 15% improvement. The mean standard error for both models was constant (0.037). It is important to know that the numbers used to run the simulation were all randomly generated (more than 10 M numbers during 50'000 trials).

In addition, we combined the results of the two different model service rates. We created a relative value between the two rates, also applying the mean arrival rate (λ). This relative parameter is (Ω) and equals μ/λ. We determined that the mean of Ω values in the Six_Queues_Model is about 0.96345, while in the One_Queue_Model it is 0.81413. This means that the Six_Queues_Model's μ is higher than the other model's μ, and these calculations satisfy our purpose of having $\mu_{new} > \mu_{old}$. Therefore, the new proposed RT-RTI design outperforms the previous simple design only by the division of processing resources.

7.1.2 Simulation Experiments

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 previously, and the results were as promising as we predicted.

From what was mentioned earlier, our goal is to have a well-structured and reliable real-time RTI system. However, a tiny overhead has been noticed by some authors through what have been seen as complications added to the previous simple designs [14], [23], [4]. The overhead that was noticed and is supported by all of our experiments, can be easily justified as the price of a real-time system. We can never have a real-time system without incurring a little overhead, since this overhead increases the reliability of our system from one side while enhancing the predictability of the system from the other side in a direct manner. This predictability is a way to ensure the efficient processing of tasks coming through up and down RTI calls within their deadlines.

From the results of our experiments, the system's predictability has been enhanced by about 52% in terms of processing tasks within their deadlines when compared to the previous designs [4], as shown in Figure 7.3. However, the overhead noticed was less than 1%, as shown in Figure 7.4, in terms of the time of the whole simulation from the creation of a federation execution until the resignation of the last federate. It is important to note that this overhead affected only the dispatching time of tasks, in that the tasks are handed more slowly to their processors' threads. This overhead has not affected the processing time of these tasks at all, as it is out of the design's scope. The increase in the dispatching time of tasks is the factor that affected the increase in the whole simulation time.

Moreover, we have noticed another major advantage of our design, which is the scalability of the system. It is a vital case when more federates are joining and all of them are interacting with each other at a time when all tasks have deadlines, and all of them are asking for these deadlines to be satisfied. We have seen that when more federates join the federation and more tasks travel between them, the system performance remains the same. Yet, much better performance has been noted when compared to previous un-managed

simple paradigms [4], with scheduling policies and multi-threaded processing resources.

Practically, it is also vital to know where any kind of parallelism can be applied. It is not always the case that we can have parallel tasks processing. The dependency issues are seen as borders, where no parallelism is allowed [21]. This is a consistency problem, where the correctness of the system depends on the sequence of processing tasks, not on processing them in a parallel manner. On the other hand, parallelism can be very efficient in some cases, especially when dealing with multiple services. As mentioned before, all services are progressing at the same time. No service is left ignored, but all of these tasks are independent from each other.

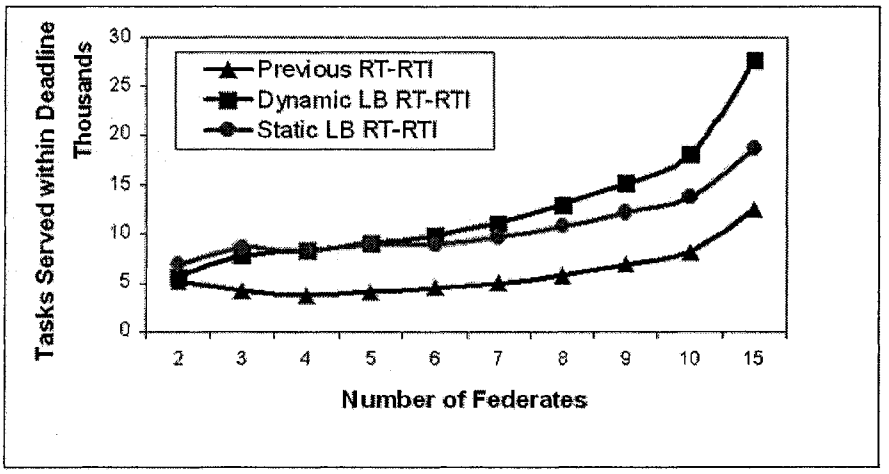

Figure 7.3: **RT-RTI Designs Comparison**

The main objective of our simulation experiments was to find the amount of tasks served within their deadlines and compare it with the previous results from the RT-RTI design in [4]. These 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.

