Winter 2000

An Architectural Framework for Performance Analysis: Supporting the Design, Configuration, and Control of DIS /HLA Simulations

David B. Cavitt
Old Dominion University

AN ARCHITECTURAL FRAMEWORK FOR PERFORMANCE ANALYSIS: SUPPORTING THE DESIGN, CONFIGURATION, AND CONTROL OF DIS/HLA SIMULATIONS

by

David B. Cavitt
B.S. June 1989, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
December 2000

ABSTRACT

AN ARCHITECTURAL FRAMEWORK FOR PERFORMANCE ANALYSIS: SUPPORTING THE DESIGN, CONFIGURATION, AND CONTROL OF DIS/HLA SIMULATIONS

David B. Cavitt
Old Dominion University, 2000
Director: Dr. C. Michael Overstreet

Technology advances are providing greater capabilities for most distributed computing environments. However, the advances in capabilities are paralleled by progressively increasing amounts of system complexity. In many instances, this complexity can lead to a lack of understanding regarding bottlenecks in run-time performance of distributed applications. This is especially true in the domain of distributed simulations where a myriad of enabling technologies are used as building blocks to provide large-scale, geographically disperse, dynamic virtual worlds. Persons responsible for the design, configuration, and control of distributed simulations need to understand the impact of decisions made regarding the allocation and use of the logical and physical resources that comprise a distributed simulation environment and how they effect run-time performance. Distributed Interactive Simulation (DIS) and High Level Architecture (HLA) simulation applications historically provide some of the most demanding distributed computing environments in terms of performance, and as such have a justified need for performance information sufficient to support decision-makers trying to improve system behavior.

This research addresses two fundamental questions: 1) Is there an analysis framework suitable for characterizing DIS and HLA simulation performance? and 2) what kind of mechanism can be used to adequately monitor, measure, and collect performance data to support different performance analysis objectives for DIS and HLA simulations? This thesis presents a unified, architectural framework for DIS and HLA simulations, provides details on a performance monitoring system, and shows its effectiveness through a series of use cases that include practical applications of the framework to support real-world U.S. Department of Defense (DoD) programs. The thesis also discusses the robustness of the constructed framework and its applicability to performance analysis of more general distributed computing applications.

This thesis is dedicated to Ellen; my wife, partner, and best friend. Her constant and tireless intention makes this thesis as much hers, as it is mine.

XOXOXO

ACKNOWLEDGEMENTS

First and foremost, I must thank my principal advisor, Mike Overstreet. During this work, he kept me pointed in the right direction, patiently stood by when productivity waned, and helped to see this dissertation was brought to fruition. The value of his experience and guidance to me has been immeasurable. Similar thanks go to Kurt Maly, my co-advisor, whose experience and insight have always been right on the mark. Additional thanks go to Dick Nance, Ravi Mukkamula, and Bowen Loftin for their outstanding guidance and critical reviews during my research. Their clear and concise viewpoints, and experience significantly enhanced the quality of this effectuation.

Thanks also go to Ed Harvey and Jack McGinn, for their continual encouragement throughout my research and writing. Ed's practical and candid editorial contributed significantly to the text. Jack's quite reserve and gentle prods were always welcome reminders to "make it happen." Thanks also need to be expressed to the rest of my compatriots at BMH who willingly and admirably filled the voids while I was working on this thesis.

As for family and friends, I am especially indebted to Lloyd and Barbara Schnuck. Their vision, kindness, and support provided the impetus for me to take the "first step." My gratitude to them is only surpassed by what I owe to my mother and father, whose encouragement and support has been constant in all that I've ever done. What they taught, has served me well in education, my profession, and my personal life. I love them dearly. Similar sentiments go to the rest of my family: my sister Cherie, two brothers Bill and Robert, Mac and Virginia Coupland, the Mundens, and the Petersons. I'm extremely fortunate to be surrounded by such caring family whose interest and encouragement supported me during the completion of this work.

Finally, I must thank my wife Ellen, and children David and Anne, who have patiently waited for me to persevere. They have been, and always will be my beacons. "Hey kids, looks like Dad finished first after all. Yeehaa!"

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES

LIST OF ACRONYMS

AAR	After Action Review
ACM	ALSP Common Module
ACTD	Advanced Concept Technology Demonstration
ADS	Advanced Distributed Simulation
AI	Artificial Intelligence
AirSAF	Air Semi-Automated Forces
ALSP	Aggregate Level Simulation Protocol
AMN	Air Management Node
ANOVA	Analysis of Variance
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
AVCATT-A	Aviation Combined Arms Tactical Trainer - Aviation
BFTT	Battle Force Tactical Trainer
CAP	Combat Air Patrol
CAS	Close Air Support
C2	Command and Control
C4I	Command, Control, Computers, Communications, and Intelligence
CIG	Computer Image Generation
CGF	Computer Generated Forces
CORBA	Common Object Request Broker Architecture
DARPA	Defense Advanced Research Projects Agency
DCA	Defensive Counter Air
DDM	Data Distribution Management

LIST OF ACRONYMS

DFD – Data Flow Diagram

DIS – Distributed Interactive Simulation

DMSO – Defense Modeling and Simulation Office

DMT – Distributed Mission Training

DoD – Department Of Defense

DR – Dead Reckoning

DSI – Defense Simulation Internet

ESM – Electronic Surveillance Mission

EXCIMS – Executive Council for Modeling and Simulation

FDDI – Fiber Distributed Data Interface

FEPW – Federation Execution Planners Workbook

FOM – Federation Object Module

FWA – Fixed Wing Aircraft

GUI – Graphical User Interface

HCI – Human Computer Interface

HLA – High Level Architecture

IDEF – Integration Definition For Function Modeling

IFOR – Intelligent Forces

IPC – Inter-Process Communications

JSAF – Joint Semi-Automated Forces

ISDN – Integrated Services Digital Network

LAN – Local Area Network

M&S – Modeling & Simulation

LIST OF ACRONYMS

ModSAF – Modular Semi-Automated Forces

MOM – Management Object Module

MSMP – Modeling and Simulation Master Plan

O/S – Operating System

OMT – Object Model Template

OPFOR – Opposition Forces

PDES – Parallel Discrete Event Simulation

PDU – Protocol Data Unit

PID – Process Identification

QoS – Quality of Service

R&D – Research & Development

RTI – Run Time Infrastructure

RWA – Rotary Wing Aircraft

SAF – Semi-Automated Forces

SIMNET – Simulator Networking

SNE – Synthetic Natural Environments

SNMP – Simple Network Management Protocol

SOAR – State, Operator, And Result

SOM – Simulation Object Module

STOW – Synthetic Theater of War

STE – Synthetic Training Environment

T&E – Test & Evaluation

TBM – Theater Ballistic Missile

LIST OF ACRONYMS

TCP/IP	Transmission Control Protocol/Internet Protocol
USACOM	U.S. Atlantic Command
WAN	Wide Area Network
XDR	External Data Representation

SECTION 1

INTRODUCTION

The evolutionary development of distributed computing systems has been driven by, among other things, the need to share resources, increase interactions, and operate in wider geographical regions. The diversity in application domains includes systems for Automated Teller Machines (ATMs), air traffic control and airline reservations, interactive remote instruction, remote medical teleconferencing, and joint military operations training. The availability of lower-cost, higher-performance computing networks continues to provide greater capabilities to solve new and bigger problems using increasingly sophisticated computer applications. Increased capabilities and more sophisticated applications can, however, lead to an increase in system complexity.

The number of hardware and software components that must be considered complicates understanding and managing the complexity associated with state-of-the-art distributed computing systems. Typical systems consist of hundreds or thousands of autonomous workstations connected by different kinds of communication sub-networks. The myriad of software must support, among other things, some or all of the following: data access, data consistency, processor synchronization, system security, fault tolerance, and system transparency. Additionally, dynamic workload characteristics can cause large variances in resource demands, dramatically affecting system run-time performance. Real-time processing requirements contribute to the complexity of distributed computing systems. Whether the application is a critical, hard real-time system or a non-critical, soft real-time system, understanding the effects of task scheduling strategies, fault-tolerance, communications delays, and clock synchronization are important factors for assessing and managing system performance. Detailed discussions on design issues and objectives

---

This thesis used the style guide model as presented in IEEE Transactions on Software Engineering; a publication of the IEEE Computer Society.

relevant to distributed systems are presented in [1,2,3]. Real-time systems are introduced in [4,5]. Although this thesis will not address it, advances in technology and increases in system complexity cause similar concerns for designing scientific and parallel computing systems [6,7].

Current trends will result in ever larger and more complex distributed, real-time systems. Understanding the behavior and effective exploration of the performance capabilities of these systems are predicated on the existence of good analysis tools and methodologies. Distributed computing applications must be evaluated and tuned to perform efficiently for varying hardware configurations and problem sizes. Persons making decisions about these tasks need to be provided useful information about issues within their control and in terms they can understand. Achieving these goals requires understanding which particular parameters of a run-time configuration are the most significant factors affecting performance of the application. The relationship among those factors is complex and the impact on performance can be significant.

The above discussion provides general motivation for the research presented in this thesis, namely performance evaluation of *distributed simulations* used by the U.S. Department of Defense (DoD). The goal is to provide decision-makers with an understanding of technology factors contributing to performance bottlenecks in DoD Modeling and Simulation (M&S) environments. Achieving this requires an architectural framework consisting of a definition of run-time performance, an analysis methodology, and effective performance monitoring tools. Department of Defense distributed simulation environments are sufficiently complex to justify the thesis focus and provide representative case studies to address significant and real problems. This thesis asserts that, although some performance issues may be unique to distributed simulations, many are applicable to distributed computing applications in general and the expectation is that much of what is learned will be useful for performance analysis across a broad spectrum of distributed computing domains (as opposed to just distributed simulation). The remainder of this introduction provides information on the evolution of distributed simulation technology used by the DoD and discusses the objectives of this thesis research, namely the specification of a unified architectural framework for monitoring and analyzing the run-time performance of DIS and HLA simulations.

1.1 Problem Definition

United States DoD military capabilities are defined as readiness, modernization, force structure, and sustainability; they are discussed in [8]. The DoD uses modeling and simulation technologies to enhance these capabilities. Applications include joint service training, development of military doctrine and tactics, development and testing of operational plans, technology assessments, systems acquisition, systems development and force structuring. The effectiveness of using simulation for these applications is in part determined by performance measures describing the run-time behavior of the underlying components used in the simulation environment. An abstraction of performance can provide guidance in defining the relevant performance measures and a monitoring system can gather the required data to derive the appropriate metrics (performance measures) during simulation execution. This performance information is used by various persons making decisions about the design and development of system models and simulation infrastructure, the configuration and control of simulation exercises, and overall management of M&S resources to support the various application domains. Although a variety of measures are used to assess the effectiveness of different technologies used to support these domains, performance measures are among the most significant since they support direct assessments regarding the realism and validity of synthetic environments. Performance measures are a key part of any evaluation of the technologies and as such, must convey information about the application domain and must be both meaningful and relevant to the appropriate decision-makers.

1.2 Application Domains

The use of military modeling and simulation for decision making has a relatively long and diverse history. As enumerated above, the application domains are numerous and a formal taxonomy can be found in [9]. For this thesis, however, it is useful to provide some background on the relevant DoD application domains, the technologies used for distributed simulations, and the system complexity in terms of run-time performance. The discussion clarifies the terminology for the ensuing text and serves to substantiate the claim that this research is addressing a significant issue to the DoD M&S community, as well as to other distributed computing domains.

Military training is one domain where modeling and simulation play a significant role. Training military personnel to function within complex systems has necessitated the use of M&S to create Synthetic Training Environments (STE). STE provide realistic training environments that can reduce training costs, control personnel and training area requirements, and in potentially hazardous training environments increase safety. STE consist of humans interacting with a virtual environment for the purpose of experimentation, study, or evaluation. Some STE rely solely on manned simulators or mock-ups to create the virtual environment. For some applications, having real-world entities (e.g., humans, aircraft) participate in or contribute to the virtual environment is feasible and adds to the effectiveness of the training environment. For STE that need many entities, a cost-reduction technique is to rely on computer simulations to create and populate much of the virtual environment. Many STE incorporate all three components; virtual simulators (mock-ups and manned simulators), live participants, and computer simulations. Although technical issues and limitations still exist, the increased performance and reliability of computers, Local-Area Networks (LANs), and Wide-Area Networks (WANs) allow these components to be geographically dispersed, sometimes across continents, and still participate in the STE. Distributed simulation technology allows trainees to be immersed in a synthetic environment that accurately simulates real-world operational environments. Human (trainee) perception plays a significant role in training environments, and near-real-time feedback and after-action-review systems are used to provide analysis capabilities, to enhance the training, and to assess the effectiveness of training exercises. The diversity (and consequent complexity) of integrated components used in an STE requires a significant amount of information regarding system performance and behavior to assess the validity and adequacy of the training environment.

The DoD also uses distributed simulations to create more effective environments to support analysis of military systems (e.g., fixed-wing aircraft, Command and Control, or C^2) and subsystems (e.g., weapons control, radar). New and emerging technologies provide enhanced capabilities but also add to system complexity. Accurately testing and/or evaluating the cost/benefit of integrating these new technologies requires the support of increasingly sophisticated real-time simulation environments; the objective is to create a higher fidelity synthetic air, land, and sea space for analysis of military systems. Programs for Research and Development (R&D), Test and Evaluation (T&E), technology assessments, and systems acquisition, can utilize the same synthetic environments as the training community. These application domains, however, may require higher-fidelity models and have stricter timing constraints, necessitating better and more predictable performance guarantees. As in STE, requirements exist for tools to support decision-makers during the design, development, and use of the M&S environment. Additional application domains include using simulations to support analysis of military logistics (materiel management, maintenance, and resourcing policies), and the analysis of military force structure (composition of military forces across different mission scenarios).

The technology used to create synthetic environments continues to emerge rapidly and as people create more complex, higher fidelity models, hardware and software limitations become restrictive. The technology choices that must be made to develop a realistic synthetic environment are numerous and add to complexity for the decision-maker. Manned simulators, typically large and expensive, are of limited use in large-scale environments. For training exercises that require many live participants the logistics of including the personnel can be significant. Computer simulations can provide the capability to consistently and accurately reproduce a synthetic environment for the purposes of experimentation, evaluation, and analysis, and distributed simulation is widely used to create these environments, partitioning the simulation processing requirements among many independent simulation nodes.

1.2.1 DoD Modeling Issues

A significant benefit of using distributed simulation is its ability to enhance realism by populating the environment with simulated entities and by implementing selective model fidelity, defining and controlling the faithfulness with which real-world objects are represented. Distributed simulations must include models that provide an accurate representation of the real-world system at a level of realism sufficient for the goals of the training and/or analysis. These higher resolution, higher fidelity M&S environments can be characterized by increasing spatial and temporal complexity. In DoD simulation terminology, physical models typically represent real-world entities, objects, or subsystems. A physical model has a state that represents real-world properties and where applicable, defines an interface for interaction with other physical models. Consider an STE consisting of a virtual battlefield simulation. The virtual battlefield simulates tanks (real-world entities) and specifies their sizes, traveling speeds, and armaments (state and properties). Each tank model has weapons and sensor interfaces that allow these subsystem states to be used to make decisions about when detections and engagements with enemy vehicles occur (interactions with another physical model).

Behavioral models for synthetic environments are required to simulate collective or individual human behaviors that control the physical models. These behaviors may be responses to some change in state of the virtual environment. In the above example, the execution of a behavioral model simulating the engagement of an enemy vehicle is a response to a simulation event detecting that vehicle. Since most real-world systems are not closed systems, distributed simulations used in synthetic environments will typically provide some form of environmental models to enhance realism. Again, using the virtual battlefield simulation example, including environmental models is necessary because in a real-world battlefield smoke, darkness, clouds, and dust contribute significantly to the effectiveness of sensor systems, weapons systems, and vehicle movement, as well as affecting human perception and performance. These physical, behavioral, and environmental models must provide an adequate representation of the real-world system with properties at the correct fidelity level and allow realistic perception, interaction, and interpretation by the people operating within or using the synthetic environment.

The modeling issues discussed above create many technical issues for the design and implementation of distributed simulation software infrastructure. Many of these are driven by the requirements for real-time man-in-loop capabilities for geographically dispersed players and valid interactions among the live, virtual, and simulation components. Design decisions for the components (networking; databases; parametric data; timing and coordination; simulation management and control; graphics and user interfaces; tracing and data logging; and security and protection) affect many important aspects of the system, performance being one of those. The following two examples clarify this point.

Invariably tradeoffs must be made between system performance and model resolution or fidelity. In the context of DIS and HLA simulations, terrain modeling is one of the more fundamental considerations. Better performing hardware and the availability of vast amounts of very high resolution data have led to requirements for highly detailed terrain representations. The representations can include, among other things, curved representations of large areas of the earth (spanning deserts, mountains, etc.), river and ocean bathymetry data, and multi-state cultural features (e.g., damaged and undamaged representations of buildings, roads, and bridges). Terrain attributes can include color, soil type, and texture. Mean high and low water marks can be represented for littoral regions along with sea walls and near-shore escarpments. One terrain representation used in a case study for this thesis contains over 289,000 sq. km. of terrain, 89,000 sq. km. of bathymetry, 4,000 km. of coastline, 30,000 km. of roads and railroads, 4,000 km. of pipelines, 13,000 building structures, 11,000 powerline transmission towers, 90,000 date palms (trees), 9,460,000 desert scrub bushes, and 300 bridges, overpasses, and

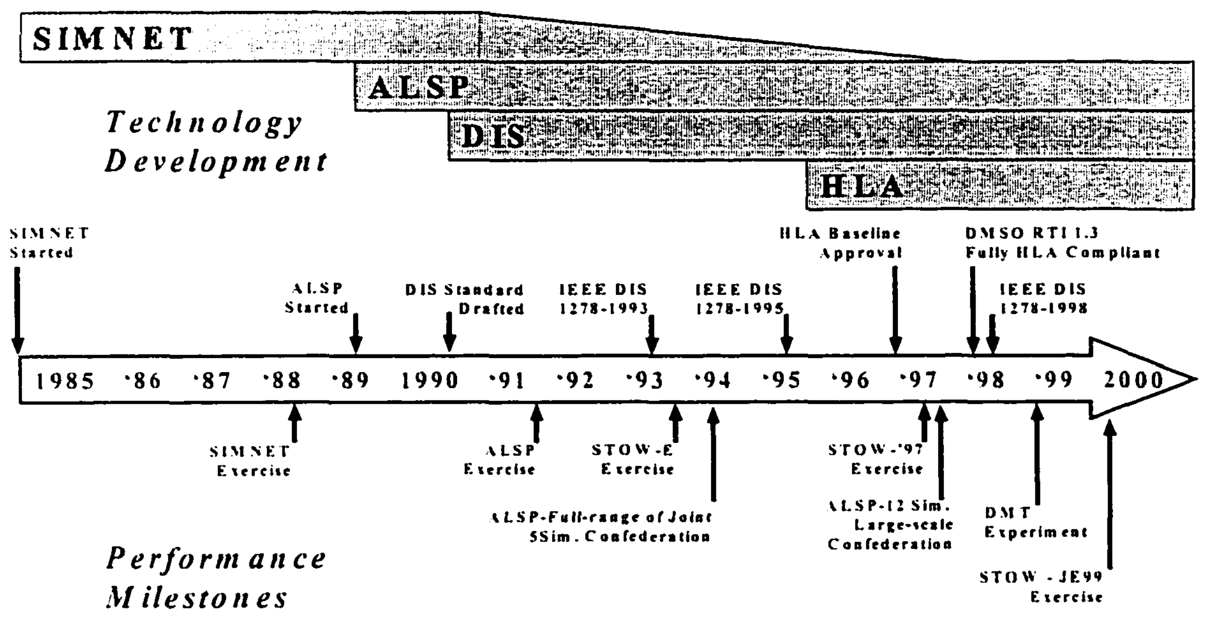


Fig. 1. The evolution of U.S. DoD distributed simulation.

causeways. Run-time processing of this high-resolution terrain and feature data can result in performance bottlenecks. For example, the run-time execution costs of intervisibility algorithms used to assess whether two simulated vehicles can see each other (i.e., line of sight calculations) can be dramatically slow when processing dense feature representations (e.g., trees, hills, buildings), and a complex road system representation can dramatically affect the processing requirements of vehicle movement algorithms (e.g., shortest-path search algorithms). Providing run-time performance data to decision-makers regarding the impact of specific terrain representations is useful for considering design tradeoffs to meet M&S objectives.

Decision-makers must sometimes make design tradeoffs on non-technical issues and the impact can affect run-time performance. Consider a highly distributed design and development environment; a characteristic of many DoD distributed simulations (i.e., many different government contractors and/or government agencies). Invariably the development process results in disparate model implementations and simulation architectures and the integration of these simulations can exhibit invalid run-time interactions and resultant model behavior. It is possible to alleviate some of these run-time problems by taking a centralized approach for specific modeling workloads (functional). As an example, one real-world system resorts to a centralized ordnance server, a simulation responsible for simulating guided weapons in training simulations. The motivation for the centralized ordnance server is driven by the need for a *level playing field* with respect to guided weapons, meaning that all simulations use the same weapons models. In this system, within each simulation the guided weapons are employed a very small percent of the time (relative to the real-time simulation). Simulating the weapons is computationally expensive and the existing implementations are compute and memory bound processes, so it makes sense to allocate dedicated compute resources in the form of a centralized ordnance server. This eliminates the need for each simulation to set aside enough compute resources to support guided weapons modeling and achieves the requirement for a level playing. The decision to centralize the guided weapons models however creates specific performance requirements for the ordnance server, namely to simulate all guided weapons in the synthetic environment (which can be large) while guaranteeing valid interactions with remotely simulated targets. Performance data was gathered and used to support network latency analysis of weapons firing messages sent to the ordnance server, and weapons detonation messages sent from the ordnance server. Timing data was also gathered on the ordnance server's execution. The performance analysis output provided positive feedback to the decision-makers regarding the feasibility of this centralized approach in the presence of realistic workloads.