Each of the federates ran on one of the cluster nodes. Interaction between federates

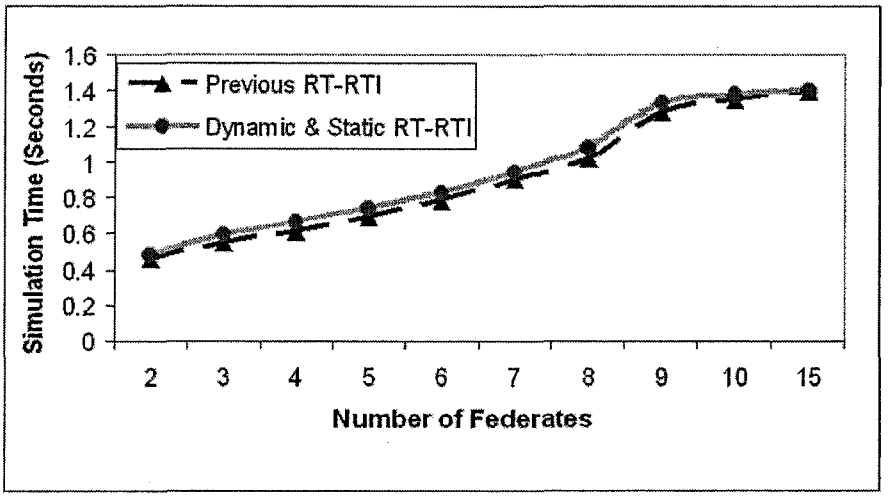

Figure 7.4: **Simulation Time Overhead**

is done through the inter-process communication channels available. The results shown in Figure 7.3 and Figure 7.4 are the average of the results obtained from our experiments. Our experiments simulated an infantry training operation that involved tanks of different types managed by different teams interacting with each other and exchanging critical data in a real-time fashion. Each of the tasks exchanged had a priority depending on its importance and criticality. These experiments were well-suited for proving the RT-RTI system predictability all over.

The number of federates that joined the infantry training operation ranged from 2 to 15 federates. Each of these federates led and directed a specific number of objects (tanks) in the common routing space. while the range of objects (tanks) for each of these federates ranged from 32 to 500 objects (tanks) for each federate. In addition, the number of cells in our routing space ranged from 100 to 1000 cells.

These experiments justify what has been claimed before about the trustworthiness of our proposed RT-RTI design. In addition, all of these experiments ran both schemes: static and dynamic load balancing. Hence, as seen in Figure 7.3, we compare the previous RT-RTI design in [4] with our proposed design for both static and dynamic load balancing strategies,

in terms of how many tasks are served within their deadlines in both RT-RTI designs. It is clear that the proposed design strategies are more efficient than those procedures mentioned in [4]. However, as mentioned before, we are enhancing the way to manage these resources for any future optimization from within the RTI itself.

Furthermore, Figure 7.4 shows the difference regarding the overhead in the total simulation time between different designs. This figure shows the very tiny overhead that increased the simulation time, which we see as the very acceptable price of a real-time system. The overall simulation time increased by less than 1%. Thus, we are adding multiple stages of assurance for preserving the predictability of the system.

7.2 RT-DEVS Formalism Experimental Results

In this section, we follow the experimental frame concepts commonly used in DEVS based modeling and simulation. We analyze our RT-RTI design using previously defined RT-DEVS models, and we focus our experiments on the effects of dynamic thread pool management and the load balancing of service components of an RTI system. For the following simulation experiments, the design models are simulated in a DEVSJAVA environment using a real-time fashion as described in Chapter 6. Our simulation experiments are designed to verify the key aspects of the experimental results obtained from our real implementation of the proposed RT-RTI system. Therefore, the following simulation experiments presented will focus on two of the most important characteristics of our new RT-RTI design: dynamic computing resources management and the load balancing of each RT-RTI core service.

It should be noted that, for the following simulation experiments, the "dispatcher" model is initially set to wait for a certain amount of time before sending its first request for receiving the queue size of each "service queue", and then send the request periodically when receiving input from the "task monitor". It is worth mentioning here also that the linkages between the "service queue" and "thread pool" are totally dynamic according to the run-time gathered queue size parameters, 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. After the linkage (or "coupling", in DEVS) is built, the tasks are transported to the "thread pool", where a set of threads is created for all incoming tasks at a given point in time. The "task monitor" is used to monitor the tasks' executions in the thread pools, and it sends a message to the "dispatcher" when all the tasks have finished. Then, the next cycle begins and continues to take out the tasks out of "service queue" and forward them to the "thread pool".

7.2.1 Dynamic Thread Pool Management in a Modeled RT-RTI System

In this simulation experiment, we study how dynamic controlled thread pools can affect the overall RT-RTI system performance in terms of the percentage of tasks that are meeting their deadlines. The dynamic thread pool management is realized in RT-DEVS models by calculating the size of the "service queue" during run-time, and it then determines the size of each "regular thread pool" and the "bigger thread pool". In this experiment, a simple mechanism is used to dynamically determine and control the size of each "thread pool". That is, the size of a "regular thread pool" is equal to a quarter of the average of the all "service queue" sizes, while the size of "bigger thread pool" is equal to a quarter of the maximum of the "service queue" size among all " service queues". As experimental frame parameters, for the original RT-RTI design model [4] , a fixed thread pool size of 20 is used for serving the incoming tasks from the service queue. For the proposed design model, on the other hand, the initial thread pool size of the "bigger thread pool" is set to 5, and the other 5 "regular thread pool" sizes is set to 3 for each; therefore, the total initial thread pool size of all pools is also equal to 20. However, in this test case, the aforementioned dynamic thread pool management mechanism is used for the new design, and this mechanism changes the size of the thread pool according to the run-time queue

size of the "service queues". In this experiment, we measure the percentage of the tasks that meet their deadlines after they enter the RT-RTI model system.