This section has thus far presented some of the modeling requirements of DIS and HLA simulation environments and discussed the importance of providing performance information to support decision-makers. The remainder of this section provides relevant background information on the evolution of DoD simulation architectures and provides additional justification for this thesis research.

1.2.2 DoD Distributed Simulation Architectures

Significant advances in simulation architectures have been made over the past 10 years. Technology improvements have facilitated these advances and include lower-cost workstations, high-performance local and wide area networks, and high-resolution, real-time Computer Image Generation (CIG) and display systems. Advances in simulation architecture have also been driven by requirements for higher fidelity models and have proved useful across a greater range of applications, as discussed in the previous section. Additionally, policy directives from the U.S. government mandate the increased use of simulations by the DoD for procurement, test, and evaluation activities. The history of DoD distributed simulation is relatively short but follows a well-defined chronological path. Figure 1.1 shows the evolution in the context of the most significant distributed simulation protocols and architectures. Its origins start with the Simulator Networking (SIMNET) project and proceed to the current technology thrust, the High Level Architecture, or HLA. The figure also shows significant simulation events and annotates some of the characteristics regarding system complexity and performance. These characteristics sound a recurring theme, namely, the need to observe and record system performance to better understand the simulation environment.

1.2.2.1 Simulation Network (SIMNET)

SIMNET was a research project sponsored by the Defense Advanced Research Projects Agency (DARPA) in partnership with the U.S. Army. Initiated in 1983, the goals of the program were to develop technology for networking large numbers of interactive manned simulators (i.e., combat vehicles and combat support elements). The objectives were to provide realistic training and practice for fully manned platoon-, company-, and battalion-level units to fight force-on-force engagements against an opposing force.

The SIMNET architecture networked individual manned tank simulators together using microprocessor-based workstations (one simulator per workstation) and 10Mbps Ethernet. Individual LANs were linked together to form a WAN connecting geographically dispersed simulators. This WAN was the precursor to the DoD's Defense Simulation Internet, or DSI. A terrain database was replicated on each simulator and provided a globally consistent view of the virtual battlespace. Low-cost, real-time CIG systems were used to provide the crewed simulators with a three dimensional view of the battlespace. An application-level protocol was developed (SIMNET protocol) to allow simulators to communicate; consisting of data packets for controlling simulator activation/deactivation, transmitting vehicle appearance data (entity state data), re-supplying and repairing simulated vehicles, weapons firing and detonations, and vehicle collisions. Another important component of the SIMNET architecture is the *data logger* used to record, replay and support analysis of simulation exercises.

The SIMNET architecture resulted in several significant distributed simulation design principles still in use today. Among them are 1) object-based M&S design and development, 2) simulation autonomy (decentralized simulation control), 3) data transmission restricted only to that relevant and required by other simulations, and 4) dead-reckoning algorithms used to reduce network and processor loads.[1] Additionally, performance studies done using SIMNET provided the foundations for understanding factors that impact distributed simulation performance in DoD application domains. The studies elucidated the [still relevant] tradeoffs in communication costs, simulation processing costs, and model fidelity. Another significant contribution of the SIMNET program was the use of Semi-Automated Forces (SAF), to populate the synthetic training environment. In the later part of the SIMNET program, the use of SAF to populate the virtual battlespace significantly added to the realism of the simulation exercises.

The SIMNET program was formally completed in 1991; however, the U.S. Army continues to use the SIMNET system. It provides real-time, interactive training, and

[1] Dead Reckoning is a method for the estimation of the position/orientation of an entity based on a previously known position/orientation and estimates of time and motion. It is employed as a network bandwidth reduction technique to limit the rate that Entity State PDUs are issued.

provides the capability to train and sustain collective (crew through battalion level) tasks and skills in command and control, communication and maneuver, and to integrate the functions of combat and combat service support. SIMNET provides this training in an environment that is significantly lower in cost and risk than comparable "live" field exercises.

SIMNET proved that distributed simulation was a viable paradigm for meeting DoD simulation goals. Its architecture provided many key design principles used in more recent DoD distributed simulation systems (i.e., DIS). Detailed information about SIMNET, its architecture, and communications protocols is found in [10,11,12,13,14].

1.2.2.2 Distributed Interactive Simulation, Semi-Automated Forces, and Intelligent Agents

The successful use of SIMNET provided the impetus for using distributed simulation technology to create larger and more realistic synthetic training environments. It seemed practical to use this technology to support other DoD simulation environments as well, such as test and evaluation of new combat vehicle systems and subsystems. In 1989 the first Distributed Interactive Simulation (DIS) standards workshop took place. The primary motivation in creating the DIS standard was interoperability. In the context of DIS, interoperability means linking dissimilar simulations together using a standard protocol to create a consistent and coherent view of a synthetic land, air, and sea space.

The SIMNET protocol provided the baseline for DIS and position papers and workshops evolved the standard throughout the early '90s. DIS is an application (i.e., simulation) protocol based on Protocol Data Units (PDUs) associated with entity state and entity interactions. The standard describes the form and type of PDUs making up DIS messages that allow communications among simulations. The PDUs include:

- Entity State PDUs communicating an entity's state (e.g., identification, physical appearance, location, and orientation, and specific capabilities).

- Fire and Detonation PDUs associated with the firing and impact/detonation of weapons rounds.
- PDUs associated with vehicle and weapons logistics (i.e., re-supply and repair).
- Collision PDUs associated with collisions between entities.
- Emission PDUs, Transmitter PDUs, Signal PDUs communicating information about electromagnetic characteristics of entities.
- Exercise Management PDUs managing among other things, the creation and deletion of entities, and the starting, suspension, and termination of a simulation.

There are other PDUs and a complete discussion of the DIS Standard can be found in [15,16,17]. From an architectural standpoint, DIS-based simulations incorporate many of the basic principles used in the design and implementation of SIMNET, including entity-on-entity interactions, dead-reckoning algorithms to reduce network and processor loads, a common terrain representation, and a common protocol for sharing information. However, the evolutionary development of DIS and its use by the DoD coincided with the integration of a larger number of disparate simulations, geographically dispersed across wider regions (throughout the U.S. and other countries). The capability emerged to simulate larger numbers of entities interacting in more realistic synthetic environments (i.e., dynamic terrain objects and weather anomalies). Different models could execute with different fidelity levels and similar models could execute at varying levels of resolution. All of these factors add to the complexity of DIS-based simulations and each can significantly impact run-time performance. Perhaps the most significant impact of the DIS evolution was the increased significance of SAF in DoD distributed simulations.

Semi-Automated Forces were simulations originally developed in the late 1980s in SIMNET. They were used to generate a relatively small number of simulated combat vehicles to enhance the realism of SIMNET training exercises. Since the number of available manned training simulators was limited, using SAFs to populate the battlespace with opposition forces increased the number of manned-simulators available for training blue forces (U.S. military forces). The successful use of SAF during SIMNET resulted in the recognition that SAF would play a significant role in the future of DoD modeling and simulation.

Semi-Automated Forces represent a broader class of Computer-Generated Forces (CGF). In the DIS environment, SAF can be characterized by an entity level representation of combat units. These entities act as credible surrogates for modeling the behavior of manned simulators. To achieve this, SAF have a wide and complex array of information requirements that includes:

- Physical battlespace data describing things such as vehicles, weapon and sensor systems, and other assets in the scope of the combat portrayed by the system.
- Physical environment data such as terrain databases and environmental effects such as smoke, weather, diurnal and seasonal effects, and electro-magnetic radiation.
- Hierarchical representations of military units (e.g., platoons, companies).
- Data and mechanisms for commanding and controlling the SAF units to mimic the real-world approach to command and control.
- Effective human-system interface so operators can interact with and control the simulation.

Different mechanisms exist for implementing behavioral models in SAF. One common technique is called *Taskframe Behaviors* and utilizes a software method for encoding task models through the use of finite state machines and arbitration methods. Taskframe technology allows simulation entities to exhibit simple autonomous behaviors (tasks), such as an aircraft flying to a waypoint. SAFs still require significant human management however when it comes to executing complex coordinated simulation activity (e.g., air-to-air engagements). In a densely populated synthetic environment, the human controlling the SAF can quickly become overwhelmed by the workload required to control entities (units). This deficiency led to the development of Intelligent Forces (IFOR) for use in DIS exercise environments. IFOR represent another class of CGF and are more highly automated than SAF, the goal being to use these intelligent agents to further reduce man-power requirements of simulation-based training exercises. A specific implementation of IFOR called TacAir-Soar is used extensively throughout this thesis research. TacAir-Soar is based on the use of Soar technology, an artificial intelligence technique initially developed at Carnegie Mellon University.[2] TacAir-Soar was developed jointly by researchers at the University of Michigan and the University of Southern California. TacAir-Soar enables fixed-wing and rotary-wing aircraft to be simulated as fully automated forces controlled by intelligent agents (simulated pilots). A rule-based logic engine controls the IFOR behaviors. These behaviors are specified as a set of goal hierarchies and are used to conduct doctrinal missions. A simulation control application can provide close control of the intelligent agents and their behaviors. A spoken language interface using the grammars of real-world pilots and controllers can also be used. A thorough overview of IFOR is presented in [18, 19]. The new capabilities provided by DIS, SAF, and IFOR dramatically affect run-time workloads of simulations. The complexity created by integrating the various technologies used to implement DIS justifies a formal approach to understanding the impact of using these technologies to assess and evaluate simulation performance.

1.2.2.3 Aggregate Level Simulation Protocol (ALSP)

The Aggregate Level Simulation Protocol is another DARPA research program that coincided with DIS development. The ALSP permits multiple, pre-existing warfare simulations to interact with each other over local or wide area networks. The grouping of

[2] Soar was originally an acronym for State, Operator, And Result (SOAR), which together constitute one basic search step in Soar. For some unknown reason the community has dropped the reference to the acronym and just uses the term Soar and references it as a general Artificial Intelligence (AI) architecture.

simulations is called an *ALSP Confederation*. The ALSP program was initiated in 1989. The system architecture continued to evolve as experiences gained in real training exercises provided requirements for design and implementation changes. By 1994, the confederation consisted of five different simulations encompassing the full range of joint military operations including, air, land, and sea warfare. By 1997, the ALSP Confederation consisted of twelve different interacting systems and the ALSP Confederation continues to support large-scale, joint military training exercises.

Like DIS, the ALSP architecture is based on the successful characteristics of SIMNET, namely autonomous simulations capable of interacting in a geographically distributed environment, and using a message-based communications paradigm (a standardized protocol). Due to the nature of the pre-existing simulations, ALSP has some unique requirements not characteristic of SIMNET. These are: 1) the individual simulations maintain and advance time in different ways so a mechanism is needed to synchronize time and coordinate simulation events, 2) each simulation uses its own databases so a standard representation of shared data is required, and 3) simulations have different design architectures so a method is needed to enable each simulation to exist within the ALSP Confederation despite its design and implementation differences.

The ALSP program addressed these issues by developing an extensible communications architecture consisting of several standardized protocols and processing components used for time and data management functions. Each simulation (called an *actor*) is tightly coupled with a *translator* component that is responsible for converting data into a common representation. The translator provides a bridge between an actor and the confederation by directly interacting with an *ALSP Common Module (ACM)*. There is one ACM for each actor/translator pair. The ACM manage the joining and leaving of actors from the confederation, coordinate the actors' local time with confederation time, and manage object and object attribute ownership and updates. An *ALSP Broadcast Emulator (ABE)* manages communications among all ACMs in the confederation. ALSP has two basic protocol layers; an actor-to-actor protocol and an actor-to-ACM protocol. The actual communications connections primarily use standard TCP/IP and Ethernet technology. The ALSP history, architecture, and design are discussed in [20,21,22,23].

As the size of the ALSP Confederation increased, run-time performance problems became a significant factor in assessing the effectiveness of ALSP to support training objectives. Similar performance problems were apparent when scaling DIS-based training exercises and the approach taken to improve performance was to reduce bandwidth requirements, message processing overheads by the actors, and code optimizations of critical simulation functions. These performance enhancements temporarily alleviated some performance problems. However, evolving requirements for greater simulation functionality and confederation capabilities continue to create performance bottlenecks that limit the quality of the training experience.

1.2.2.4 High Level Architecture (HLA)

Prior to the early-90's, the DoD M&S programs can be characterized as narrowly focused, stove-piped simulation design and development efforts, meaning the simulations were typically implemented and executed in an autonomous manner and provided very little interoperability with other systems. Many DoD simulation development efforts were plagued with cost-overruns, late deliveries, and limited reuse. To address these concerns, in 1991 the U.S. Under Secretary of Defense (Acquisition), established the Executive Council for Modeling and Simulation (EXCIMS). The council was tasked to provide a focused vision of modeling and simulation to support and enhance U.S. military capabilities. This resulted in the U.S. DoD Modeling and Simulation Master Plan (MSMP) completed in 1995. The plan outlines strategies for achieving future DoD M&S-based capabilities, defines an initial step in a process for developing M&S functional objectives, and attempts to foster the development of a common set of M&S standards, processes, and methods among civilian and defense industries. The MSMP defines a set of objectives required to realize the DoD M&S vision; the first being to "Provide a common technical framework for M&S". This objective is the basis for the design and development of the High Level Architecture (HLA), a facility that promotes the interoperability and reuse of models and simulations. In September, 1996, HLA was adopted as the standard technical architecture for all U.S. DoD simulations. The HLA consists of three principal components: 1) the HLA Rules, 2) Interface Specification, and 3) the Object Model Template.

The rules specify the key principles behind HLA and provide a basis for meeting the objectives of using HLA, namely reuse and interoperability. The rules can be partitioned into two groups, those that apply to a *federation* (a group of interacting federates with a common goal), and those that apply to individual *federates* (simulations). Federation rules require the development and use of a Federation Object Model (FOM) that specifies what types of data are shared, and when and how the data is exchanged among federates. The federate rules require the development of a Simulation Object Model (SOM). The SOM specifies those objects, attributes, and interactions of a federate that can be made public in a federation. Both the federation and federate rules discuss object ownership policies and provide rules that allow individual federates to have different time management mechanisms and still participate in the federation in a coordinated fashion.

The HLA Interface Specification provides a well-defined interface that allows federates to interface with the Runtime Infrastructure (RTI), invoke its services, and respond to RTI requests. The RTI is in essence a distributed operating system that provides services that support the federate-to-federate interactions. The principal service categories are: 1) Federation Management, 2) Declaration Management, 3) Object Management, 4) Ownership Management, 5) Time Management, and 6) Data Distribution Management. The HLA Interface Specification defines how a federate accesses these RTI services.

The Object Model Template, or OMT, is a standard form that provides common documentation to define the SOM and FOM. The OMT provides a structured format with commonly understood terminology and objects. It provides a source of information for assessments about the suitability of a federation for specific applications, or the suitability of an individual federate for participation within a specific federation. The goal is to use the OMT as a means of selective development and reuse of HLA federates and federations. A more thorough overview of HLA can be found in [24,25].

The mandate for new simulations to be HLA-compliant is having a significant impact on DoD M&S community and will continue to do so. The current HLA specification provides a baseline for building distributed simulation environments that can be characterized by greater interoperability and reuse. Interoperability among different simulations is achieved by using the RTI to facilitate data sharing. The requirement to document characteristics of object representations using the OMT enables consistent data interpretation among federates participating in a specific federation which also enhances interoperability. Initial implementations of the infrastructure, however, have had dramatic effects on run-time performance. Performance evaluations are presented in [26,27,28].

A likely artifact of greater interoperability and reuse is increased complexity in terms of the static distributed simulation architecture and the dynamics of the run-time environment. It is very difficult to anticipate all the run-time interactions among the various aspects (network topology, hardware, operating system, RTI, simulation infrastructure) that constitute a DIS or HLA simulation environment. The need to understand the impact of current and future DIS and HLA design and implementation decisions provides additional impetus for the research presented in this thesis and reinforces its significance to people making decisions about the use of distributed simulation technology.

1.3 Thesis Objectives

Sections 1.1 and 1.2 characterize the complexity of DoD M&S environments and justify a well-defined framework of DIS/HLA performance. The discussion provides the motivation for the objectives of this thesis. The original concept of the research was to provide a generalized architecture for distributed simulation performance analysis. During the initial stages of thesis development, however, it became readily apparent that the scope of performance analysis within DoD distributed simulation environments was sufficiently complex and that focusing research objectives on DIS and HLA simulations provided a significant, yet reasonable, bound on research objectives. Further rationale regarding the refinement of thesis objectives is found in the conclusions Section. The formally defined thesis objectives are the following:

- Define a framework useful for characterizing DIS/HLA simulation performance. The framework shall include a conceptualized view of performance in the context of DIS and HLA simulations, and a taxonomy of performance measures useful to different decision-makers involved with the DIS / HLA life-cycle.
- Develop a measurement, monitoring, and analysis infrastructure useful for supporting DIS and HLA simulation performance.
- Relate the costs of obtaining the performance information for use in both dynamic and static performance analyses in terms of the intrusiveness of run-time monitoring and measurements of DIS/HLA simulations.
- Provide a baseline of practical experiences for future work related to performance measurement and monitoring for the design, configuration, and control of DIS and HLA simulations.

The significance and contribution of the research are to:

- Provide a framework to identify and understand a set of meaningful and useful performance measures for persons making decisions within the context of DIS and HLA simulation environments.
- Provide a performance monitoring software implementation that is useful and extendible to support different modeling and simulation applications and domains and that could be used for other distributed computing applications where performance monitoring and evaluation are desirable .

- Provide documented experiences useful for understanding the tradeoffs associated with gathering specific types of performance information and metrics as they relate to data granularity, data collection rate, and the overall run-time intrusiveness of the monitoring system.

- Present results from real simulation exercises that provide those persons making decisions about performance measures with meaningful and relevant data for understanding such things as the utilization of simulation and network resources, partitioning the simulation workload to account for scenario effects, estimating hardware / software requirements, and understanding the impact of inserting specific technology into a distributed computing environment.

The goal is to provide a set of tools to effectively provide guidance for the diagnosis and analysis of simulation run-time performance. The metrics should provide information to model developers and programmers, exercise and configuration planners, system analysts and program managers, and other decision-makers who need to understand performance of a distributed simulation's execution in high-level terms, relating the myriad of DIS and HLA simulation technologies to the objectives of simulation studies. The maturity and scope of DIS and HLA simulation environments provide a realistic environment to develop this thesis and ensure that real performance issues are addressed. The expectations are that results of this research and implementation will be reusable beyond the DIS and HLA domains; therefore, a secondary objective is to provide a performance analysis framework useful for other distributed simulation environments.

Run-time acquisition and analysis of the performance information should provide feedback on the execution of the simulation and allow decisions to be made about the efficiency of the current distributed simulation environment. The framework and monitoring infrastructure should support decisions about the configuration and control of the available hardware and software resources for future distributed simulation exercises. It should provide information that lets a decision-maker anticipate problems which will invariably exist in any technically complex system (i.e., distributed simulation environment) and provide insight to where problems exist in the system. The information is useful to develop a timely resolution strategy, assess risks, and document and track problems. The performance information should also be useful to help remove biases from design, operational, or analysis viewpoints that different people bring to the table.

1.4 Thesis Approach

The thesis will clearly conceptualize the meaning of distributed simulation performance in the context of DIS and HLA simulations and will present a mechanism for capturing performance data in the run-time environment. Meeting the research objectives requires a well-understood and controllable simulation environment, suitable for validating the performance monitoring and analysis methods presented in this thesis. An ideal case study was selected; the DARPA Synthetic Theater of War (STOW) program. STOW is an evolving Advanced Distributed Simulation (ADS) technology that has successfully demonstrated the capabilities of high-resolution simulation to support military training, systems acquisition, analysis, and test and evaluation [29]. The origins of STOW technology are DARPA's previous work in synthetic forces including the SIMNET program. STOW creates a realistic, distributed simulation environment where synthetic forces are modeled at the platform level and the synthetic environment includes representations of real-world terrain, space, oceans, and environmental effects [30,31]. STOW has been successfully used to support U.S. Atlantic Command's (USACOM) Joint Task Force (Tier III) training. STOW requirements included the ability to generate enough ground, air, and sea forces to simulate theater level operations. Additionally, it had to integrate real-world Command, Control, Computer, Communications, and Intelligence (C4I) systems to support component-level training requirements. The STOW component of this exercise consisted of 500 computers generating up to 8000 simulated objects, distributed across sites throughout the United States and United Kingdom. To achieve the large-scale, networked requirements of this exercise, a synthetic battlespace was created using HLA. The STOW program developed an advanced, high-performance networking infrastructure based on ATM and Multicast/IP technologies, the goal being to reduce bandwidth requirements and transmission latencies [32]. The showcase for DARPA's STOW program was the STOW Advanced Concept & Technology Demonstration (ACTD) conducted in 1997. The performance abstraction, the taxonomy of performance measures, and the performance monitoring software presented in this thesis were initially developed to support the STOW ACTD. STOW technology is an excellent case study that:

- Provides a complex simulation environment with many run-time characteristics affecting performance, supporting the design and iterative refinement of an abstract representation of distributed simulation performance.

- Provides a large and sufficiently complex simulation environment (both run-time behavior and software architecture) suitable for designing, implementing, and analyzing different monitoring schemes and for understanding the cost/benefit of doing performance analysis in terms of the perturbation of the analysis results.

- Allows assessments to be made regarding the impacts on performance of a dynamic simulation architecture and configuration and the framework's flexibility to changing requirements in performance information.

- Supports real-world simulation exercises, with a diverse set of decision-makers, and allows assessments to be made about the utility of the performance framework and techniques for measuring and monitoring performance information.

- Is used in multiple simulation domains and supports a broad range of requirements for different kinds of performance information.

Interacting with the STOW program provided an opportunity to develop a baseline of knowledge regarding the kinds of performance information that was meaningful and useful to different decision makers. An initial concept of useful performance information was developed and the simulation application was instrumented. Data collected during real exercises was analyzed and used to provide feedback during model development, support workload partitioning during pre-exercise planning, run-time monitoring of the distributed environment during the simulation exercise, and post-exercise analysis of distributed simulation performance. Experiences from the STOW ACTD supported iterative refinement and extensions to the initial performance abstraction and monitoring software. As STOW continued to evolve and its technology transferred to other DoD programs and M&S domains, the conceptual understanding of what DIS and HLA performance means was refined. Additionally, the costs of monitoring have been quantified and weighed qualitatively against the value added by having run-time performance information available to make important decisions regarding the availability and use of the different resources and assets in a distributed simulation environment.

Acceptability criteria for successful completion of this research is based upon meeting the research objectives and showing the utility and breadth of application of the performance framework which relates to the effectiveness with which the research contributions can be used and which is in large part a qualitative assessment of the successful completion of the validation requirements. Validation is based on examining the case studies presented in this thesis, understanding the positive impact the performance information has on real-world programs, and successfully initiating the transfer of the research and technology to the simulation community. This is significant as the crux of the validation effort is getting decision makers to agree on the applicability and usefulness of the performance measures as well as the methodology for capturing the data. The two most fundamental validation questions to answer are: does the performance information provided meet the user's objectives of analyzing simulation performance and does using the proposed framework significantly impact the performance analysis process (succinctly, what is the added-value of using the proposed framework?).

The approach taken in this thesis will establish a clear, logical path from the conceptualization of distributed simulation performance to the conclusions drawn about distributed simulation performance during the case studies.

1.5 Thesis Organization

The first section of this thesis has introduced characteristics of typical distributed simulation environments and relates the complexity of these environments with the enabling technologies underlying the system implementation. It also traces the evolution of U.S. DoD distributed simulations from the founding SIMNET program to the emerging HLA standards and discusses the components that make up the synthetic environments (dynamic virtual worlds) associated with the various military M&S domains. Section 2 discusses related research and further articulates the significance and contribution of this thesis research. Sections 1 and 2 provide a foundation that justifies the need for a well-defined framework for performance analysis of not just DIS and HLA simulations, but other distributed simulations in general and sets the context for the research presented in this thesis.

1.5.1 A Framework for Characterizing DIS and HLA Simulation Performance

Section 3 establishes the basis for the specification of the framework presented in this thesis. It defines an abstraction that identifies the most significant factors affecting the run-time performance of DIS and HLA simulations. These factors are used to establish a taxonomy of performance measures useful for characterizing distributed simulation performance. This taxonomy is a helpful tool for persons trying to define a set of meaningful performance metrics useful to meet DIS and HLA performance analysis objectives. Requirements for a performance monitoring system to monitor, measure, and collect performance data is also discussed. Finally, the section introduces the crux of the thesis, a model that represents a unified, architectural framework for the performance analysis of DIS and HLA simulations. The notion of "unifying" in this context means a framework that provides an identified set of performance measures common among DIS and HLA simulations, a mapping of those measures to a set of metrics collected with a general performance monitoring infrastructure, and the transformation of those metrics to meaningful information used by an array of decision-makers involved in different activities associated with a simulation life-cycle. This unified framework is depicted in an important graphic in Figure 3.4.

1.5.2 Performance Monitoring

Section 4 discusses the design and implementation of the PerfMETRICS monitoring system, a software-based monitoring system created as a part of this thesis research. The system architecture and design are presented along with the motivation for specific implementation techniques. Details regarding the various logical and physical components of the software are discussed in detail including the instrumentation component, the collection daemon, and the graphical user interface component for displaying and logging the performance data.

1.5.3 Performance Monitoring Use Cases and Conclusions

Section 5 presents use cases associated with the application of the framework to real-world M&S programs. The use cases are significant for demonstrating the relevance of the framework across a range of performance study objectives to support the design configuration, and control of DIS and HLA simulations used in different M&S domains. All of the cases are based on the use of PerfMETRICS and the underlying framework describing the relevant performance metrics. The use cases discuss in detail the use of the framework to support model design and testing, scenario configuration, monitoring of the resources, dynamic load-scheduling, capacity planning, and technology assessment and insertion, support application design and development, and run-time control of DIS and HLA applications. Conclusions from this thesis research are presented in Section 6.

SECTION 2

RELATED RESEARCH

This section reviews literature related not only to the performance analysis and evaluation of DIS and HLA-based simulations but also to other distributed simulations, applications, and computing environments. The objective of this section is to show the need for an analysis framework suitable for acquiring and presenting performance information to people who design, configure, and control DIS and HLA simulations. This information must be presented at various levels of abstraction depending on its intended use by decision-makers. Research and performance studies of distributed systems are applicable across a variety of applications and this section discusses issues relevant to performance evaluation and modeling of distributed systems as well as distributed simulation. Although the emphasis of this research is on developing an analysis framework and deriving suitable performance metrics for DIS and HLA simulations, this section introduces the monitoring, measuring, and presentation of performance information of different distributed application domains; each application exhibits aspects of performance studies that are integral to any analysis method. These existing and well-documented topics provide a basis for integration and reuse within the scope of this research.

2.1 Performance Analysis of Distributed Systems and Other Distributed Simulations

Measurement and monitoring are integral components of any performance analysis for distributed simulation. The specification of performance information and the data acquisition used to derive that information can significantly affect the feasibility of the performance analysis. Additionally, the intrusiveness of monitoring must be understood to assess whether the monitoring process has perturbed the results of the analysis. Generally, monitoring of distributed systems is considered an event-based activity. For simulation monitoring, the definition of performance events of interest is based on the goals of measurement, the events necessary to describe the behavior of the simulation, and the simulation instrumentation.

Typically, three fundamental design techniques exist for monitoring distributed systems. Hardware, hybrid (consisting of both hardware and software components) and software monitors each provide benefits that make them more suitable for specific applications, depending on the type of information required for analysis and the level of intrusiveness that can be tolerated [33]. Hardware monitoring consists of dedicated hardware that passively monitors the target application. It is a low-level monitoring system that fetches signals transmitted on system buses. Hardware monitoring is the least intrusive but can be limited in the kinds of process-level events it can capture and use for performance analysis. In the context of distributed applications this means data shared among different processes and transmitted via I/O channels. Another disadvantage is the cost and limitations of hardware that is typically machine-dependent, a significant factor given the current trend toward more heterogeneous distributed computing environments.

Software monitoring is an approach that uses only instrumentation code to detect and process the target program's run-time behavior. Since the instrumentation code is embedded within the monitored program, it consumes compute cycles that are otherwise used by application code. As a result, software monitoring can be intrusive. It does, however, provide the greatest flexibility in terms of detection, collection, and analysis of process-level performance information. A hybrid monitoring system relies on software instrumentation of the target application but uses dedicated hardware to capture and process the performance data. This technique reduces monitoring intrusion by minimizing the overhead of detecting and collecting the performance data. It does not, however, eliminate the cost of executing the performance instrumentation code. The intrusive effects of monitoring go beyond the timing errors induced by executing performance instrumentation code. Timing errors can induce the re-sequencing of simulation events, possibly leading to incorrect execution of the distributed simulation and misleading or incorrect analysis of results. Training environments based on simulations are particularly susceptible to this since trainees' interactions with the simulation are based on decisions and responses to the events they perceive happening during run-time. A complete discussion of the issues related to monitoring of distributed and real-time systems is found in [33,34,35]. Examples of software monitoring approaches for distributed applications and operating systems are found in [36-40].

Run-time monitoring is just one issue related to performance analysis. Performance visualization and prediction are two other areas of interest for performance analysis. The distributed computing community has recognized the need for an integrated environment to monitor, visualize, and model the behavior of distributed and parallel applications. Considerable interest exists in the development of automated tools that can help isolate and correct performance problems of distributed applications. However, existing tools' capabilities raise many concerns. Typical complaints include the requirement to have a sophisticated understanding of the application to use the tool, widely varying interfaces and functionality in a heterogeneous environment, inability to reuse components of existing tools to build or extend other tools, and hand-coded instrumentation of the application for data monitoring. Pancake, Simmons, and Yan provide a short overview of the issues surrounding parallel and distributed processing tools for both prediction (modeling) and measurement (monitoring) [41].

The importance of the visualization of performance information for distributed and parallel applications is discussed in Heath, Malony, and Rover [42]. This paper presents a clear and concise model for presenting performance information, based on the successes gained in the visualization of scientific and engineering data relating physical systems and their behavior. The significance of this research as it relates to the performance analysis methodology presented in this thesis proposal is its emphasis on presenting the information at a level of abstraction meaningful to its intended audience. Another interesting paper deals with the use of feedback from monitoring tools to assess and engineer program modifications [43]. Fickas and Feather introduce the idea of using program requirements and assumptions to define what components of the software implementation should be monitored. The monitor detects violations of the system's requirements and provides feedback to people responsible for maintaining or designing the system. Additional examples and case studies are discussed in [44,45].

Other approaches to performance evaluation of parallel and distributed applications are based on modeling, both analytically and through the use of simulation. In [46,47] Dickens, Heidelberger, and Nicol illustrate the use of simulation to provide a predictive model of a parallel code's performance. It is essentially a timing simulation to estimate the speedup/slowdown performance of a parallel code on an Intel Paragon. It has proven to be accurate to within 5 percent of the actual execution times when the application was run on native hardware. Although in this paper, the application's domain was not a simulation, the paper illustrates the efficacy of using simulation to predict certain performance characteristics of a distributed program. Sarukkai and Mehra provide another example in [48] of a predictive model used for scalability analysis of parallel programs.

Research on the performance of distributed simulations can be considered based on the type of simulation and its environment. Typically, the Parallel Discrete Event Simulation (PDES) community bases its performance analysis exclusively on speedup, since the primary goal is to get the simulation to execute as fast as possible, decreasing the time it takes an analyst or decision-maker to obtain output and analysis data. Often PDES are run on multi-processor architectures, but even in distributed environments the most frequent presentation of performance results is the speedup relative to the number of processors participating in the simulation exercise. Another comparison is the speedup relative to the message density (number of messages per processor). Fujimoto presents a detailed discussion of PDES and its performance goals in [49]. Fujimoto and Falsafi present results of PDES performance studies in [50,51]. Many aspects of PDES affect performance including models and their interactions, simulation infrastructure overhead, and distributed processing overhead. Although the metric of speedup is measured relative to the number of messages and/or processors used, other static characteristics and run-time behaviors of the PDES such as queuing disciplines and operating system overhead affect performance. Often the information reported is not adequate to assist people in making decisions and assessments about the configuration and run-time characteristics of the distributed simulation. In [52], Carothers, et al. present a visualization tool (PvaniM) for providing insight into the run-time performance of a variant of the Time Warp (TW) Operating System used to execute PDES in a distributed computing environment. The tool provides graphical views of the TW component, the middleware, or simulation infrastructure. Performance of PDES has proven to be very dependent upon the communication patterns exhibited by an application and the PVaniM tool provides low-level information about the application's time management and synchronization characteristics (i.e., rollback, aborted events, time-advance).

2.2 Performance Analysis of DIS and HLA Simulations

Distributed Interactive Simulation is a distributed simulation based on entity-level interactions (as opposed to aggregate-level interactions among groups of entities). In some DIS-based applications it is desirable to simulate as many entities as possible to enhance the realism of the simulation environment. This makes scalability an important characteristic for DIS. Vrablik and Richardson present the results of benchmarking a DIS that generates CGF for doing military training, doctrine development, and test and evaluations [53]. The paper emphasizes a performance metric of entities per workstation (vehicle count), relating the metric to various software and hardware configurations. White reports the results of a similar study but discusses the details and implementation of the software architecture, trying to further explain why certain configurations obtain differing levels of performance [54]. The discussion, however, relates execution times for certain functions in the simulation's implementation and this low-level analysis of why a certain vehicle count is achievable is meaningful only to a programmer or model designer knowledgeable and familiar with the simulation's software architecture.

Previous DIS exercises required the simulation of up to 5000 entities on the DSI, a network of workstations geographically dispersed throughout the United States, England, and Korea. Prior to the evolution of multicast technology, DIS relied on broadcast communication services; all hosts on a network received every other hosts' data, regardless of its interest level.[3] The bandwidth requirements for sharing data in this kind of DIS environment proved to be the limiting factor and past performance studies and evaluations focused on the networking and communications costs associated with operating in the local and wide-area networks. Smith, Schuette, Russo, and Crepeau proposed a mathematical model to characterize the performance of Distributed Synthetic Forces (DSF) [55]. It discusses how the major limiting factor in DSF is the costs of distributing (replicating) entity information throughout the simulation environment. It accounts for the networking costs (the protocol stack and data transmission) and discusses some experimentation done. Experiment results are used to verify a scalability metric. Once verified, the scalability metric could be used to assist people who need to make decisions about resource requirements and hardware configurations for future distributed simulation exercises.

Recent advances in communications services (e.g., multicasting, ATM, fiber optics) have relaxed bandwidth requirements associated with wide-area networks linking individual DIS LANs. Early bandwidth reduction techniques (e.g., dead-reckoning) significantly reduced the amount of traffic seen at the interfaces to individual simulation hosts. The development of the HLA RTI has further reduced bandwidth requirements using techniques such as relevance filtering. The cost, however, in the case of the RTI is the overhead in making real-time decisions about how to route data among potentially hundreds of hosts participating in a simulation exercise. Additionally, as the realism (validity) of the synthetic environment increases, typically so does the density (number)

[3] For many DIS-based exercises, entity count requirements would exceed achievable capabilities. One of the most successful DIS-based exercises in terms of entity counts was DARPA's STOW-E (Europe) exercise; only achieving an entity count around 2800.

of simulated objects. The models and algorithms that need data on the state of these objects will typically also require more processing time (CPU cycles). Many present day DIS and HLA simulations are actually CPU and/or memory-bound processes.

The models and studies discussed in the previous paragraphs are based on run-time monitoring and measurements of existing parallel simulations. The results were obtained from monitored data in empirical performance studies. The DIS and HLA communities have also attempted to use predictive models to assess simulation performance. In [56] Guha and Bassiouni assert the use of Petri-nets as a predictive analytical tool for the performance evaluation of an HLA-based run-time environment. This is preliminary work, however, and the aim of this tool is to identify reusable simulation components that can be combined into a single Petri-net module, reducing the size of the net's reachability graph, and thereby reducing the size and complexity of the solution. The reduced size of the graph maps the resulting implementation into fewer software components and could enhance the run-time performance of the simulation. Srinivasan and Reynolds propose simulation as an alternative technique to predict run-time performance [57,58]. They have developed a simulation model to allow HLA federation designers to conduct first order performance analysis before constructing the federation. The defining characteristic of their model is that the semantic information about individual federates is not part of the model. The model focuses on resource usage and contention and ignores application level details. The system consists of executing federate objects and their interactions based on probabilities. Similarly, overheads for communications (message transmission) and the RTI are accounted for using probabilities. The use of probabilities can be a limiting factor when configuring the model for analysis since the use of HLA is newly emerging and many of the run-time characteristics of the model components are unknown. Additionally, the simulation model does not include semantic information about federate behavior which can dynamically change simulation performance based on the stochastic nature of the federate object behaviors. What is needed is empirical data to validate the model's execution probabilities; this need provides incentive for the performance monitoring proposed in this research.

Different strategies have been proposed and used for the monitoring and measurement of distributed simulations. The goals of the monitoring are often to observe performance in a very limited area of interest, so the monitoring techniques tend to be tailored specifically to study only one aspect of the distributed simulations behavior. Nair, McGregor, Root, and Barth present an architecture for the monitoring and analysis of networking control components (software) used to control bi-level multicasting, Quality of Service (QoS), and network overload management (flow-control) of a DIS communications architecture [59]. The relevant performance data gathered by an SNMP "sub-agent", is passed throughout the network using SNMP.[4] The paper suggests that reuse of this architecture for communicating higher-level simulation performance data is feasible. Sudnikovich proposes the standardization of two new Protocol Data Units (PDUs) in the DIS standard, a Global Data Query PDU and a Global Data PDU [60]. The formats are primarily intended for requesting and receiving entity level performance data as it relates to the use of the lower levels of the network protocol stack (e.g., the transport layer, network layer, and down to the physical layer). Intrusiveness of the use of these PDUs could be significant, as many requests for performance information will reduce network bandwidth available for the transmission of more critical PDUs. The paper does present some solutions and proposals for research to control the level of intrusiveness of the new PDUs.

Unlike many PDES where success is based solely on the speedup attained, the diverse set of DIS domains (e.g., training, test and evaluations, planning and control) typically use a broader set of success measures. Making decisions and assessments about the effectiveness or utility of an exercise, or the hardware and software configurations of the DIS, requires analyzing the many factors contributing to the run-time performance of the simulation. This need warrants the development of a suite of techniques and tools to

[4] The Simple Network Management Protocol, or SNMP, is a network management protocol based on the exchange of information using messages to monitor and control network events (e.g., terminals or workstations joining or leaving a network).

analyze such functions as simulation control, network and protocol analysis, and simulation event analysis. In [61], Stender discusses simulation management for large exercises. The paper defines exercise managers' responsibilities including monitoring the exercise execution (performance information), identifying and resolving problems, and providing information and guidance to the program manager or exercise sponsor who, in turn, will provide guidance and instructions on assuring the exercise will meet its objectives. The paper also introduces an interesting high-level performance metric, a network breaking point that assigns a threshold to identify network saturation.

The need for an effective framework for DIS exercise management and review is significant enough that the IEEE has proposed a standard, the "IEEE Recommended Practice for Distributed Interactive Simulation - Exercise Management and Feedback", that is presented along with some suggested improvements in [62,63]. A tool for DIS exercise support and feedback is discussed in [64]. The tool requirements are for an effective real-time or near-real-time monitoring tool that is similar in functionality to the utilities used for DoD After-Action-Reviews (AARs). The tools should gather simulation trace data used for data analysis including assessment of resource utilization, simulation participation and interoperability analysis. This paper also concludes that the adoption of HLA/RTI (High-Level Architecture and Run-time Infrastructure), the Common Object Request Broker Architectures (CORBA), and multicasting, are going to significantly complicate performance monitoring tools and utilities. Information about CORBA is presented by Mowbray in [65].

Since the culmination of the DARPA STOW ACTD in 1999, related research has included initial design and development efforts for an HLA development tool suite, RTI performance studies, and other studies related to HLA federation performance. Designing, building, and using an HLA federation is complex and as in any software development project, a method for automating the data exchange among the various development activities is desirable. In [66], Hunt introduces a proposed suite of tools that are either being prototyped or have been identified as needed to support the complete life-cycle of an HLA federation. Among them are: 1) a performance modeling/prediction tool to support the design phase of federation development, 2) a test tool for federation testing, 3) monitoring, runtime management, and data collection tools to support federation execution, and 4) a data analysis/post processor tool to support the analysis activities phase.

Coinciding with HLA tool development, continuing concerns regarding HLA run-time performance resulted in a series of studies focused on understanding and defining HLA RTI performance [67, 68, 69, 70]. These studies are RTI-centric and focus exclusively on the throughput and latency characteristics in terms of HLA objects and interactions. The Defense Modeling and Simulation Office (DMSO) has initiated efforts to develop a Federation Execution Planners Workbook (FEPW); a developer framework that requires the articulation of distributed simulation environment characteristics including performance requirements. This data is used to support logical and physical resource estimates for an HLA federation [71,72]. Some commercial products (http://www.virtc.com/) are also starting to appear on the market and intend to automate the federation development process as well as provide run-time feedback regarding simulation execution.

2.3 Summary of Related Research

Much of the related research presented in this section discusses the performance analysis and evaluation of distributed and parallel codes for scientific and engineering applications. Many of the problems associated with identifying, collecting, and analyzing performance information are similar, regardless of the type of distributed application. However, the complexity and diversity of components making up the software and hardware architecture of distributed simulations justifies the need for a well-structured and unified framework for performance analysis. The complexity of simulation models and the computer architectures on which they run makes the assessment of the costs and benefits of distributed simulation a difficult task and requires a diverse range of performance information. Most performance studies of distributed simulations and the tools used for analysis have been loosely-coupled and focused on one aspect of simulation performance (e.g., network analysis or queuing performance, rollback performance). What appears to be missing from the literature is a unified framework for identifying the factors throughout the entire spectrum of system characteristics affecting distributed simulation performance and a model for deriving performance information at a high enough level so as to be meaningful and useful to simulation exercise managers, configuration planners, and other decision-makers.