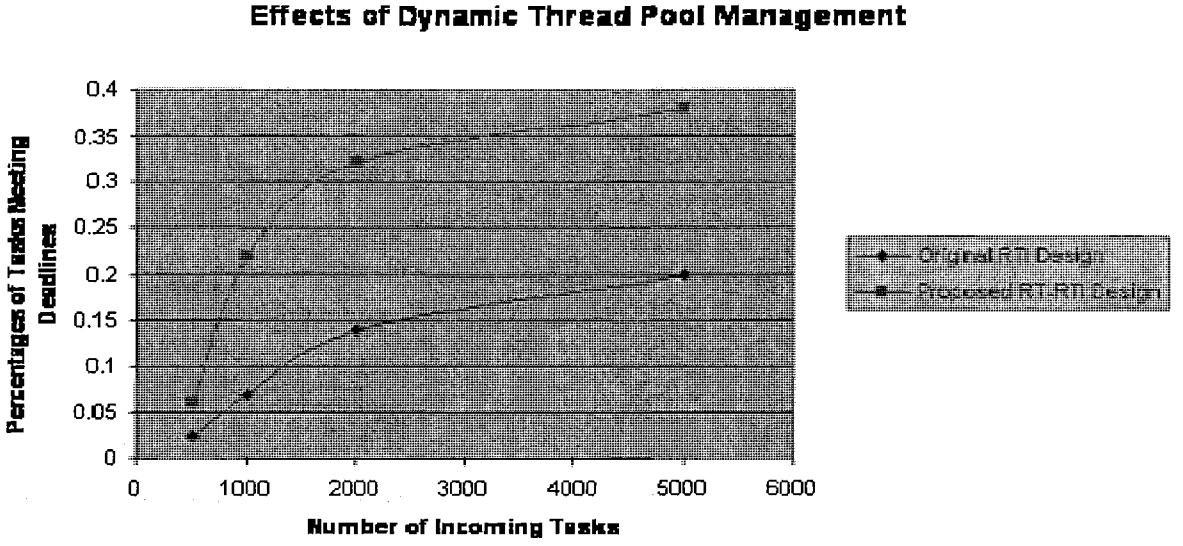

Figure 7.5: **Effects of Dynamic Thread Pool Management**

As shown in Figure 7.5, the simulation result shows a significant improvement in the percentages of the tasks being served within their deadlines when dynamic thread pool management is used. This result directly verifies the result we obtained from the real implementation of our RT-RTI design. The designated simulation result also indicates that thread pool management is one of the keys in improving the RT-RTI core real time capability.

As we have seen in Figure 7.3, for the experiment conducted in the real implementation of our proposed RT-RTI system, the number of tasks meeting their deadlines is significantly increased when static and dynamic load-balancing strategies are used. In fact, both of the strategies focus on managing the computing resources in a more efficient way. The simulation experiment shown above Figure 7.5 applied a generic strategy similar to the dynamic load-balancing of thread pools used in the real implementation, and our simulation result

did verify the significant advantage of using this kind of dynamic thread pool management in our proposed RT-RTI design. Note that our simulation experiment is designed for the purpose of comparing our proposed new RT-RTI design with the original one, so the absolute values of "percentage of tasks meeting their deadline" only depends on the used experimental parameters such as random interval for generating tasks, total number of threads, and etc.. Therefore, the comparison is actually the key when this simulation result is used for the verification of our proposed design.

7.2.2 Load Balance of Services in the Modeled RT-RTI System

It is easy to see that the new design considers the load-balance of all service components using separated service queues. While considering the random nature of incoming tasks from federates, it is not so obvious that using separated service queues will benefit the overall load balance of the served tasks requesting different services from the RT-RTI system. In this experiment, however, we continuously manipulate our modeled RT-RTI system, and carefully look into the load balance characteristic of the new design compared to the standard design.

In the following experiment, we intentionally injected non-uniform distributed tasks from our "taskGen" model, and then measure the percentage of the tasks that meet their deadlines for each task category (according to the requesting service for the task). In this test, we in fact increased the number of tasks for task 1 (which requests Object Management Service) from the "task generator" to make the incoming tasks for different service categories in the RT-RTI model system imbalanced. We then measured the processing capability of the two-design system in terms of the percentage of tasks meeting their deadline for each task category. Table 7.1 shows the simulation results we obtained. We can see that the proposed design in each task category has a significantly higher percentage for tasks meeting their deadlines, and that the percentage difference of "service queue 1" with the average of the other service queues is significantly smaller for our proposed design. This means that the proposed design has a better load-balancing capability for non-uniformed

Table 7.1: **Deadlines Satisfaction Percentage of RT-RTI Designs**

Number of Tasks	S.Q. 1	S.Q. 2	S.Q. 3	S.Q. 4	S.Q. 5	S.Q. 6	AVG(SQs) - SQ 1
1000							
Original Design	0.1	0.02	0.038	0.02	0.037	0.04	0.07
Proposed Design	0.22	0.35	0.19	0.25	0.2	0.21	0.02
2000							
Original Design	0.16	0.08	0.07	0.08	0.03	0.09	0.09
Proposed Design	0.32	0.37	0.24	0.23	0.36	0.34	0.01

distributed tasks that enter the RT-RTI core system.