Advances in enabling technologies continue to improve the already successful use of DIS and HLA simulations. DIS and HLA objectives, specifically interoperability and reuse, result in increasingly larger-scale simulation environments exhibiting greater complexity. Changing technology also results in shifting performance bottlenecks among the various components comprising a distributed simulation environment. An example is the shift in bandwidth contention from the network media to the application-level algorithms that process shared data, a result of faster transmission media, more efficient routing technologies, and more robust protocols (e.g., multicast vs. broadcast technology). There is one constant; the exercise scenario and associated workload is a driving factor behind simulation performance. As reported, most of the research related to DIS and HLA performance analysis has been focused on very specific aspects and lower-level components affecting distributed simulation performance. This fact provides further justification for the research presented in this thesis, specifically the design and development of a framework for performance analysis addressing the diverse set of physical and logical distributed simulation resources and suitable for acquiring and presenting performance information to people who design, development, configure, and control DIS and HLA simulation environments.

SECTION 3

A FRAMEWORK FOR CHARACTERIZING DIS AND HLA SIMULATION PERFORMANCE

The DoD distributed simulation architectures presented in Section 1 have proved effective for augmenting systems studies in research, systems acquisition, and training environments. We have also seen how the complexity of these systems makes it difficult to understand the effect of integrating new M&S technologies. What is needed to improve our understanding and management of DoD distributed simulation environments is a framework for characterizing performance. This framework provides a performance abstraction; a set of performance measures that convey what is important from the decision-makers perspective. This section discusses, based on lessons learned during this research, those aspects of performance that are most meaningful and useful to different decision-makers involved with the distributed simulation life-cycle. It provides a taxonomy of relevant performance measures and discusses issues regarding the monitoring and measurement of DIS and HLA simulations to obtain the relevant information.

3.1 A DIS and HLA Performance Abstraction

During the different phases of the simulation life-cycle people assume various roles with different responsibilities. The roles include those of model developers and simulation programmers, configuration and experimentation planners, and system analysts or program managers. Each role may require different kinds of performance information for decision-making. Performance information gathered and used by these people during the various phases of the simulation life cycle differentiates between the distributed simulation's different physical and logical components (i.e., hardware, O/S, application, models, experimentation). Table 3.1 provides examples of questions different people might ask when trying to understand the performance behavior of a distributed simulation environment. For related questions (in each row of the table), the principle difference is the granularity of the data needed to answer the question. Data *aggregation* or data *fusion* may be required to present the information in a meaningful way. Referring to Table 3.1, a simple example of aggregation takes the data rates for different message types (useful to a model developer) and combines them to derive an average throughput (meaningful for a program manager). A more complex example combines interaction PDU rates (e.g., radio or sensor transmissions) with entity counts to provide a useful performance measure for persons trying to make decisions about how to partition a workload of interacting entities among different simulation engines. A systems analyst or project manager will generally be interested in performance information on the simulation's capabilities, resource utilization, and bottlenecks as they relate to the simulation study goals. Performance studies can provide information to assist the analyst and managers in assessing the impact of decisions made about simulation requirements and exercise goals. A model developer or programmer can use some of the same information but will most likely need additional measured data to derive more detailed information about the system's hardware and software performance. Configuration and experimentation planners also need information that allows them to assess the performance of the simulation. With the recognition that the distributed environment imposes certain constraints on the simulation's processing requirements, configuration and experimentation planners must have performance information at varied levels of granularity that allows them to understand the effects of computer, network, and other architectural components. Meaningful performance measures are useful to all of these people when making assessments about how various hardware and software components used in distributed simulation impact meeting simulation objectives.

TABLE 1

Example DIS and HLA Performance FAQs (Frequently Asked Questions)

	Program Analyst and Managers	Model and Simulation Developers	Configuration / Experimentation Controllers
Networking / Connectivity	Is network performance sufficient in terms of data latency and throughput?	What are the data rates for specific messages using specific network services (e.g., best-effort, reliable, etc.)?	What are the transmit, receive, and error rates at the network interfaces?
Modeling	What model interactions are most expensive from the perspective of compute resources?	What are the run-time characteristics for specific model (e.g., entity) interactions?	What entity models can be efficiently simulated within the same simulation engine?
Simulation	What is the simulation workload (e.g., entity count, terrain processing, etc.)?	What are the processing rates for different simulation components?	How idle is the simulation (i.e., scheduler not processing simulation events)?
Scenario Design / Workload Partitioning	Can enough entities be created to meet training objectives?	How many entities can be simulated per workstation?	How should the entity/object workload be distributed among available workstations?
Execution	How well are we using available compute resources?	What is the resources utilization for a specific workstation?	How many simulations are currently executing?
Analysis	Can we afford to collect specific types of data for AAR, Debrief, or other analysis?	What are the costs, in terms of simulation performance to collect specific kinds of data?	What are the costs, in terms of bandwidth to collect specific amounts of data at specific rates?

A performance analysis of a DIS or HLA simulation, or some component therein, requires an initial specification of the performance requirements of the system and the consequent identification of relevant metrics for quantifying run-time performance. Although the analysis objectives for different DIS and HLA simulation environments invariably result in different sets of useful metrics, experience has shown that a set of metrics exists which have broad application to many decision-makers needs and is defined by the set of physical and logical resources in a typical distributed simulation environment. The existence of, and the interactions among, these resources define metrics

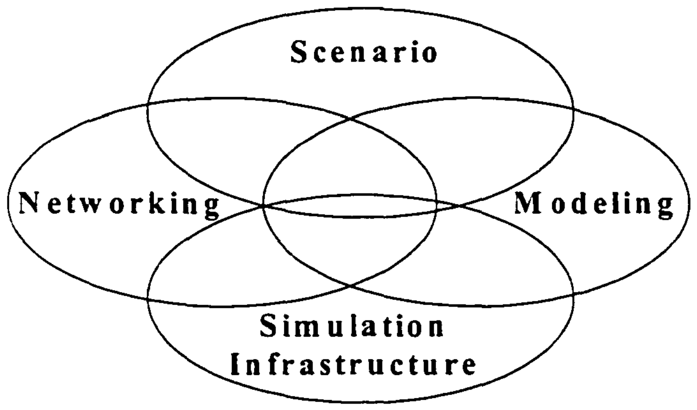


Fig. 2. Distributed simulation components used to define useful metrics.

useful for analysis, independent of the application domain. Figure 3.1 presents a diagram showing a conceptual partitioning of the major functional components required for a distributed simulation: 1) a networking component, 2) a simulation infrastructure component, 3) a modeling component, and 4) a scenario or workload. Useful metrics might come from the union of all components, or just a subset of the metrics (as defined by an intersection of the components).

3.1.1 Network Performance Factors

Modern networks have been modularized and built into a complex, yet reliable, communications infrastructure based on a fairly mature set of standards. It would be too costly to reconstruct this networking infrastructure for every distributed simulation project; therefore, current distributed simulation environments bank heavily on the reuse of these networking "toolsets". The fact that the networking components are reused means many questions can exist regarding performance within the context of a new application and workload, so network performance monitoring and analysis is useful to support decision-makers. Useful network performance data include bandwidth, latency, and error rates as they relate to any of the different low-level networking technologies (protocols) used in DIS and HLA environments: ATM, TCP/IP, Multicast IP, FDDI, or

Ethernet, to name a few. A meaningful presentation of the metrics (e.g., kilocells/sec., packets/sec., etc.) is dependent upon the level at which the protocol is being examined and by the end user of the performance information (decision-maker).

For HLA federations, an RTI is a mandatory network component (often referred to as middleware) and metrics related to its performance are of significant interest, especially in the context of a large-scale distributed simulation exercise. From a performance viewpoint, the RTI acts as a conduit channeling and regulating the workload effects between the networking component, the simulation (federate) infrastructure, and the modeling component. For some HLA application scenarios, tens of thousands of entities may exist and require high throughput rates for attribute updates (or reflections). Entities may be dynamically created and destroyed at bursty rates, pushing the capabilities of the RTI Declaration and Ownership Management services. Frequent calls to advance the simulation clock could have adverse performance impacts when using the RTI Time Management services. A common set of metrics useful for understanding the impact of an RTI implementation can be derived from measurements related to a) data transmission, and b) processing overheads associated with RTI. Throughput, bandwidth, and latency descriptions are also useful for characterizing the performance of data transmission at the RTI level. Additionally, packet and message loss data are important measures to support utilization assessments about the underlying network infrastructure and any potential resource contention within the RTI implementation (e.g., buffering, packet bundling, etc.). Timing data related to the relative processing costs of an RTI are useful for persons trying to understand the overall costs associated with using HLA/RTI to share data. The RTI intrinsically maintains some performance information in its Management Object Model (MOM) data. MOM data can provide useful information on the operation of the RTI, individual federates, and the integrated federation.

The bandwidth, latency, and error rates are relevant throughout the entire protocol stack of DIS and HLA simulations. Within the DIS and HLA, end-to-end delays (latencies) are especially significant due to the impact on valid object interactions (time delays and event reordering). Unacceptable end-to-end delays can be a result of contention for the transmission media or in the case of a wide-area network, the length of transmission (i.e., speed of light laws). Networked simulations have additional factors affecting network performance not directly related to the physical transmission of the data. An example is the encoding and decoding (i.e., data packing, byte ordering, etc.) of messages passed among machines; a process adding to end-to-end delays and complicated by the nature of heterogeneous networks and compute platforms used in DIS and HLA-based simulation environments.

The low-level functionality in the network component (as it relates to the protocol stack) provides a desirable place to filter and format data and provides control-flow mechanisms. Data rates and packet sizes are used to design buffer space and account for compute cycle requirements. Decisions must be made regarding interest management (filtering), bad data (sending and receiving), exercise control, timing requirements (synchronous vs. asynchronous), network partitioning (bandwidth reduction), and network configuration (hardware and software topology). Networking performance metrics are essential to provide meaningful and useful information to developers, configuration planners, and other persons addressing these issues to connect DIS and HLA simulations as well as other distributed applications.

3.1.2 Simulation Infrastructure Performance Factors

Metrics associated with simulation infrastructure help characterize the run-time performance of simulation processing overheads or the costs of doing simulation. Typically, DIS and HLA simulations are scaled, time-stepped; or discrete, event-stepped implementations. DIS and HLA simulation environments (or federations) may consist of simulations (or federates) having similar or dissimilar architectures. An example would be an HLA federation consisting of engineering design models implemented as discrete event simulations interoperating with a fixed-time increment, analysis simulation used to assess the impact of various design alternatives. Regardless of the type of simulation, a set of common functional components is used for time management, events and event schedulers, entity management, simulation administration and control, graphics and visualization, and support routines for data collection (including performance measurements). DIS and HLA time management functions are complicated in that each of the computers participating in the distributed simulation must invoke events in a globally coordinated or synchronized manner, assuring either correct program behavior or the recovery from errors induced by uncoordinated activity. The schedulers must execute efficiently, independent of the number of events that must be serviced as the size of the simulation experiment is scaled up or down. As is the case for most simulations, maintaining event lists, entity *databases* and other data structures used in DIS and HLA-based simulations can greatly affect performance so the cost of maintaining, synchronizing, and accessing these structures must also be considered when characterizing performance. Graphics displays and visual systems capabilities must be weighed against the modeling and scenario complexity. Performance measures related to rendering and displaying graphical images, as well as any network traffic associated with remote visualization are useful.

All simulations have some cost associated with initialization and start-up, termination and cleanup operations, and general simulation administration, control, and reporting. Having these services replicated, executing concurrently, and coordinating operations and decisions, affects performance. The control of a distributed simulation is a characteristic that is pure overhead, part of the cost of doing simulation, and must be considered when understanding the performance of a distributed simulation. Additionally, for some simulations, graphical displays and high levels of user interaction and intervention can be intrusive to the simulation; the impact on performance must be considered. Lastly, the library and utilities associated with data collection in a distributed simulation environment are diverse and application dependent. The effect on performance can be dramatic depending on the granularity or volume of data collected.

When used in conjunction with performance data from the other defined components, simulation infrastructure metrics provide an intuitive feel and, at some level, a quantitative assessment for whether the workload is too excessive for a given simulation implementation. This is useful as feedback for testers and configuration and workload planners. Specific metrics such as number of events processed per second, state update rate, scheduler idle time, and total simulation time are useful for developers to identify implementation deficiencies or problems, and to provide real-time feedback regarding the *health* of the simulations executing in a distributed simulation environment.

3.1.3 Modeling Performance Factors

The previous two subsections describe lower-level, quantifiable measures that most directly relate to the performance of specific components of the simulation architecture. Performance measures related to these components provide useful information for assessing the utility of certain enabling technologies (networking) or the reasonableness of a specific implementation (simulation infrastructure); however, to support assessments about the success or achievement of simulation goals, networking and simulation infrastructure performance data must be interpreted in the context of specific modeling and scenario workloads using terminology meaningful to decision-makers. Performance assessments at this level can be complicated because of the multi-variate relationships among interacting simulation components and non-linear performance behavior in the presence of changing model and scenario workloads.

DIS and HLA-based simulations are predominately used by the U.S. DoD, so most of the scenarios consist of entities (e.g., tanks, aircraft, dismounted infantry) and environmental (e.g., clouds, dust, rain) models used to create a synthetic battlespace. The CGF and Synthetic Natural Environments (SNE) may have different levels of fidelity and resolution leading to very different run-time performance characteristics. CGF model design and implementation issues affecting run-time performance include physical modeling of entity motion (within its own coordinate system and within a world coordinate system), sensor modeling, weapons handling, behavioral modeling, and visualization. Entity motion model performance is dependent upon whether the models are physics-based or state-based. When different or multiple axis coordinates are used to simulate entity dynamics (i.e., body-axis coordinates for equations of motion and earth-axis (world) coordinates used to describe the kinematics), consistency and accuracy among algorithms and databases for entity position and orientation transformations are critical. Frequent transformations can significantly impact model performance.

Another factor that can affect run-time performance is related to dead-reckoning. Different thresholds may be required for vehicle models that exhibit very different movement characteristics (i.e., slow-movers vs. fast-movers). High-performance DR algorithms lead to more accurate representations with greater computational costs. Lower DR algorithms provide lower computational costs but can lead to anomalous or invalid simulation performance in terms of visualization, simulated collisions, and other model aspects. Factors affecting CGF sensor modeling include the capabilities of the simulated sensors, the size and complexity (temperature, density, humidity, weather, dynamic environmental effects, etc.) of the synthetic environment, and the number of sensors the simulated entity is modeling. The same factors are relevant to any simulated weapons processing as well.

Cognitive and other behavioral modeling is another performance factor. It has been recognized that an essential aspect of CGF realism is the use of accurate behavioral representations that provide context for CGF physical representations. Different behavioral models will have different computational requirements (performance), as is the case, for example, between the SAF taskframes (lower cost) and the TacAir-Soar mechanisms (higher cost) discussed in Section 1.2.2.2. Taskframes are based on a finite-state machine mechanism and TacAir-Soar uses the Soar architecture and programming language, an actual Artificial Intelligence (AI) cognitive model [73].

Environmental modeling is the representation of the simulated world or the gaming area in which entities operate and interact. Terrain modeling (land and ocean) can have extensive processing and memory requirements; dependencies include the number of static versus dynamic representations that are used to create the terrain, the consistency requirements within the distributed simulation environment (network performance impact), and the internal representations within a specific simulation implementation. SNE run-time performance measures can be related to algorithms (temporal) or data (spatial) and are affected by such design decisions as a homogeneous (globally consistent environmental state) or gridded (e.g., different weather conditions in different regions of the simulation playbox) models, or the fidelity of physics-based implementations (barometric pressure effects; viscosity and water movement; energy propagation such as shock waves, sound, light, medium variability; and discontinuities such as temperature or salinity).

Important performance considerations must be given to the interaction among the CGF and SNE models. The complexity of the interactions is the principal reason why assessing or predicting performance of DIS and HLA simulations is so difficult. Possible interactions between entities/environment and environment/environment must be defined and performance measures used to determine the impact on performance. Examples include line-of-sight algorithms critical for air/land/entity interactions, and *traffic models* to support realistic movement of entities on terrain including effects such as slope, surface type, and effects of recent weather. Other factors affecting performance can include the use of dynamic terrain (e.g., craters, tidal flow) and multi-state objects used to show the results of weapons interactions and create animation sequences for the visual system (e.g., buildings blowing up).

3.1.4 Scenario Factors

The use of performance measures related to the conduct of a scenario is useful for many purposes including: 1) utilization of existing resources in the synthetic environment, 2) planning for growth in the presence of changing technologies, and 3) evaluation of "level playing fields" among interoperable simulations. DIS and HLA scenarios are necessarily constrained by the technologies (capabilities and limitations) of any existing network, models, and simulation infrastructure. Consequently, scenario rehearsal is a critical factor in determining the "proof of the total system" (architecture and configuration). This requires making sure the available resources are sufficient and the run-time performance of all components is adequate for a given simulation exercise. Performance information characterizing network, simulation infrastructure, and modeling capabilities support assessments by decision-makers regarding the feasibility of a specific scenario and the workload resulting from that scenario. Useful performance metrics related to run-time performance of the distributed simulation environment include overall processor and network utilizations, bandwidth characteristics (network and application protocols), minimum and maximum model update rates, and sensitivity metrics addressing the impact of changing workload due to the scenario.

Two significant factors to consider during scenario design are the types of simulated entities to be used and their expected interactions. Performance data can be captured and used to provide insight on the processing requirements of specific entity types. Performance assessments can then be made to estimate overall scenario performance based on the number of entities and their expected interactions. This approach allows the capabilities of the networked system (network, simulation, modeling capability) to be properly paired with scenario objectives and supports evaluation of scenario design compatibility with the physical and logical configuration of the DIS and HLA environment. As an example, in a SAF-based simulation various models of real-world physical systems (hull, sensors, weapons, etc.) are combined as *sub-models* to make up the entity representation of a vehicle. Each of the sub-models will have known processing requirements and when combined in a run-time context will define the overall processing requirements of the entity representation. A real ship has a number of sensor systems and the associated SAF representation of that ship will have the corresponding number of sensor sub-models. If the ship model is used in a large-scale scenario, the model may spend all of its time processing the shared (distributed) data inputs to the sensor models (e.g., emissions, radio packets), which limits the number of ships that can be simulated on a single simulation engine. As a result, tradeoffs must be made in scenario design regarding the size of the synthetic environment (number of entities) and the types of entities that are a part of the scenario.

In WAN environments or environments that integrate DIS and HLA simulations with real-world systems, networking limitations can also constrain scenario design. For example, in one recent distributed simulation exercise, a transoceanic, 128Kbyte ISDN line was used to link an HLA federation with the combat systems used onboard a U.S. Navy AEGIS class ship. The HLA federation modeled a Theater Ballistic Missile (TBM) threat and passed the simulated entity position and orientation data over the ISDN line as input to the ship's sophisticated tracking radar and fire control systems. To properly stimulate these real-world systems, the model was required to update the simulated TBM's position and orientation state data at a minimum 5 Hz rate. In this case, the constant, known size (data volume) of the ground truth data (TBM position and orientation), the specified model update rate, and the limited capacity of the network were all factors that imposed constraints on the size and complexity of the scenario used for this simulation exercise. In this case, by previously monitoring and understanding the processing requirements of the TBM model, it was determined that a maximum of two TBMs could be successfully simulated at any instant in time..

Performance measures related to the conduct of a scenario can also be used to assess the adequacy of the synthetic environment in providing a "level playing field"; otherwise interactions between entities may be invalid. An example could involve the interaction of a CGF Fixed-Wing Aircraft (FWA) and a virtual FWA simulator (man-in-loop) acting as an opposing force (air-to-air threat). If the performance of the network and CGF simulation engine is such that latent entity state data is used to update the image generator in the virtual cockpit simulator, it is possible that the trainee in the virtual cockpit will have an incorrect perception of the actual position of the CGF FWA, while the CGF FWA has a more accurate perception of the virtual cockpit simulated FWA. This creates an unnatural advantage favoring the CGF FWA and invalidates any interaction the two entities might have in the synthetic environment (e.g., decision to engage). Performance information (real-time and post-analysis) can be useful for assessing the validity of certain scenario workloads and can be useful for adjudicating invalid interactions similar to the example described above.

3.1.5 Generalization

Typical hardware and operating system performance information in distributed simulation applications, like CPU and network utilization, are easy to identify and measure using traditional tools. Obtaining other meaningful application specific performance information requires alternative methods, deriving the information from the composition of data characterizing run-time performance of individual simulations and global data on the structure and performance of the entire distributed simulation environment. The perception of both local detail and global structure provides both fine and course-grained views of distributed simulation performance and can be used to understand the effects of decisions made regarding model design, implementation, and scenarios (run-time workload). Traditionally, most performance measurement and analysis methods for distributed simulations have been application specific and primarily focus on providing information meaningful to the software developer. For non-DoD applications, the goal of simulation speedup dominates most of the literature. For many DoD applications (e.g., synthetic training environments), the number of entities per workstation (where an entity is defined as some simulated real-world object) is also used as the singular metric. These two metrics are based on a set of criteria that indicate achievement, or not, of a particular performance goal but do not articulate the reasons for the observed performance and behavior. Decision-makers need a broader array of performance metrics than speedup or entities per workstation, yet the number of components that make up a distributed simulation and the complexity of interactions among the components make identifying all the performance factors a difficult task.

Consider a hypothetical case of an HLA exercise manager being told that a simulation engine modeling five aircraft is performing poorly. The following metrics convey increasingly greater amounts of information:

- Entites per workstation - not meaningful in this case because the number of locally simulated entities is low and the number and the types of remote entities is variable.
- Packets per second - useful for detecting the overload condition but does not provide the manager with enough information to help resolve the problem.
- Reflections per second - allows the exercise manager to know if the RTI is passing a large amount of state information to the application models and is able to associate the performance degradation with model activity.
- Radio updates per second – indicates the rate a specific model, a radio in this case, is receiving and processing data.
- Radios per aircraft – indicates that number of radios an aircraft has associated with its representation.

The exercise manager could use the above metrics to detect and identify performance problems due to a specific aircraft receiving inordinately large amounts of radio traffic. This information could be used to make judicious decisions about how to partition the workload (simulated aircraft) to minimize the impact of radio traffic. An important point about the metrics listed above is they are presented in terms meaningful to the exercise manager who needs to understand the performance impact of a specific scenario workload.

To improve the benefit of distributing a simulation's computations and run-time environment, it is necessary to understand the simulation's performance characteristics in the context of the distributed environment. The characterization of distributed simulation performance is multi-faceted. Decisions about concurrent execution of simulation models using replicated resources and services are based on abstract models of computation, and performance characterizations of this system can be difficult to understand. Speedup and entities per workstation, however, are not sufficient for a full characterization of distributed simulation performance. This is especially true when distributed simulation is used in training and other man-in-loop environments. Performance information relates not just to the speed of computation, but to the efficiency and effectiveness of the simulation in utilizing the shared resources and services within the distributed environment. Performance information characterizing concurrency, scaling, availability, reliability, and interactivity can articulate the simulation's ability to meet the many goals of a simulation study.

Characteristics of simulation architectures, simulation models, scenarios, and distributed systems provide a basis for identifying factors affecting distributed simulation performance. A well-defined framework for performance analysis can provide techniques to establish relationships among these factors and the constructs and abstractions used in the simulation's design and implementation. The intended use of the performance information defines which metrics are useful to develop an understanding of the simulation's performance. The level at which the performance information is presented is determined by the role and intended use of the information. This establishes the motivation for the development of an effective framework for analysis of distributed simulation performance.

3.2 A Taxonomy of DIS and HLA Performance Measures

The previous section highlighted the significance of different components affecting distributed simulation performance. At run-time each of these components manifest themselves as a set of diverse, yet interrelated, performance measures and for this discussion it is useful to classify and organize the performance measures into a meaningful hierarchy. The taxonomy shown in Figure 3.2 is not necessarily inclusive of all measures but provides a logical organization based on the conceptualization discussed in the previous section. The organization of these measures is useful for persons trying to understand the scope of analyzing the run-time performance of DIS and HLA simulations. It provides an initial definition of performance measures useful for making decisions about designing, configuring, and controlling the simulation environment.

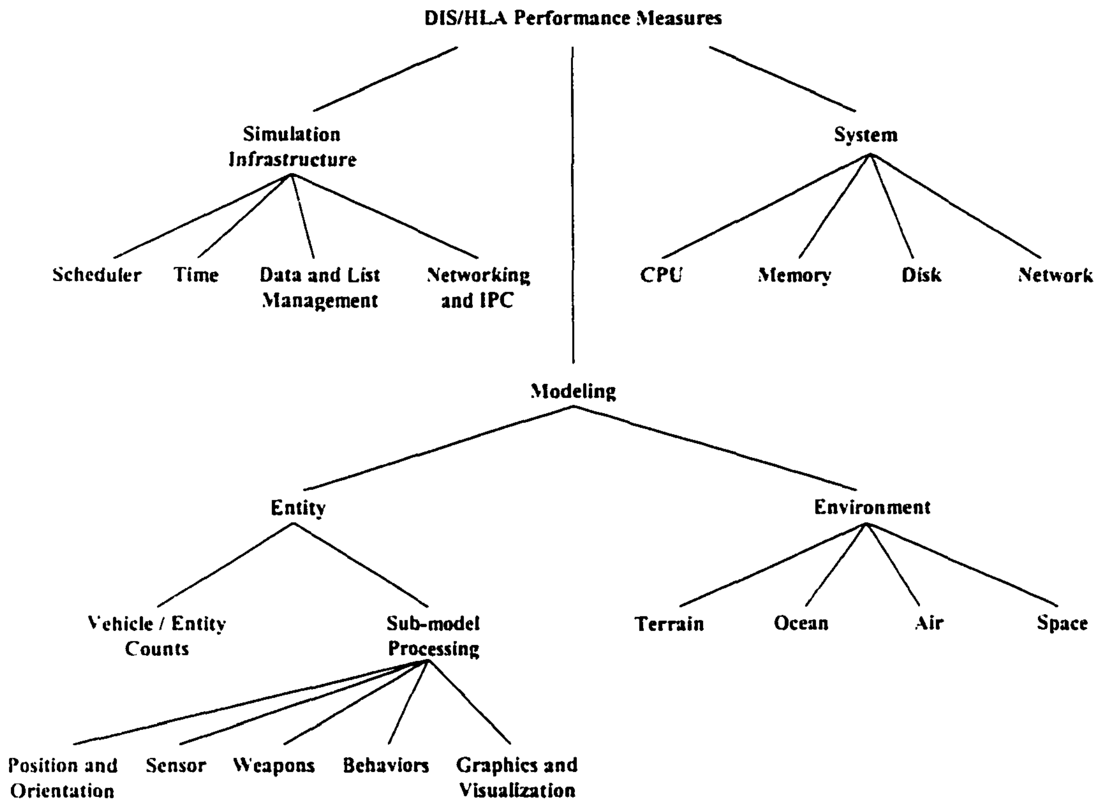


Fig. 3. A taxonomy of DIS and HLA simulation performance measures.

These measures, when considered with the capabilities and limitations of enabling simulation technologies and scenario effects, provide a model for structuring the monitoring, data collection, and analysis process.[5]

Meaningful performance information related to the components discussed in Section 3.1 is associated with performance measures in three top-level categories: 1) modeling, 2) simulation infrastructure, and 3) system. Modeling performance measures are related to simulated entities (e.g., vehicles, platforms, etc.) and simulated environment (e.g., terrain

[5] Depending on the objectives, the analysis may consist of a series of interacting or collaborative activities.

representations, bathymetry, etc.). Specific to entity modeling, it is important to provide an *Entity Count*, the number of each entity or vehicle type (air, ground, and water vehicles, mines, etc.) in the simulation environment. This count data should be available on a per application basis, and as a set of global metrics (the entity count of the entire simulation environment). Entities (vehicles) are a natural data point for persons involved with DIS and HLA simulations because of their direct correlation with the real-world objects and activities, especially in the context of DoD and military environments. Related to each run-time instance of a simulated entity, is its sub-model processing requirements. A frequent question asked by persons developing or configuring DIS/HLA simulation exercises is what are the processing costs associated with a specific entity type. The *Sub-model Processing Times* provide timing data that characterize the processing costs of each component used to represent specific entities. For DIS and HLA simulations, experience has shown that it is reasonable to view the synthetic environment as a set of entities that may or may not move around, sensing and interacting with their environment, and that are potentially capable of engaging other entities with weapons. This view allows an entity sub-model categorization of: 1) position and orientation models (dynamic representation of the entity such as flight dynamics or kinematics), 2) sensor models (e.g., visual, radar), 3) weapons, 4) behaviors (e.g., following a route, flying a formation), and 5) graphics and visualization. A meaningful way to present sub-model processing times is as a relative percentage of the over-all simulation processing time for some specified time interval, including the elapsed simulation time. It is possible to collect the sub-model timing data on a per entity basis or on a per application basis, depending on the level of detail required for the analysis.

Modeling SNE entails providing realistic representations of the ground, air, sea, and space domains. High-fidelity SNE can improve the fidelity of entity modeling but can also adversely impact application processing and network performance. In the context of DIS and HLA, dynamic high-fidelity SNEs are an emerging technology. A proven paradigm for architecting environmental models within a distributed simulation system is to have a centralized network server [31]. An environmental server typically uses compute-intensive, physics-based volumetric modeling techniques. Performance measures related to environmental modeling include basic *timing measurements used to understand the compute costs of specific environmental models* (e.g., fractal-based cloud models, smoke models, and water-column models). More important performance measures are related to the impact that environmental modeling has on the overall distributed simulation environment; specifically, how the receiving entity models and their respective sensors process the environmental inputs, and the effect the potentially voluminous data has on the network resources. The real-time (or faster-than-real-time) constraints of many DIS and HLA simulations may further hinder performance. Important environmental performance measures related to entity modeling are the time it takes entity sensor models to perform intervisibility (*Intervisibility Lookups*) and collision avoidance (*Collision Lookups*) calculations. These can be potentially expensive algorithms and timing data supports decisions regarding modeling tradeoffs in terrain fidelity and entity model processing requirements. Other performance measures related to environmental models and their performance impact are the *Total Number of Environmental Objects Transmitted* (uniform- and gridded-weather), *Environmental Object Update Rate and Variance*, *Average Time to Transfer* (bytes/sec), *Average Object Size Transmitted*, *Re-transmit Requests* (specifically for new simulation joins), *Total Number of Bytes Transmitted*, and static measures dealing with the *Size and Extents* of the environmental data sets.

Performance measures suitable for describing simulation infrastructure include characterizations of: 1) the simulation scheduler or executive, 2) the networking and Inter-process Communications (IPC), 3) data and list management, and 4) general timing information. *Scheduler Idle Time* provides a measure of the scheduler workload as it relates to servicing events. *Slack in the Entity Tick Rate*[6] provides a relative measure of

[6] A metric derived from the notion of real-time process scheduling. This metric is applicable to simulations requiring periodic updates to all entities within a specific time period. Slack is the time remaining in the time period after all entities have been updated. As processing times for entities and other state information increase, the amount of slack decreases.

the amount of available room for additional entity state processing. The two metrics above are especially important as they relate to DIS-based real-time or scaled, real-time simulations. They provide quantitative measures that can be used to relate scenario workload to the perceived run-time performance of the synthetic environment. If workload becomes too great such that the simulation processing cannot keep up, the scheduler will not be able to keep up and simulation time will begin to lag behind the wall-clock (real-world time). *Entity Tick Rate* describes the average update frequency for each instance of entity and *Entity State Update Performance* provides a measure of the overall update performance for all simulated entities within the simulation. Metrics related to interprocessor communications using the HLA RTI include *RTI Tick Time* (time spent in the RTI), *RTI Tick Rate*, *Attribute Updates/Reflections* (per second), *Object Interactions* (per second), *Average Size of Attribute Updates*, *Attribute Update /Interaction Delay*, and *Time Advance Request Delay*.

System-level performance measures are related to the utilization and performance of the operating system services and hardware resources on the computers running the simulations (applications). These measures map to the processing, memory, and networking components used in any general distributed computing environment and provide a low-level performance characterization of the simulation process and the computer on which it is executing. Metrics include relative *CPU and Memory Utilizations* for the application and the system. One potential artifact of the high fidelity, high-resolution representations used in many DIS and HLA simulations is very large storage requirements to manage data (e.g., terrain databases) in physical memory and on disks. DIS and HLA simulations can be memory bound processes so additional performance measures include the amount of physical memory (*Resident Memory*) consumed by the simulation as well as the amount in virtual memory (*Swap*). *Hard disk activity* (blocks or bytes per second) metrics extend the notion of memory hierarchy performance measures. These measures are useful for identifying applications or simulations with inefficient working sets or memory leaks. Simulation workload characteristics such as fast-moving platforms (e.g., theater ballistic missiles) can sometimes induce thrashing situations, as the interactions between the entity and the terrain require rapid and large-scale memory updates. Networking performance measures are used to assess the impact of the simulation workload on the available bandwidth as seen on the transmission media and as seen at the computer network interface card. Standard metrics include the *Transmission and Receive Rates* (packets/sec.), *Error Rates*, and *Collision Rates*.

3.3 Performance Monitoring

An important component of any framework for performance analysis is the monitoring system used to collect data. Many issues must be considered when designing a performance monitoring system. Monitoring in the DoD distributed simulation environment requires additional considerations. The size and complexity of current distributed simulation environments require a person designing and implementing a monitoring system to be intimately familiar with many aspects of the simulation architecture. A key objective is to locate where in the simulation environment the data can be obtained for the required data abstraction level. Non-invasive monitoring gathers data only from the network and is not comprehensive due to the limited content available by looking at the network packets. Invasive monitoring allows for a more complete evaluation by looking at the internals of the simulation. Distributed monitoring systems that rely on instrumented simulation source code collect raw performance data on individual simulations; effective techniques are required to collate this information into an aggregated characterization of global simulation performance. Another difficult problem is the large number of hardware and software interactions in distributed simulations masking the direct causal relationships among specific simulation activities, model behaviors, and observed system performance. This necessitates identifying the significant factors affecting simulation entities and interactions and establishing a mechanism for correlating the run-time events with simulation outcomes.

From a practical perspective, system administration issues affect the usability, reliability, and transparency of a distributed simulation performance monitoring system. It is important and necessary to understand the status of all monitored and monitoring components. For example, if a simulation or monitor crashes, the system should be aware of and log the event. For monitoring systems where location transparency is an issue (a good characteristic for any distributed system), daemon processes can provide a mechanism to manage shared performance data. Heterogeneous simulation networks are the *de facto* result of the evolution of distributed simulation technology and its use in cooperative simulation design and development programs. As such, an important characteristic for a monitoring system is portability and/or interoperability, byte ordering being a significant case-in-point. A harder issue surrounding the interoperability of a performance monitoring system is the specification of a common class of shared performance information useful among dissimilar simulation architectures and system components; the dependency being related to any protocols associated with the monitoring system.

Specific to an HLA environment, distributed monitoring is complicated by, among other things, the interest management mechanisms intrinsic to the RTI. The specifications that restrict the visibility of certain entities and/or their attributes mean that it is possible for inconsistencies to exist among simulations' global view of the simulation execution space.[7] Another important issue is related to current RTI implementations: modular libraries linked with a simulation application (federate) during the executable build process. A standardized RTI Application Programming Interface (API) provides logical points for instrumentation to gather RTI timing data. Since access to RTI source code is

[7] In the context of HLA, interest management is also referred to as "relevance filtering" or Data Distribution Management (DDM). The HLA RTI implements DDM as a run-time services that reduces network and processor bandwidth requirements by restricting the transmission of shared data to only those federates that explicitly express and interest in receiving the data. Any distributed monitoring system that wants to have a global view of the entire synthetic battlespace will have explicitly declare its interest in the relevant performance information from all federates, which might not necessarily be available in certain segments of the network topology.

typically restricted, performance data related to the internal operation of the RTI has to be included in the HLA MOM data, transmitted periodically by each RTI during federation execution.

Another important characteristic of DoD M&S environments is the ability to scale the size of the simulations. This is especially true in the training environments where populating the virtual battlespace with more entities can enhance the realism of the synthetic environment. A performance monitoring system for DIS and HLA environments must exhibit similar scalability characteristics.

Finally, perhaps the most significant issue concerns the costs associated with obtaining meaningful performance information. Collecting performance data can be intrusive to system performance and if monitoring is required for other than the development and testing phases of the simulation life-cycle (i.e., for experimentation or production use), the cost of monitoring might be prohibitive. Therefore, a compromise must be made on run-time measurements necessary to obtain performance data, the objective being to establish a balance between the adequacy of measured data (for the purpose of analysis) and the intrusiveness of the monitoring system (and its perturbation of the performance analysis). The value of the data depends on analysis goals (performance problem) and what is currently understood or not understood about the run-time environment. As such, a mechanism to tune and control the data collection process is desirable. Timing delays induced by executing simulation code instrumented to detect and collect performance data potentially affects the correct execution of the simulation. These timing delays can be measured and their impact on simulation performance assessed. A more difficult timing error to account for is the potential reordering of simulation events among workstations participating in a simulation exercise. Assessing the impact of performance monitoring is required to establish a compromise between the volume of performance data collected and acceptable simulation run-time performance.

TABLE 2
Summary of Performance Measures

Simulation Infrastructure	System	Modeling
Scheduler	CPU	Entity
Time	Memory	Vehicle / Entity Counts
Data and List Management	Disk	Position and Orientation
Networking and IPC	Network	Sensor
		Weapons
		Behaviors
		Graphics and Visualization
		Environment
		Terrain
		Ocean
		Atmospheric
		Space

3.4 Summary: A Unified Architectural Framework for Analysis

Although performance analysis goals may be unique for different kinds of decision support, experience has shown that some common objectives exist among DIS and HLA performance studies that support the specification of the framework presented in this thesis. The commonality of analysis objectives lies in the fact that DIS and HLA simulation can fundamentally be viewed as a set of simulated real-world objects interacting within some kind of simulated time/space analog to the real-world environment where those objects exist. Understanding the run-time performance of these objects and the synthetic environment in which they exist means characterizing the simulations in terms of the number of objects (or synonymously, the number of entities), the level of activity and interaction among the objects, and their impact on the physical and logical resources used to create the distributed simulation environment.

Reemphasizing the previous discussion, a general DIS and HLA performance characterization is useful in many contexts. Performance monitoring can provide valuable information to persons trying to design and implement application models and simulation infrastructure. They use performance data for making decisions related to such things as algorithms, model fidelity, data communications patterns, network topology, timing

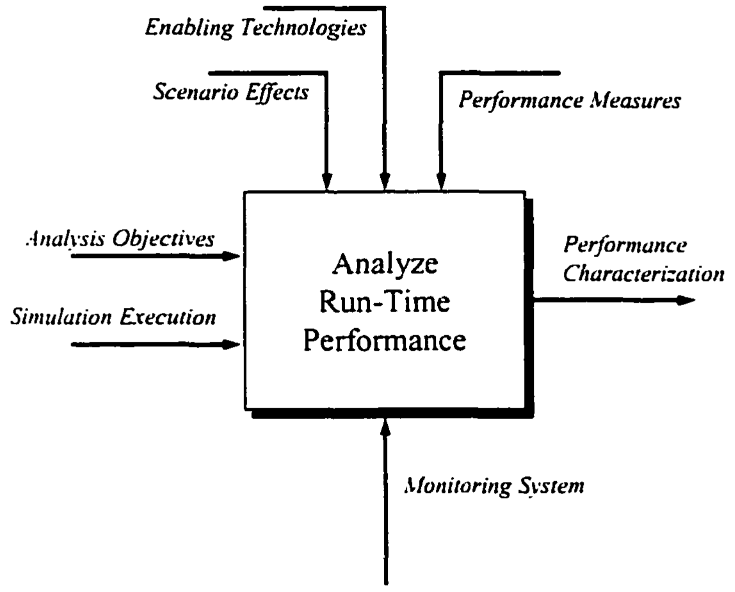


Fig. 4. Performance analysis activity process.

requirements and policies, and system scalability issues. Many DIS and HLA applications have soft, real-time requirements so performance data (i.e., timing data) is useful for testing and debugging activities, especially in the context of a complex, distributed run-time environment. Coinciding with model design and development are issues surrounding the planning and execution of DIS/HLA exercises so decisions must be made regarding hardware and software configurations. Simulation performance information is also useful to support run-time assessments regarding the simulation environment and for making near real-time assessments about the validity of the current simulation activity. Depending on the information provided, decisions can be made that include dynamic load balancing, diagnosis of simulation performance, and validity of perceived simulation activity.

The performance abstraction presented in Section 3.1 discusses the most significant factors that affect DIS and HLA performance and provides a basis for the taxonomy of performance measures that delineate the boundary of logical and physical components used to create a distributed simulation environment. The set of performance measures (summarized in Table 3.2) provide a basis for metrics selection useful for characterizing DIS and HLA run-time performance as it relates to the low-level resource and services used in a distributed simulation environment.

The process of metrics selection must also consider the capabilities and limitations of enabling technologies as well as the workload impact of specific scenarios. The relationship between low-level system and simulation performance, and the scenario workload is a critical factor to provide meaningful information to persons making decisions about the configuration, control, and management of DIS and HLA simulations and for those who might not be able to interpret lower-level performance data. Figure 3.3 shows a process model (IDEF0 [74])[8] that represents the taxonomy of performance measures, scenario (workload) effects, and enabling technologies (hardware, software, and architectural) as constraints on the performance analysis activity. Based on the analysis objectives, a set of meaningful metrics is selected and collected using the run-time monitoring system.

Figure 3.4 depicts the notion of a unified architectural framework for the design, configuration, and control of DIS and HLA simulations. At the core of the framework is the taxonomy of performance measures discussed in this section, specifically the measures related to modeling, simulation infrastructure, and the systems (compute platforms) on which they are instantiated. Data (metrics) that capture these performance measures are collected at simulation run-time by a monitoring system. This monitoring, measurement, and data collection system embodies the knowledge about the run-time execution of HLA and DIS applications. The performance information related to the run-time performance of the simulation environment is useful during the entire life-cycle of the simulation; during modeling and simulation design and development activities,

[8] Equivalent representations could be created using other IDEF representations such as IDEF4 (object-oriented design methods for client-server architectures).