This experimental result further verifies our proposed design in terms of the load-balancing of each core service in the RT-RTI infrastructure. The result here also conforms to our real implementation of the proposed design, because each service has a better real-time capability that results in the overall advantage of our new design in terms of a greater percentage of tasks being served within their deadlines. At the present time, we can see clearly that the services provided by the core service components are better balanced when non-uniformed tasks are injected into our proposed design system. In fact, in a real HLA/RTI system, we cannot expect tasks to be uniformly distributed from the federates to the RTI core; therefore, this service load-balancing capability is an important characteristic and advantage of our proposed RT-RTI design.

In this section, we have demonstrated the powerfulness of the simulation based modeling design approach for a novel RT-RTI system. The RT-DEVS model simulated in DEVSJAVA provides the key design parameters for predicting the crucial performance characteristic of the real RT-RTI system. Also, the results from the RT-DEVS simulation verified the experimental results from the real realization of the design in a formal and strict manner. It is also necessary to mention that the simulation experiments used here only examine some of the key interests for our RT-RTI design, and further more experiments can be done by changing the experimental frames. The purpose of the above simulation

experiments is to verify our design through formalized simulation based modeling using RT-DEVS.

Chapter 8

Conclusion and Future Work

In this chapter, we summarize our contributions toward the Real-Time RTI infrastructure. It is obvious that the need for a real-time extension appended to the RTI and HLA is of great importance to many application fields. We start by a summary of our contributions and then we show what would be a useful route for future efforts in this subject.

8.1 Summary of Contributions

In this thesis, we have proposed an adaptive and efficient real-time RTI architecture, a novel approach that enhances HLA-RTI based large-scale distributed interactive simulation systems. The proposed design architecture can be suitable for any real-time system, as it characterizes a successful and competent structure of these systems. We have reported on a set of simulation experiments that evaluated our architecture and compared it with other designs [4]. The results showed an improvement of 52% in the number of tasks being served within deadlines.

In addition, we presented a RT-DEVS based formal language approach for our RT-RTI system design verification. We have used simulation based modeling for the RT-RTI design and, in particular, used RT-DEVS to formalize our design as a real-time responded discrete event model system. We then measured the key design related parameters by simulating our design models in different scenarios. Our simulation experiments predict some of the key advantages of our proposed new RT-RTI design, and the results conform to the experiments we took in a real representation of the RT-RTI system. We have found some of the key points that should be addressed in designing and implementing a RT-RTI

system, and such key points are obtained in a much easier way by using the simulation based modeling approach with RT-DEVS.

As a summary, through our simulation based modeling approach for the RT-RTI system design, we have made these findings:

1. Dynamic load-balance based thread pool management is one of the keys to the significant improvement of the real time capability of the RT-RTI core.

2. Differentiating service components in the RT-RTI core is an efficient way to maintain the balance for serving each core service in a timely manner as required by the RT-RTI system.

3. The model based formal approach is able to help with finding the optimal design for a RT-RTI system.

8.2 Future Work

In the future, we will focus on reducing the overhead that was seen in our experiments as much as possible. In addition, we would further develop our RT-DEVS design model in more depth to tackle the issue of model continuity for the next level of RT-RTI design and implementation. In such an approach, RT-DEVS models implement the RTI interfaces and therefore have the ability to function as a real RTI system, not just as a model for the testing and verification of a RT-RTI design. Designing the RT-RTI system as a distributed infrastructure is also an interesting topic that is worth investigating.

References

[1] A. Boukerche , S. K. Das , A. Fabbri , O. Yildiz, "Exploiting model independence for parallel PCS network simulation" In Proceedings of the 13th workshop on Parallel and distributed simulation, Pages: 166-173.

[2] A. Boukerche and Carl Tropper, "A static partitioning and mapping algorithm for conservative parallel simulations". In Proceedings of IEEE/ACM PADS 1994, Pages: 164 -172.