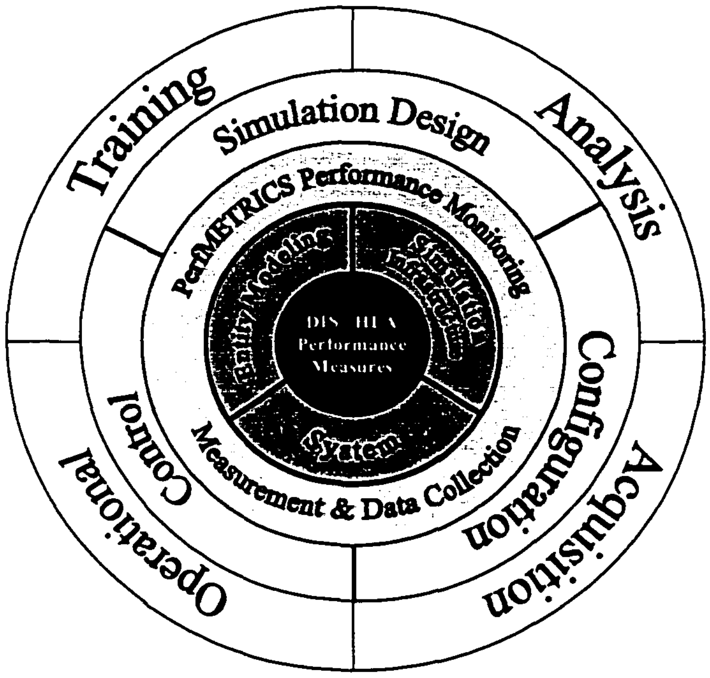

Training
Analysis
Acquisition
Operational
Simulation Design
Configuration
Control
PerfMETRICS Performance Monitoring
Measurement & Data Collection
Entity Modeling
Simulation Infrastructure
System
DIS HLA Performance Measures

Fig. 5. The unified, architectural framework for performance analysis of DIS and HLA simulations.

configuration of simulation exercises or studies, and for run-time monitoring and control of the simulation environment (especially in the context of large-scale and geographically disperse environments). The outer ring in the framework depiction represents the various M&S domains (training, analysis, acquisition, and operational) using DIS and HLA simulations that require decision support regarding the impact of simulation run-time performance. Fundamentally, the only aspect of the framework not relevant to non-DIS or non-HLA distributed simulations is the performance measures associated with physical and behavioral representations of military entities (i.e., sensor and weapons sub-models). As such, the DIS and HLA performance characterization presented in this thesis is a useful characterization in general since it is based on a high-level abstraction of components relevant to most distributed simulation environments.

The next section in this thesis continues the discussion of an integral part of the performance analysis framework; the PerfMETRICS monitoring, measurement, and data collection component. The design and implementation of this software was a significant portion of this thesis development and its role in DIS and HLA performance analysis is discussed in the follow-on section on use cases.

SECTION 4

THE PERFMETRICS MONITORING SYSTEM

As discussed in the previous section, a limited number of data collection tools exist to support DIS and HLA performance evaluations and most of the tools that do exist are focused on providing low-level, application specific performance data (e.g., network traffic). To meet the objectives of this thesis, a tool was developed to monitor and collect performance data useful for decision-makers when analyzing simulation and system performance. The tool embodies knowledge about the run-time execution of HLA and DIS applications. It is capable of identifying performance bottlenecks that can lead to errors in terms of application behavior and its interactions with the systems on which they run. PerfMETRICS is a performance monitoring tool developed to meet the following requirements: 1) provide capabilities to monitor, record, and report simulation performance data, 2) provide real-time analysis of simulation performance, 3) provide logging capabilities for post-mortem analysis of simulation data, 4) provide an infrastructure for monitoring and controlling a diverse set of HLA and DIS applications, 5) provide an implementation that is flexible and extensible, 6) provide a mechanism for controlling the monitoring process, and 7) limit the intrusiveness of the system. PerfMETRICS can be used to collect performance information to guide persons making decisions related to model fidelity and resolution, the design and implementation of simulation infrastructure, and the control and configuration of simulation scenarios and their run-time environments. A tool like PerfMETRICS is required to allow decision-makers to better understand the complexity of the DIS and HLA distributed simulation environment.

The data PerfMETRICS collects relate to the simulation model components, the simulation infrastructure, and the operating system and its application interfaces. PerfMETRICS gathers timing data from individual model components and correlates the data with specific entities or entity types. Performance data gathered on simulation infrastructure includes timing information for implementation mechanisms such as the event scheduling, idle time, entity state update performance (tick rates), and data sharing (e.g., time spent in the RTI). Operating system performance information includes CPU, memory and network utilization metrics. A detailed description of the data PerfMETRICS currently collects is found in the appendix.

Real-time analysis of simulation performance information includes the capabilities to look at the performance of individual workstations and to view multiple machines concurrently to assess the aggregate performance of all machines participating in the simulation exercise. PerfMETRICS provides a Graphical User Interface (GUI) that provides a numeric/tabular display of the relevant performance metrics and a means of showing the data as it changes during simulation run-time (time-series analysis). PerfMETRICS provides logging capabilities for post-mortem analysis of simulation data. The performance data of each of the individual simulations is saved so fine-grained data analysis, if desired, is possible. The data from all simulation/applications can be collated to provide a more global view of simulation performance. A mechanism (i.e., external utility) for reading the performance information is implemented and is capable of formatting the data so it can be imported into a diverse set of analysis software. A subset of available performance information can be selected for analysis since typically large amounts of data are available but are not relevant to the current problem of interest. This capability, the use of compiler directives to statically include or exclude instrumentation code, and the run-time capability to dynamically start and stop data collection supports tuning the monitoring and collection process to meet analysis requirements.

To provide a flexible and extensible monitoring system, the design of PerfMETRICS is loosely-coupled with the applications it is monitoring. Data links used to pass performance information are independent of the simulation application and protocols. Doing so enhances the monitoring system's usability in both DIS and HLA simulation environments. Ideally, a monitoring systems architecture should generalize to make it useful for a broad range of distributed applications used in distributed simulation environments. PerfMETRICS has been used to monitor and control various DIS and HLA applications. An artifact of application diversity is the use of the monitoring system within a heterogeneous distributed simulation environment. This results in the need for data marshalling and PerfMETRICS is designed to deal with byte-ordering differences among big-endian and little-endian workstation architectures. The PerfMETRICS implementation is written using the C programming language and uses standardized X-Windows and Motif toolkits and libraries. This makes PerfMETRICS a readily portable application on any processor/compiler architecture with an operating system that supports the use of shared memory.

As requirements for simulation performance information and simulation control change, a monitoring system should be implemented in a way that allows the relevant simulation performance and control information to be transmitted as needed throughout the distributed simulation environment. The underlying application and network protocols the monitoring system uses should be able to support the data exchange. Also, the architecture must be able to support increasingly stringent performance requirements (in the context of the monitoring system's run-time execution). As the size of the distributed simulation environment scales, the volume of performance information collected by the monitoring system must scale respectively. PerfMETRICS provides capabilities to control the rate of performance data collection, manage the hierarchy of simulation engines that are reporting and/or collecting performance data, suspend and resume the collection and reporting process, and define what low-level network resources the monitoring process uses (e.g., TCP/IP service ports, multicast group addresses).

4.1 System Architecture

The PerfMETRICS architecture (see Figure 4.1) consists of DIS and HLA applications interfacing with the PerfMETRICS monitoring system via a shared memory interface. The monitoring system has a daemon process on each workstation running a DIS or HLA

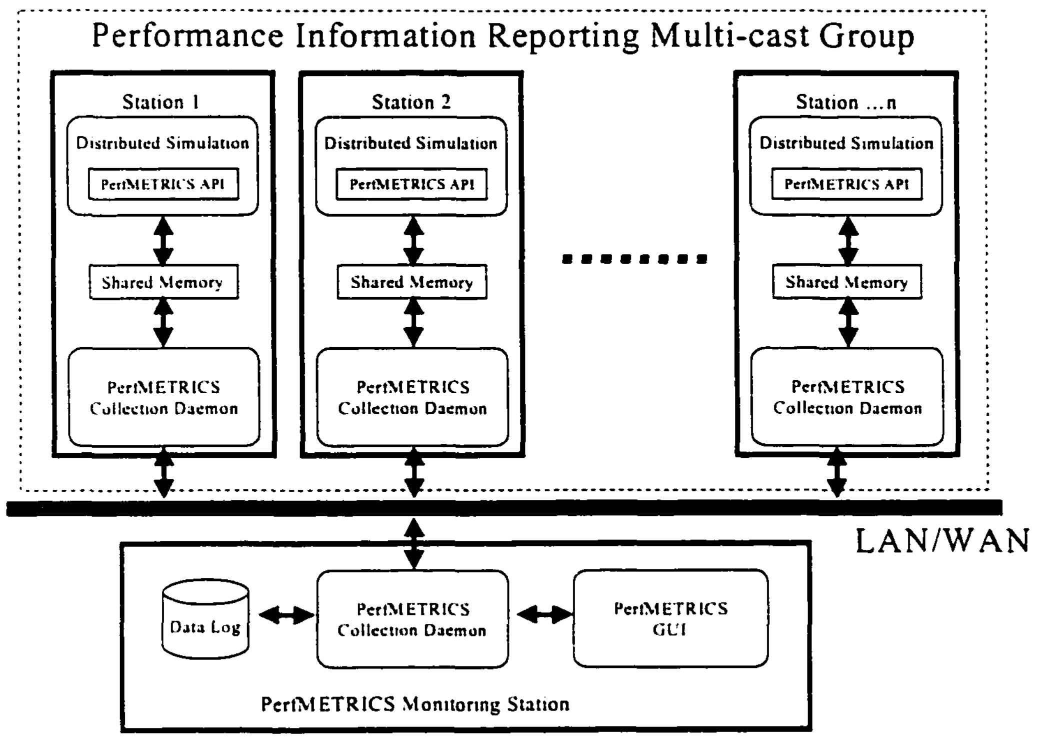


Fig. 6. PerfMETRICS architecture.

application. The daemon processes transmit control and performance information throughout the LAN and WAN networks using multicast/IP. One or more workstations designated as monitoring control stations receive the PerfMETRICS control and performance data packets and log and display the relevant information. Performance data is obtained from the application and stored in a shared memory segment, either through periodic processing or on the occurrence of specific events. This shared memory segment is implemented using standard System V IPC. The performance data in shared-memory is a static structure used by the DIS or HLA application and the PerfMETRICS collection daemon. As mentioned, the PerfMETRICS collection daemon runs on each workstation executing a monitored application (may or may not be a simulation). The collection daemon is implemented as a simple state machine. Depending on whether or not a simulation or other monitored application has attached to the shared-memory segment, the collection daemon will either be transmitting (send state) or receiving (receive state) performance data. Every daemon process has a global view of the PerfMETRICS control information being transmitted on the network. As mentioned, the daemon gets performance data from the application by reading the shared memory segment. At a periodic interval, the daemon will transmit the performance data over the network.

The PerfMETRICS system supports dynamic analysis of simulation performance by using a Graphical User Interface to display relevant performance data. Simulation performance is conveyed as entity, simulation infrastructure, and system (i.e., operating system) information. The interface can display data in time-series to help visualize the complex relationships among different factors that affect simulation performance. The PerfMETRICS system is capable of logging all performance data it receives to a binary file. The file structure is well defined; it is identical to the performance data packet structure transmitted by the PerfMETRICS collection daemon. Logging is done by the workstations designated as the monitoring control stations (whose PerfMETRICS collection daemons are in a receive state). If logging is turned on (it is optional), as the collection daemon receives a performance data packet it will write the packet to a binary file.

4.2 System Design

The design of PerfMETRICS can be viewed as three principal components: 1) the application instrumentation, 2) the collection daemon, and 3) the user interface (facilitates the presentation and display of the performance information). Performance data is obtained by hand-instrumenting the application source code. This requires specific knowledge about the application's implementation. The performance measures discussed in Section 3 are used as a guide to determine what data is to be pulled out of the application and where the instrumentation code is inserted. A library (PerfMETRICS API) is compiled and linked into the application that provides the functional interface to the shared memory segment used by the application and the collection daemon. The interface consists of both function calls and macros that may be used for things such as starting and stopping timers or writing performance event data into the shared memory segment. The instrumentation code is either turned on or off using compiler directives. This mechanism results in PerfMETRICS either being turned on or turned off at compile time but avoids the additional overheads of doing run-time conditional checks to determine whether or not the instrumentation code should be executed. As mentioned, the application type and defined performance metrics define the structure of the performance data in shared memory. An example C data structure used for the use cases presented in this thesis looks like:

```
typedef struct {
   /* structure containing the Entity Statistics Data */
   ENTITYSTATS emetrics;    /* queue of vehicles */
   /* structure containing the Simulation Statistics Data */
   SIMSTATS simmetrics;
   /* structure containing the System Statistics Data */
   SYSSTATS sysmetrics;
   /* contains simulation's version, host, exercise, PO
   database id,  and terrain database version */
   EXSTATS exstats;
   /* contains HLA ⇔ DIS Gateway performance data */
   HLAINTERFACE_STATS hlainterface_stats;
} METRICS, *METRICS_PTR;
```

During initialization the application attaches to the shared memory segment via the PerfMETRICS API, registers to use a semaphore controlling access to shared memory, and then clears the data structure. This data structure in shared memory acts as a buffer, holding the most recent performance data until the PerfMETRICS collection daemon is ready to transmit the data to the monitoring station(s). Additionally, at initialization time the application uses the PerfMETRICS API to set up signal handlers used to notify the collection daemon of the status of the application. For instance, at startup the application sets a shared variable with an enumeration defining its application type (e.g., HLAINTERFACE is an enumeration used to indicate an HLA⇔DIS Interface

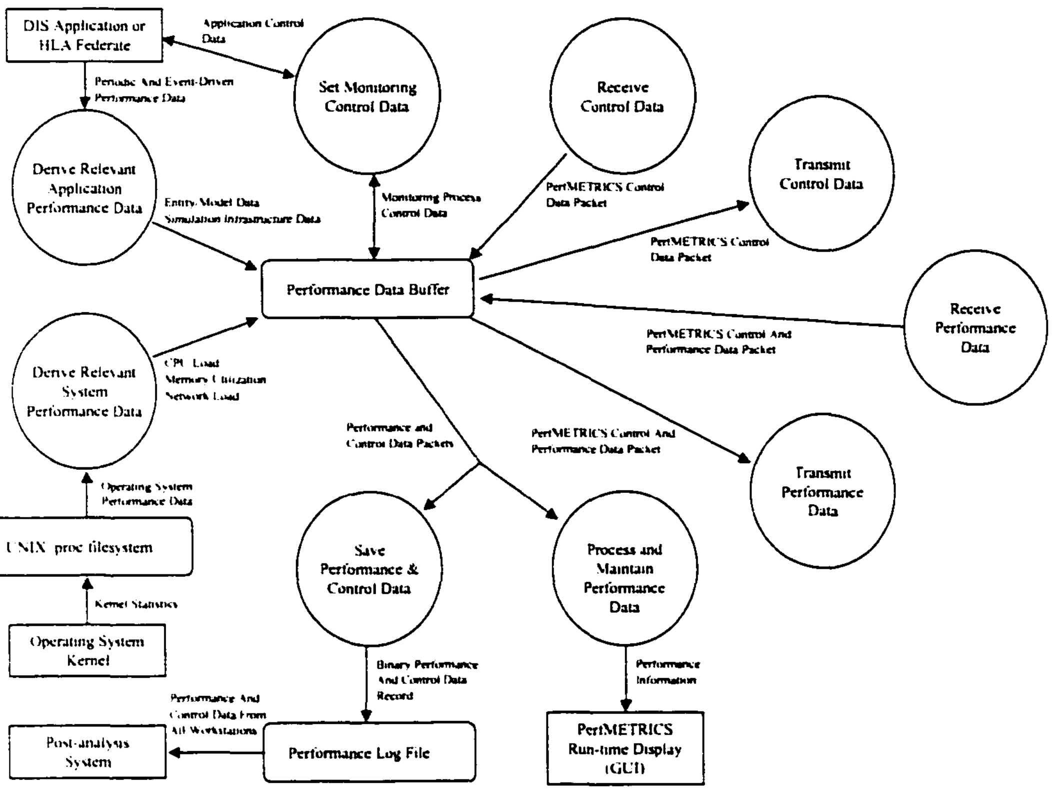

DIS Application or HLA Federate
Application Control Data
Periodic And Event-Driven Performance Data
Set Monitoring Control Data
Receive Control Data
Transmit Control Data
Derive Relevant Application Performance Data
Entity/Model Data Simulation Infrastructure Data
Monitoring Process Control Data
PerfMETRICS Control Data Packet
PerfMETRICS Control Data Packet
Performance Data Buffer
Receive Performance Data
PerfMETRICS Control And Performance Data Packet
Derive Relevant System Performance Data
CPU Load Memory Utilization Network Load
Performance and Control Data Packets
PerfMETRICS Control And Performance Data Packet
Transmit Performance Data
Operating System Performance Data
UNIX proc filesystem
Save Performance & Control Data
Process and Maintain Performance Data
Kernel Statistics
Operating System Kernel
Binary Performance And Control Data Record
Performance Information
Performance And Control Data From All Workstations
Post-analysis System
Performance Log File
PerfMETRICS Run-time Display (GUI)

Fig. 7. Data flow diagram of the PerfMETRICS collection daemon.

application) and then sends a signal to the collection daemon indicating it is ready, meaning the PerfMETRICS component of the application has been properly initialized and is ready to write performance data to the buffer. The application also catches operating system signals that cause the application to terminate, crash, or abort. This allows PerfMETRICS to properly manage the IPC and to send a control message that the application is no longer running.

4.2.1 Collection Daemon

Figure 4-2 is a data flow diagram illustrating the functional components of the PerfMETRICS collection daemon. The key daemon functions are: 1) gather the performance information from the application and operating system, 2) transmit, receive and process control and performance data, and 3) process and manage the information for logging and run-time display. The shared memory buffer provides the conduit for data flow. Shared memory was selected because of its better performance compared to other IPC mechanisms. Traditional pipe and named pipes (FIFOs) are subject to performance overheads since they are implemented using file I/O and system calls. Other drawbacks associated with pipes include the number of pipes that must be created for a client/server architecture consisting of multiple clients, explicit deletion of pipes after process termination, and the fact that data passing through the pipe can only be viewed as a byte stream that has no persistence once read. Other IPC mechanisms such as message queues and the STREAMS/socket interface offer better performance than pipes because they are implemented in the operating system kernel. However, they still are generally slower mechanisms than shared memory IPC; multiple copies and buffers of data must be managed by the kernel as the data is shared among processes.

A shared memory interface provides a fast mechanism to share data. Access to the shared region is like any other memory access, requiring no system calls to read or write data and in terms of implementing a software-based performance monitoring system it is the least complex in terms of instrumentation code. One limitation of shared memory is the lack of any intrinsic synchronization for processes reading and writing data to the shared memory. To address this problem, PerfMETRICS uses a semaphore mechanism to coordinate access to the shared memory segment. Logical communications between the application and the collection daemon are via software interrupts and data stored in the shared memory region. The software interrupts (signals) are used to establish a logical connection between the monitored application (typically a simulation) and the collection daemon. This requires their Process Identification (PID) numbers be placed in the shared memory segment. These PIDs are used by the signaling mechanism to notify either of the two processes that some event has occurred related to control of the monitoring process (e.g., change the frequency of reporting data). Table 4.1 lists the notifications that are currently used in the PerfMETRICS monitoring system. The actual signal sent and delivered via the kernel is a POSIX defined SIGUSR1 interrupt. Upon receipt of this interrupt, the collection daemon or application will examine a *Control Command* field in the shared memory segment that specifies the particular notification type. The daemon or application will then execute the required processing.

Communications between the collection daemon and the lower-level networking facility use I/O multiplexing to know when data can be read from the networking service ports without blocking. The multiplexing mechanism is implemented in the collection daemon by using a *select* system call, allowing the daemon to know which service ports are ready to be read from or written to without blocking. The use of I/O multiplexing in the PerfMETRICS design is significant because when reading or writing data from the shared memory segment, the daemon acquires a semaphore lock on that region. If the daemon blocks while trying to read or write to the network service ports, the semaphore it is holding prevents the application from accessing the shared memory region. If this is the case, then the application could spend an inordinate amount of time blocked while waiting for the semaphores, increasing the overall intrusiveness of the monitoring process. Using shared memory, signals, and I/O multiplexing reduces the overhead of the PerfMETRICS monitoring process by decreasing the time it takes to access shared data and making the most of event-driven IPC.

The PerfMETRICS collection daemon is implemented as a simple state machine capable of either sending or receiving performance data depending on the type of application with which it has established communications. The collection daemon states and transitions are illustrated in Figure 4.3. The transitions are shown as condition/action pairs. After creating and initializing the shared memory segment and its networking service ports, the collection daemon enters an initial Sleep state where it waits for an interrupt from an application that wishes to use the daemon process. When an application wakes the daemon process the Ready state is entered. In the Ready state the daemon transitions either to the Sender, Receiver, or Sleep state depending on what kind of application or process, if any, has established a logical connection with the collection daemon via the shared memory segment. If the daemon determines a monitoring control GUI or an

TABLE 3
PerfMETRICs Monitoring Event Notifications.

Notification	Generated by (at)	Seen and / or Used by
New Collection Interval	Monitoring Control GUI	All collection daemons
Suspend Collection	Monitoring Control GUI	All collection daemons
Resume Collection	Monitoring Control GUI	All collection daemons
Simulation Start	Simulation	All collection daemons
Simulation Stop	Simulation	All collection daemons
Simulation Suspended	Simulation	All collection daemons
Simulation Resumed	Simulation	All collection daemons
Simulation Crash	Simulation	All collection daemons
Start Data Logging	Monitoring Control GUI	Monitoring Control collection daemon
Stop Data Logging	Monitoring Control GUI	Monitoring Control collection daemon
Configure HLA Interface	Monitoring Control GUI	All collection daemons
HLA Interface Control	Monitoring Control GUI	All collection daemons
HCI Interface Control	HCI Application	Simulation

application that needs to send or receive performance data has awakened the daemon, the daemon transitions into the Receiver or Sender state accordingly. Once in the Sender state, the collection daemon will transmit system and application performance data along with control information. Some applications may have requirements to receive performance data, which is also possible in the Sender state. All relevant performance and control data is transmitted to members in the PerfMETRICS multicast groups. The data is transmitted at the rate specified by the collection interval, a parameter specified during daemon initialization and may be modified during run-time. Upon receipt of a monitoring control interrupt (SIGUSER1), the collection daemon transitions out of the Sender state, processes the control command and returns to the Ready state. Transitions between the Receiver state and the Ready state are the same as from the Sender to Ready state.

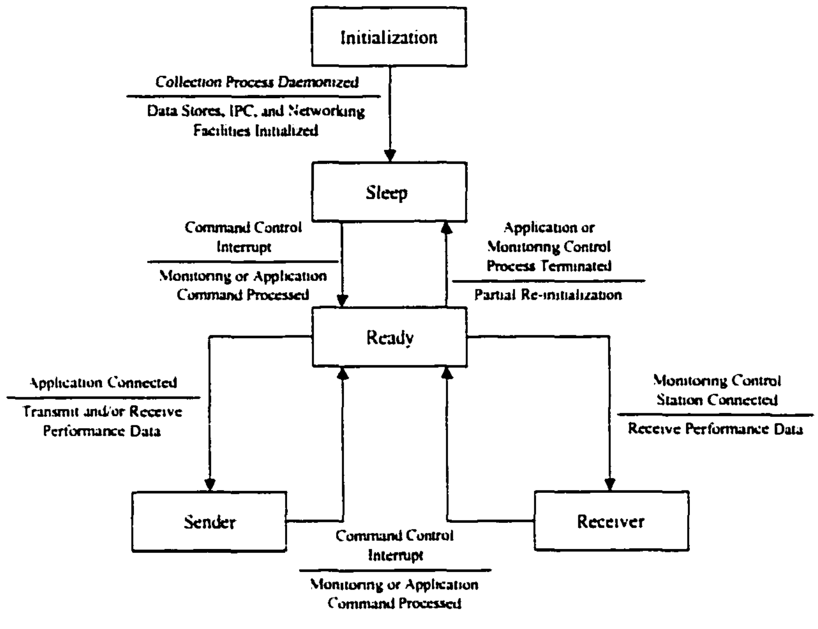

Initialization
Collection Process Daemonized
Data Stores, IPC, and Networking Facilities Initialized
Sleep
Command Control Interrupt
Monitoring or Application Command Processed
Application or Monitoring Control Process Terminated
Partial Re-initialization
Ready
Application Connected
Transmit and/or Receive Performance Data
Monitoring Control Station Connected
Receive Performance Data
Sender
Receiver
Command Control Interrupt
Monitoring or Application Command Processed

Fig. 8. State transition diagram of the PerfMETRICS collection daemon.

The PerfMETRICS protocol consists of event-based transmission of monitoring control packets and periodic transmission of performance data packets. The two packet types are transmitted via service ports associated with different multicast groups. Using multiple multicast addresses provides the flexibility to create a hierarchy of reporting groups, allowing the monitoring system to scale more easily. The monitoring control packet is used to disseminate information about the monitoring process such as applications starting, stopping, or abnormally terminating, Control packets are also used to suspend or resume the transmission of performance data and to change the interval in which the data is collected and transmitted. Application control can be implemented as well, using the PerfMETRICS control packet. An example is the HLA⇔DIS Interface application discussed in the use cases presented as part of this thesis. PerfMETRICS is able to control the startup and shutdown of the interface application as well as control the flow of specific DIS PDUs and HLA data packets. Additionally, another extension to the monitoring control packets allows application specific data to be transmitted and received by the daemon and used by an HLA federate to make dynamic load scheduling decisions. Details on this application are also found in the use cases.

The monitoring control packet acts as the header for the PerfMETRICS performance data packet; the performance data packet, therefore, consists of a control packet and a copy of the shared memory buffer (described early in this section) containing the performance data. One drawback to the current implementation of this protocol is that it does not allow for only application specific data to be passed, meaning that, depending on the number and types of applications using PerfMETRICS, all data packets have the same structure, regardless of the type of monitored application. The current implementation should be modified to allow for variable length data packets whose contents would only contain data relevant for the application type sending the packet. The collection daemon uses the External Data Representation, or XDR, to perform data marshalling. The decision to use XDR was a tradeoff between the flexibility of XDR for porting applications across different platforms, and the constraint of requiring other applications to use XDR to translate the data from a PerfMETRICS data packet into a usable format. A detailed description of the PerfMETRICS monitoring control packet and the performance data packet is found in the appendix.

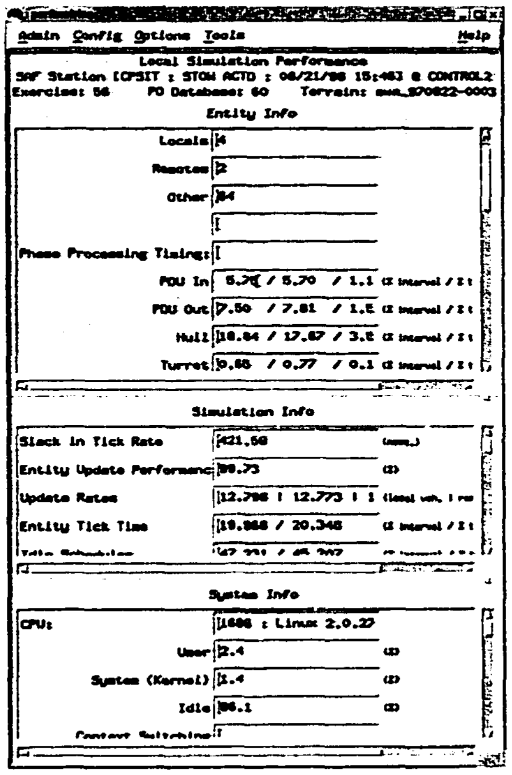
An important component of the PerfMETRICS collection daemon is the logging capability. The daemon process logs information related to the entire run-time monitoring environment. It logs important initialization information as well as monitoring control information. Figure 4.4 shows a sample log file and provides a time series example of the collection daemon processing as discussed in this section. This particular listing indicates a successful initialization of the daemon process; the signal disposition is set, IPCs and Multicast/IP communications is initialized, access to the kernel statistics is initialized, and then the daemon enters the Sleep state. When the application (in this case, a Human Computer Interface, or HCI, on a workstation called pseudo_1) attaches to the shared memory segment and is ready to send and receive performance data, it sends a signal waking the daemon from its Sleep state. The daemon enters the Sender state (referenced

```
Tue Jun  8 15:18:27 1999 Hostname: pseudo_1 (192.5.11.54)
Tue Jun  8 15:18:27 1999 Starting perfcollectd (pid=28479) daemon.
Tue Jun  8 15:18:27 1999 Signal disposition set.
Tue Jun  8 15:18:27 1999 Removed existing shared memory segment # 1
Tue Jun  8 15:18:27 1999 Removed existing shared memory segment # 2
Tue Jun  8 15:18:27 1999 Removed existing semaphore set # 1
Tue Jun  8 15:18:27 1999 Removed existing semaphore set # 2
Tue Jun  8 15:18:27 1999 IPC initialization complete.
Tue Jun  8 15:18:27 1999 Data Group: FD=4 PORT=6000 GROUP=224.0.1.255
Tue Jun  8 15:18:27 1999 Data Group: FD=5 PORT=6000 GROUP=224.0.1.255
Tue Jun  8 15:18:27 1999 Control Group: FD=6 PORT=6002 GROUP=224.0.2.255
Tue Jun  8 15:18:27 1999 Control Group: FD=7 PORT=6002 GROUP=224.0.2.255
Tue Jun  8 15:18:27 1999 Comm. initialization complete.
Tue Jun  8 15:18:27 1999 System data initialization complete.
Tue Jun  8 15:18:27 1999 Going to SLEEP.
Tue Jun  8 15:20:10 1999 Entering SIM STATE
Tue Jun  8 15:20:10 1999 pseudo_1:c1:HCISTART:Tue Jun  8 15:20:10 1999
Tue Jun  8 15:20:10 1999 HCI Connected.
Tue Jun  8 15:21:31 1999 pseudo_4:c1:SIMSTART:Tue Jun  8 15:21:22 1999
Tue Jun  8 15:21:41 1999 pseudo_7:c1:SIMSTART:Tue Jun  8 15:21:37 1999
Tue Jun  8 15:24:11 1999 pseudo_4:c2:SIMCRASH:Tue Jun  8 15:24:02 1999
Tue Jun  8 15:24:18 1999 pseudo_1:c2:HCISTOP:Tue Jun  8 15:24:18 1999
Tue Jun  8 15:24:18 1999 Going to SLEEP.
```

Fig. 9. PerfMETRICS collection daemon log file.

in the figure as SIM STATE) and is able to send and receive performance data. As seen in the figure, the daemon logs monitoring control information. The figure shows two other workstations (pseudo_4 and pseudo_7) each starting up a simulation. Pseudo_4's simulation terminates abnormally (SIMCRASH) and then the HCI process on pseudo_1 terminates normally, transitioning the daemon back into the Sleep state. This kind of information is valuable for assessing not only the collection daemon's processing but also provides a way of analyzing the entire performance monitoring environment.

4.2.2 Data Presentation (GUI)



```
Local Simulation Performance
SAF Station [CPSIT : STOW ACTD : 08/21/98 15:46] @ CONTROL2
Exercise: 56     PO Database: 60     Terrain: [illegible]_970822-0003
Entity Info
Locals 4
Remotes 2
Other 84
Phase Processing Timing:
FOU In 5.70 / 5.70 / 1.1 (% interval / % t
FOU Out 7.50 / 7.81 / 1.E (% interval / % t
Hull 18.84 / 17.87 / 3.E (% interval / % t
Turret 0.88 / 0.77 / 0.1 (% interval / % t
Simulation Info
Slack in Tick Rate 421.50 (msec.)
Entity Update Performanc 98.73 (%)
Update Rates 12.798 | 12.773 | 1 (local veh. | rem
Entity Tick Time 19.868 / 20.348 (% interval / % t
System Info
CPU: i686 : Linux 2.0.27
User 2.4 (%)
System (Kernel) 1.4 (%)
Idle 96.1 (%)
Context Switching
```

Fig. 10. PerfMETRICS GUI

PerfMETRICS uses a Graphical User Interface (GUI) to display data collected at run-time. The interface provides a tabular display of the performance information collected from each application reporting data during an exercise. Figure 4.5 provides an illustration of the main PerfMETRICS window. The data presentation is partitioned into three sections displaying the entity metrics, simulation metrics, and system metrics discussed in Section 3. Data unit conversions are handled within source code. The appearance of the data labels and notes on the display is controlled via the X11 resource files. The display is initialized to show performance information for the application associated with the first data packet received by the collection daemon. Performance information for different applications (different workstations on the network) can be selected by pressing the right mouse button anywhere in the GUI window and selecting the desired workstation from the pull-down menu. It is possible to view multiple workstations at the same time by selecting the "New View" option from the main "Admin" menu. The windows can be tiled on the monitor screen allowing performance comparisons of multiple machines at run-time. Logging of the performance information to disk for post-exercise analysis is also controlled via the PerfMETRICS GUI.

The window shown in Figure 4.5 is the default window for showing performance information. The user interface however, is designed to show alternative views depending on the kind of application being monitored. An example is the view developed to specifically show performance data for an HLA⇔DIS interface discussed in detail in Section 5. When a user selects a machine that is reporting performance data from a different kind of application (e.g., an HLA Interface), the view will automatically change, triggered by an "application indicator flag" in the data packets associated with that machine. Using the "Config" menu option it is also possible to readily integrate customized windows to provide, among other things, exercise or application control functions. This capability was utilized to implement a control function for the HLA⇔DIS interface; turning on and off the flow of different DIS and HLA data packets through the interface.

Similar to simulations instrumented for PerfMETRICS, the GUI process communicates with the PerfMETRICS collection daemon through a shared memory interface (refer to Figure 4.1). Constraints are programmed into the collection daemon to prevent the GUI from connecting to the collection daemon if a simulation is already connected. A signal (software interrupt) is sent to the GUI indicating the daemon process is being used by another process. The GUI traps the signal and then terminates with an error message. The collection daemon logs the condition of the GUI failing to connect with the daemon process. Depending on the hardware / software configuration and analysis objectives this constraint may or may not be desirable and is readily removed, although the collection daemon has not been tested for concurrent reading and writing of performance data.

An important point to emphasize is the design and development focus for PerfMETRICS was not on the GUI. The intent was to concentrate efforts on other aspects of the monitoring system such as data semantics and communications infrastructure. The GUI provides basic capabilities for a user to monitor, control, and log data.

4.2.3 Data Analysis

Data analysis associated with DIS and HLA exercises can be considered from two aspects: real-time and post-exercise analysis. The graphical interface used to present real-time data was discussed in the previous section. Its practical application to real-time (or near, real-time) data analysis for DIS and HLA simulations is more closely related to exercise monitoring and control, the objective being to support timely decisions regarding the conduct of a given simulation study or exercise. The GUI provides a means to observe system-level, application-level, and model-level performance characteristics of individual workstations (i.e., simulation engine or other DIS/HLA application). The collection daemon can be configured to control the rate that performance information is collected and determines the timeliness and accuracy of the data on which decisions are based. It is possible to make comparative assessments regarding the performance of different or similar applications when multiple windows are tiled on the monitor screen.

For post-exercise analysis, the current PerfMETRICS implementation logs all performance data to a file. The file format is simply a binary dump of the PerfMETRICS protocol data packets. This provides a simple and fast method for saving the data with the least impact on run-time performance of the collection daemon. A separate process is used to extract the desired data, perform data or unit conversions (if required) and redirect the processed data to another file in an ASCII format. Figure 4.6 shows a segment of the configuration file used to specify exactly what data is extracted (*data mining*) from the binary data file and for which workstation (application). Figure 4.7 shows a segment of an ASCII output file generated by the data selection process. The resultant performance information can be imported into a suitable analysis application (e.g., Minitab, Microsoft Excel, SAS).

PerfMETRICS provides the capability to observe the workloads from each major component of the simulation architecture. Time-based analysis of performance data from individual workstations provides detailed information on simulation (application) performance over a narrow aspect of the overall distributed simulation system. The same process used to extract the desired performance data also aggregates utilization data for all workstations reporting PerfMETRICS data (*data fusing*). This operation allows global simulation metrics to be generated and helps decision-makers to understand how well

```
/** convert.config - configuration files to specify which PerfMETRICS
             output variables to print **/
hostid  3
input     /usr1/_082598083113.air15
output  gw3_1.txt
starttime  00:00:00
stoptime   24:00:00
month          0        /* month */
day            1        /* day of month */
time           1        /* time of day */
host           1        /* simulation hostname */
exid           0        /* exercise id */
poid           0        /* po database id */
collint        0        /* data collection interval */
seqnum         1        /* data packet sequence number */
lveh           0        /* # of local vehicles */
lmis           0        /* # of local missiles */
lstr           0        /* # of local structures */
lstl           0        /* # of local stealths */
lenv           0        /* # of local environmentals */
```

Fig. 11. PerfMETRICS data selection configuration file.

available resources (in this case, workstations) are being utilized. The next section of this thesis provides greater detail regarding data analysis and includes examples of how data collected using the PerfMETRICS monitoring system can be used to analyze and assess the run-time performance of different distributed simulation environments.

4.3 Instrumentation Costs

If performance monitoring is required for other than the development and testing phases of the DIS/HLA life cycle (i.e., for experimentation or production use), a software monitoring architecture could be too intrusive on system performance. A compromise must be made on run-time measurements necessary to obtain performance data, the objective being to establish a balance between the adequacy of measured data (for the purpose of analysis) and the intrusiveness of the monitoring system (and its perturbation of the performance analysis). Prerequisites for determining the appropriate compromise (tradeoffs) include assessing the value of the performance data as it relates to the analysis activity, developing an intuition regarding the potential impact of monitoring on model or

```
day time host seqnum lairveh rairveh sns_int tsk_int cp_slack cp_ratio
ltk_rate rtk_rate tktm_int wttm_int rtitm_int
  8  18:06:34  PSEUDO_3    138     23     50 12.97 5.12   396.90 100.00
5.87   6.18   59.42  0.12   5.11
  8  18:06:39  PSEUDO_3    139     23     50 12.97 5.12   396.90 100.00
5.87   6.18   59.42  0.12   5.11
  8  18:06:44  PSEUDO_3    140     25     50 18.19 6.14   396.90 100.00
7.28   7.36   74.95  0.00   4.93
  8  18:06:49  PSEUDO_3    141     25     50 18.19 6.14   396.90 100.00
7.28   7.36   74.95  0.00   4.93
  8  18:06:54  PSEUDO_3    142     25     50 17.93 5.98   396.90 100.00
8.02   7.99   77.75  0.00   4.63
  8  18:06:59  PSEUDO_3    143     25     50 17.93 5.98   396.90 100.00
8.02   7.99   77.75  0.00   4.63
  8  18:07:04  PSEUDO_3    144     25     50 18.19 6.11   361.06 100.00
8.02   8.08   76.63  0.00   4.89
  8  18:07:09  PSEUDO_3    145     25     50 18.19 6.11   361.06 100.00
8.02   8.08   76.63  0.00   4.89
  8  18:07:14  PSEUDO_3    146     25     50 17.68 5.89   361.06 97.68
8.23   8.17   76.70  0.00   4.53
  8  18:07:19  PSEUDO_3    147     25     50 17.68 5.89   361.06 97.68
8.23   8.17   76.70  0.00   4.53
```

Fig. 12. PerfMETRICS data selection output file.

simulation behavior, and understanding the monitoring effects (perturbation) on the analysis results. Perturbations in performance analysis may be caused by the delays induced from executing performance measurement code (instrumentation); this also conditionally affects other low-level resources such as memory caches, pipelines, and register allocation. Other perturbations that occur when monitoring distributed simulations are created when the execution of instrumentation code results in delays causing a re-sequencing of simulation events; event reordering can potentially lead to incorrect execution of the simulation.

Analysis of the run-time performance of a distributed simulation must, at some level, account for the perturbations due to performance measurements. An estimate of the costs of performance measurements must be determined. These costs can be accounted for during the presentation of performance information.

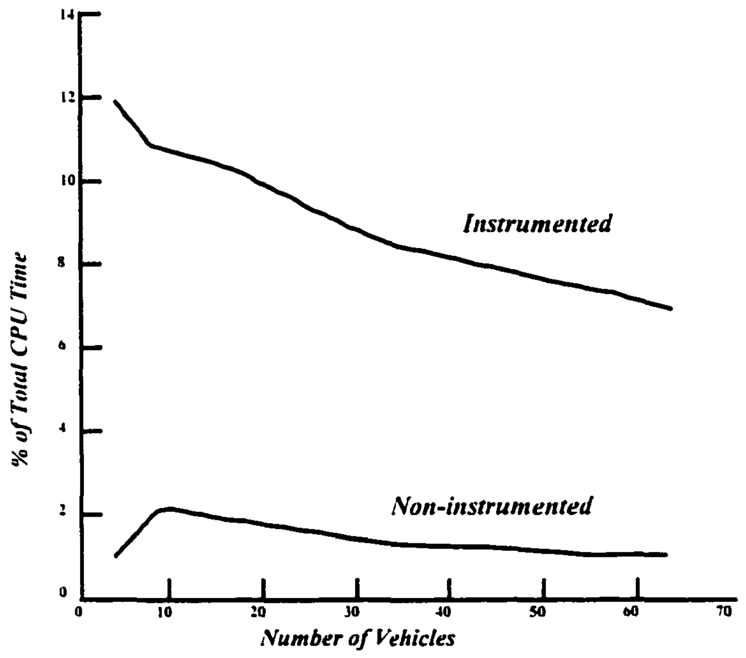


Fig.13. Instrumentation costs for SAF "high-frequency" function.

The above discussion motivates the need to understand monitoring and measurement overheads so choices can be made regarding monitoring system design and implementation. Early on in the PerfMETRICS development process, studies were conducted to understand the costs associated with the instrumentation component of the PerfMETRICS architecture; the most intrusive component since it directly affects the simulation's run-time execution path. Figure 4.8 presents results from a timing analysis done on a SAF application instrumented to use PerfMETRICS. The results show the percentage of the total CPU time used by a "high-frequency", run-time function, specifically the SAF sub-scheduler responsible for invoking the execution of each entity's sub-models. The data shows the worst-case overhead induced by the instrumentation is less than twelve percent. This instrumentation cost was deemed reasonable considering the value of the sub-model performance data gathered in this function. The relative instrumentation costs in this function decrease as the number of entities increase. This is an artifact of the larger entity count creating a greater simulation

TABLE 4
Sample Instrumentation Costs for a SAF Run-time Function

Instrumentation Code	**System Calls**	
	_times	**_semop**
36.24 +/- 1.3 %	**41.74 +/- 1.0 %**	**22.08 +/- 0.08 %**

workload in other functional areas of the processing, resulting in fewer calls to the sub-scheduler function. This example illustrates the dependency between instrumentation costs (intrusiveness) and data frequency. Other dependencies exist in terms of the granularity of the data that is captured and the volume of performance data that is monitored and/or collected.