[3] A. Boukerche, A. Roy: Dynamic Grid-Based Approach to Data Distribution Management. J. Parallel Distrib. Comput. 62(3): 366-392 (2002)

[4] A. Boukerche, K. Lu: Design and Performance Evaluation of a Real-lime RTI Infrastructure for Large-Scale Distributed Simulations. DS-RT 2005: 203-212

[5] A. Boukerche, S. K. Das, "Dynamic Load Balancing Strategies for Conservative Parallel Simulations" In Proceedings of IEEE/ACM PADS 1997, Pages: 20-17.

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

[7] 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

[8] Bruzman, M. Zyda, K. Watsen, and M. Macedonia. "Virtual Reality Transfer Protocol (VRTP) Design Rational". In Proceedings of Sixth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Distributed System Aspects of Sharing a Virtual Reality, pages 179-186, June 1997.

[9] BURNS, A. AND WELLINGS, A. J. 1994. HRT-HOOD: A structured design method for hard real-time systems. Real-Time Syst. 6, 1 (Jan. 1994), 73-114.

[10] C. Gill, D. Levine, D. C. Schmidt, and F. Kuhns, "The Design and Performance of a Real-Time CORBA Scheduling Service," International Journal of Time-Critical Computing Systems special issue on Real-Time Middleware, 1999.

[11] "DMSO HLA Official Website". https://www.dmso.mil/public/transition/hla/.

[12] D Gross and CM Harris: "Fundamentals of Queuing Theory" John Wiley & Sons, Inc. New York, NY, USA, 1985.

[13] DoD Defense Modeling and Simulation Office. In "HLA Run-Time Infrastructure RTI 1.3-Next Generation Programmer's Guide Version 5", May 2002.

[14] H. Zhao and N. D. Georganas. "HLA Real-time Extension". In Proceedings of Fifth International Workshop on Distributed Simulations and Real-Time Applications, pages 12-21, 2001.

[15] Hong, J. S. and T. G. Kim (1997). "Real-time Discrete Event System Specification Formalism for Seamless Real-time Software Development." Discrete Event Dynamic Systems:Theory and Applications, 7: 355-375.

[16] Hyman, A. Lazar, and G. Pacifici, "Real-time scheduling with quality of service constraints, " IEEE J. Select. Areas Commun., vol 9, pp 1052-1063, Sept. 1991.

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

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

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

[20] Kangsun Lee , Paul A. Fishwick, OOPM/RT: a multimodeling methodology for real-time simulation, ACM Transactions on Modeling and Computer Simulation (TOMACS), v.9 n.2, p.141-170, April, 1999

[21] Kirner, A. Davis: Requirements specification of real-time systems: temporal parameters and timing-constraints. Information & Software Technology 38(12): 735-741 (1996)

[22] M. Kargahi and A. Movaghar, "Non-preemptive earliest-deadline-first scheduling policy: a performance study". In Proceedings of 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005, Pages: 201-208.

[23] McLean, R. M. Fujimoto, and B. Fitzgibbons. "Middle-ware for Real-Time Distributed Simulations". Journal of Concurrency and Computation: Practice and Experience, 16(15):1483-1501, November 2004.

[24] OMG. UML 2.0 Superstructure Specification, OMG Adopted Specification, August 2003.

[25] Pyarali, M. Spivak, D. C. Schmidt, and R. Cytron, Evaluating and Optimizing Thread-Pool Strategies for Real-Time CORBA, proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM '01) in Snowbird, Utah, June 18, 2001.

[26] "Real-Time CORBA 2.0: Dynamic Scheduling Specification", http://www.omg.org/ , November 2003.

[27] R. Alur and D.L. Dill, "A Theory of Timed Automata", Theoretical Computer Science 126, 183-235, 1994.

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

[29] X. Hu, and B.P. Zeigler, "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, pp. 867- 878, November, 2005.