Another significant factor affecting run-time costs of monitoring and data collection is the design and implementation of the instrumentation code itself. Alternative designs and implementations should be considered in conjunction with the data design issues discussed above. Consider the impact of making frequent calls to the operating system to get the value of the system clock or some other function. Table 4.2 presents more data from the timing analysis discussed in association with Figure 4.8. The table shows the relative costs associated with the instrumentation code and the system calls it invokes. The expense of making frequent calls to the operating system (in this case time-of-day and requests to acquire a semaphore) is obvious. An alternative implementation to using kernel semaphores is to implement "user" semaphores by using a shared variable.[9] This was implemented in a Linux-based version of PerfMETRICS using a machine-level instruction intrinsic to Intel-based processors, specifically an *exchange word* (xchgw) instruction. The required semaphore lock and unlock functionality was implemented using in-line assembler and the xchgw instruction. The code segment was as follows:

```
#define LockInit(p)       (p=0)
#define UnLock(p)         (exchange(&p,0))
#define Lock(p)           while(exchange(&p,1)) while (p)
#define lock_t            volatile int
__inline__ int exchange(volatile * addr, int reg)
{
  int oldbit;
  __asm__ __volatile__( LOCK_PREFIX
                    "xchgw %0,%1"
                    :"=q" (oldbit), "=m" (*(addr))
                    :"m" (*(addr)), "0" (reg));
  return oldbit;
}
```

The resulting implementation eliminates the overhead associated with the _semop system call and reduces the costs of the instrumentation code (user-defined) by almost fifty percent. These results illustrate the significant effect of certain implementation decisions on the costs associated with performance monitoring. In some instances, executing instrumentation code for inordinately large amounts of time is tolerable. An example is the time spent executing the instrumentation code for measuring scheduler idle time. Since the simulation is obviously not busy (i.e., its idle waiting for an event to occur), there are no adverse effects on executing the instrumentation code because it is using compute cycles that are otherwise unused for real simulation events.

To summarize, software-based performance monitoring systems such as PerfMETRICS mandate a careful balance between the volume and accuracy of the data. Excessive amounts of instrumentation and a poor implementation will perturb the monitoring process and analysis results; not enough instrumentation limits the accuracy of the data used to characterize system behavior. Lacking perturbation models to quantify the effects of instrumentation costs relative to data requirements, intuition and trial-and-error must be used to develop a monitoring system that adequately meets performance analysis objectives without being too intrusive on system performance.

---

[9] This is possible provided there are guarantees that writing to the shared variable (the semaphore) is an atomic operation (no-preemption) from the operating system kernel perspective. Note that this mechanism is generally non-portable since it relies on the explicit use of a processor-specific instruction set.

SECTION 5

PERFORMANCE MONITORING USE CASES

This section presents use cases containing performance monitoring and analysis results from real-world programs. The examples illustrate the utility of performance monitoring during the design, development, and use of DIS and HLA-based applications in different M&S environments including military staff- and unit-level training, technology research and development, and system acquisition and procurement. The use cases are imbued with the framework presented in this thesis, and demonstrate its practical application to support model and simulation design, as well as the configuration and run-time control of different applications in DIS and HLA simulation environments.

The common theme among all the use cases is the general applicability of PerfMETRICS to collect data useful for decision-makers. Each use case involved the monitoring and data collection from a STOW application as discussed in the thesis approach (Section 1.4). The first case made use of the framework during the complete simulation life-cycle of the DARPA STOW ACTD. This involved using PerfMETRICS to monitor and collect performance information useful for *model design and testing*, *scenario configuration*, and overall *monitoring of the resources* used in the air component of the synthetic environment during the ACTD. The second case discusses how PerfMETRICS is used to provide *dynamic load-scheduling* of the entity workload in the U.S. Navy's Battle Force Tactical Trainer (BFTT) Air Management Node (AMN). The next use case presents results from performance studies done to make *technology assessments* about different modeling techniques for use in the U.S. Army's Aviation Combined Arms Tactical Trainer -Aviation (AVCATT-A) program, the underlying objective being to provide *capacity planning* data for system procurement. The final case study presents an application of the framework during an experimental *technology insertion* program conducted under the aegis of the U.S. Air Force's Distributed Mission Training (DMT) program. The goal was to *understand the performance impact* of integrating DIS-based virtual cockpit simulators using HLA-based STOW technology, the principal performance objective being to minimize latency and maximize throughput of the M&S data used by the image generators in the simulator visual displays. During this experiment, PerfMETRICS was also used to *support application design and development* and provide *run-time control* of the integration mechanism, an HLA⇔DIS Interface application.

5.1 Synthetic Theater of War (STOW) and DARPA's Advanced Concept Technology Demonstration (ACTD)

As mentioned, the ACTD was a technology demonstration. The primary focus was the development of new models that enhance the realism of the synthetic environment. Over 270 models and software libraries were designed and implemented by different model developers. In this kind of simulation development environment, different understandings of model requirements invariably led to different levels of resolution among interacting models. Additionally, model implementations frequently led to inordinately high processing costs that were intolerable in terms of execution costs. The timing requirements of the DIS/HLA real-time applications only exacerbated the problem of controlling these costs. Performance monitoring during the ACTD simulation development and pre-exercise testing provided feedback to model developers and assessments were made about the performance impact of certain modeling design and implementation decisions.

The principal simulations used during the STOW ACTD were SAF variants of Modular Semi-Automated Forces (ModSAF), a simulation system designed to meet the DoD's distributed simulation training requirements. The SAFs were originally implemented using the DIS protocol but to satisfy DoD requirements the architecture, design, and implementation evolved into an HLA-based simulation. PerfMETRICS was used to support technical and operational tasks associated with the Air Synthetic Forces (AirSF) component of the ACTD. AirSF simulates Fixed-Wing Aircraft (FWA), Rotary-Wing Aircraft (RWA), and the munitions employed by these aircraft. The FWA and RWA were primarily simulated using the TacAir-Soar (hereafter, also referred to as just "Soar") technology described in Section 1.2.2.2 and the simulated actions of each entity type are based on its real-world capabilities and military doctrine.

5.1.1 Model Design and Testing

The entity performance data monitored from AirSF consists of count data for local and remote entity types and the time spent ticking the sub-model components used to model each entity type (i.e., FWA, RWA). The AirSF architecture is implemented so that every entity in the system is ticked at a periodic interval. During each entity tick, a sub-scheduler is called that executes each sub-model used to construct the overall representation of the aircraft. A radar sensor model is one example of a sub-model used by an aircraft. Most simulated aircraft representations include common sensor, munitions, and flight dynamics sub-models. Therefore, the only significant difference among aircraft representations is the behavioral models. With this realization, PerfMETRICS was used to collect relative execution times for entity sub-models including hull modeling (e.g. FWA flight dynamics), weapons modeling, sensor models (e.g., radar and visual), and vehicle tasking and behaviors (e.g., Close Air Support, or CAS, and Combat Air Patrol, or CAP). Critical simulation metrics were derived by collecting run-time data from the appropriate libraries used in the AirSF implementation. Data collected consisted of, among other things, timing data for entity state update rates (i.e., tick rate), idle scheduler time, and time spent in the RTI. The RTI timing data proved to be of special interest because people were looking for critical feedback on the costs/benefits of migrating to HLA-based simulation environments. The need to characterize performance of the RTI component has resulted in previous performance studies that made assessments about throughput and latency characteristics [56]. These studies provided feedback for RTI developers by characterizing the performance of algorithms, protocols, and software implementation. Meaningful information for application developers, however, requires a characterization that is indicative of the RTI performance impact on DIS/HLA federate modeling and development. Figures 5.1 – 5.2 illustrate the wide variance in RTI processing requirements among different Fixed-Wing Aircraft (FWA) missions simulated in AirSF during the ACTD. The figures provide moving averages of data taken from three scenarios executed during the first three hours of the exercise. Each scenario was executed on a different simulation engine.

Figures 5.1 and 5.2 reveal very similar workloads among the scenarios in terms of the amount of network traffic and the volume of remotely simulated vehicles. Figure 5.3, however, shows a distinct difference in the number of remote radio entities that the Electronic Support Mission (ESM) receives. The impact is shown in Figure 5.4. Due to the comparatively large number of remote radios the RTI requires a significantly greater proportion of simulation processing time for ESM than for CAS or CAP missions. The correlation values for the three mission scenarios are shown in Table 5.1; note the much stronger correlation of RTI tick rate with remote radios than with remote vehicles. The data reinforces our conclusion that the number of radios to which the ESM platforms subscribe significantly affects performance of the simulation in terms of the time required to manage simulation data. This information is useful to DIS/HLA developers when considering a different design of the radio spaces (i.e., repartitioning of the subscription space). As an alternative, ESM modelers could modify the requirement for certain radio frequencies. Regardless, this data provide insight into the impact of the RTI implementation and its effects on modeling FWA and certain mission scenarios.

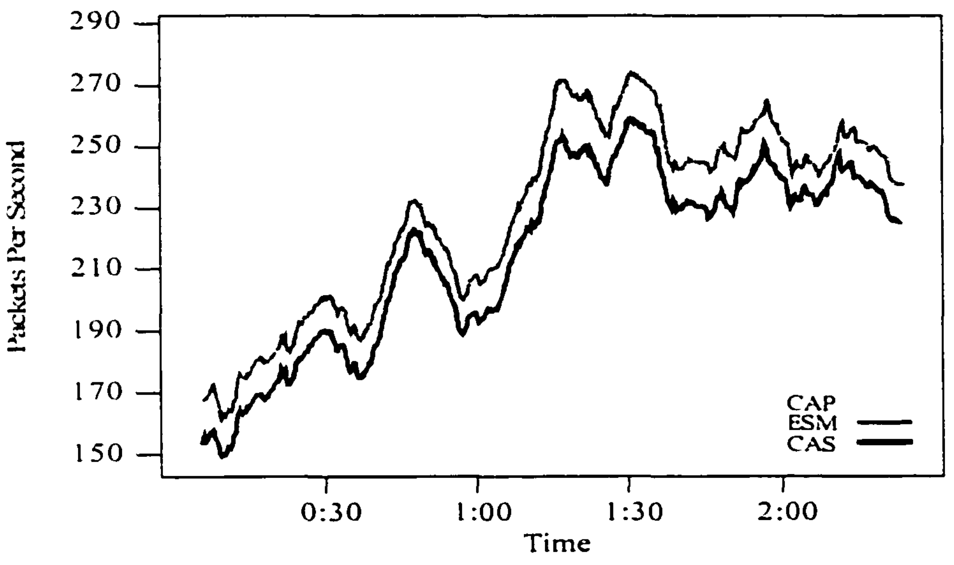


Fig. 14. STOW ACTD AirSF IP traffic workload by mission type.

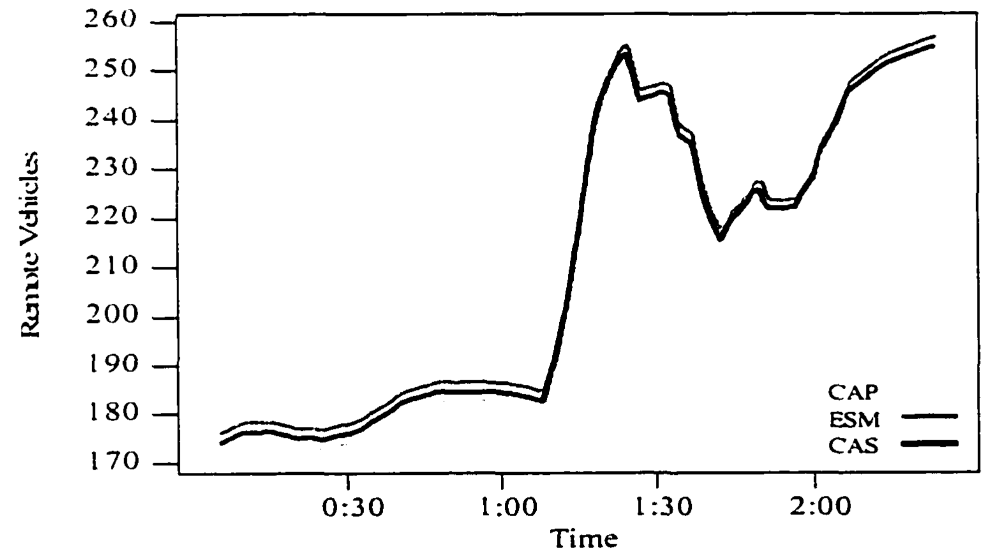


Fig. 15. STOW ACTD AirSF remote vehicle count workload by mission type.

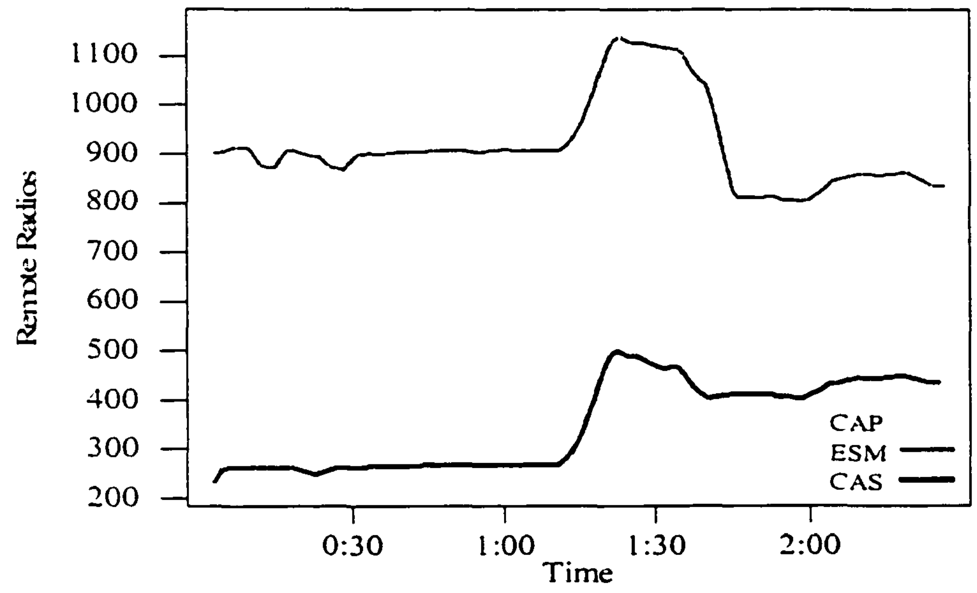


Fig. 16. STOW ACTD AirSF IP remote radio workload by mission type.

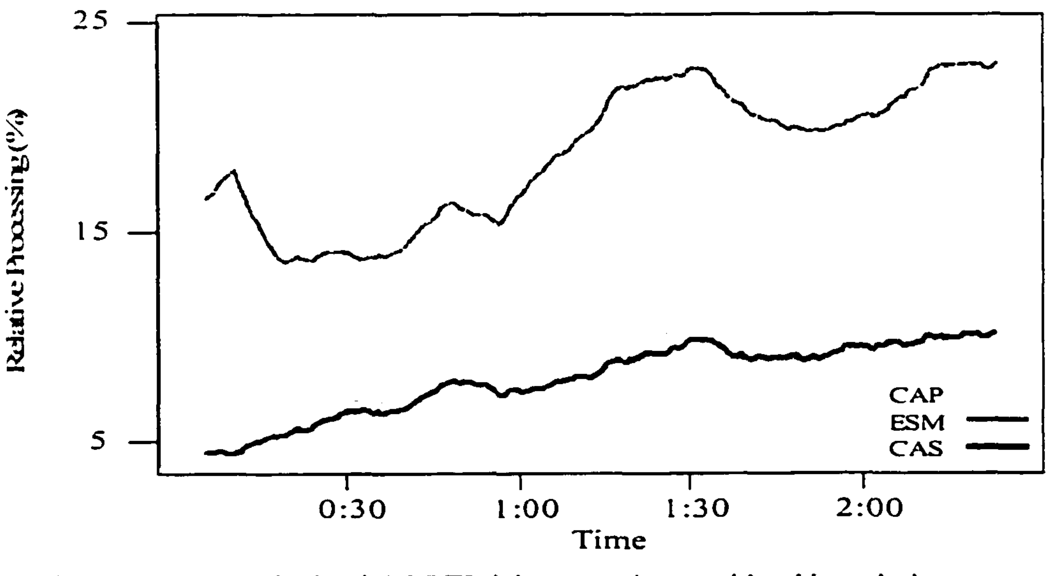


Fig. 17. STOW ACTD AirSF RTI tick processing workload by mission type.

TABLE 5
STOW ACTD AirSF Correlation Of Performance Factors

	Remote Vehicles	Remote Radios	RTI Tick
Remote Radios	0.266		
RTI Tick	0.403	0.887	
IP Packets	0.434	0.118	0.303

As another example, AirSF showed drastic and unexpected performance degradation late in the testing process (prior to the STOW ACTD). Initially, the only reliable correlation the development team was able to make with this degradation was high entity counts on the network. When the PerfMETRICS system was used to monitor performance, developers were able to observe the impact of the high entity counts on specific simulation models. The real-time feedback PerfMETRICS provided allowed the developers to test various mission and aircraft types and to quickly isolate the specific model(s) causing the problem; in this case the sensor model. Correcting the deficiency was a non-trivial problem involving several iterations of code changes. During each iteration the capability of PerfMETRICS to provide detailed information in real time allowed the developers to assess the impact of the code changes made to the model designs and simulation infrastructure.

5.1.2 Scenario Configuration

An important objective for using ADS technologies like DIS and HLA to support military training is to minimize the number of persons required to support a simulation exercise or study. The SAFs used during the ACTD did not require user intervention to control the low-level behaviors of the physical and logical models. Operators were required to make the high-level decisions regarding the design of mission scenarios and entity behaviors. SAF operators can override entity behaviors if required to do so.

DIS/HLA exercises frequently incorporate dynamic free-play in terms of the training audience's interaction with the simulation. However, the exercises are most commonly structured around the execution of pre-scripted events that reflect training objectives and represent scenarios for real-world military operations. Thus workloads are in part predetermined; this provides an opportunity for configuration planners and simulation operators to more effectively utilize available hardware and software resources. As mentioned, accurate assessments about the number of people required to support the simulation exercise are required. Performance monitoring can be used to provide guidance during these activities.

In the past, developing pre-scripted events typically relied on estimates of an initial number of entities (vehicles) that could be simulated on a workstation. In the case of ModSAF, these estimates were made using a benchmark based on simulating tanks. However, different vehicle types have significantly different processing requirements depending on the activities (missions) in which they engage. Specifically, the simulation of fast moving aircraft over a large area of the synthetic battlespace has very different performance characteristics than that for slower moving tanks on a more restricted area of the battlespace.

Additionally, the dynamic behaviors of vehicles during the simulation can result in large fluctuations in workload, depending on the level of interaction with other simulation entities. By monitoring DIS/HLA simulations, metrics can be established that allow the workload on available workstations to be managed based not only on vehicle type but also on the type of activities they are assigned and the entity interactions expected with each scenario. This information is then used to make accurate assessments about the total number of machines and persons needed to support the simulation of specific scenarios during a DIS/HLA exercise. For the STOW ACTD, PerfMETRICS provided the simulation site test director with valuable performance information useful for making decisions about AirSF scenario design and workload configuration. This process involved incremental test phases, the goal being to achieve the best possible utilization and performance of the available resources.

The first phase included tests with only one or two workstations simulating a "best guess" number of entities (based on model developers' recommendations). During this phase expected entity (FWA) behaviors and interactions were verified and the general performance of the simulation was observed. For the AirSF component of the STOW ACTD, performance measures gained during this first phase were especially useful since the simulation of FWA was implemented using TacAir-Soar. The existing TacAir-Soar implementation can have severe processing requirements depending on the number of agents and number of interactions with remote entities. Gaining preliminary performance estimates was critical for exercise planning. Preliminary studies done by Soar developers resulted in a goal to update an agent's entity state at a 3 to 4 Hz. rate. Achieving this goal allows the agents to simulate what is considered "good cognitive behavior." Using PerfMETRICS to monitor the entity state update rates during the initial test phase resulted in better estimates of the maximum number of Soar agents per machine. The numbers, dependent upon the type of mission the agent performs, were as follows: CAS – 2 agents, CAP – 4 agents, ESM – 2 agents.[10]

Initially, it made sense to group different missions together when Soar agents exhibited extensive interaction. An example is Defensive Counter Air (DCA) missions and Airborne Early Warning (AEW) missions. Using PerfMETRICS to monitor overall simulation performance revealed that combining these two missions on the same simulation engine resulted in excessive paging activity (poor working set size characteristics) and consequently very poor performance. This resulted in early repartitioning of the agent workload based on missions. Phase one tests provided a good baseline for assessing simulation and Soar agent performance.

[10] Critical workstation parameters were: 200 MHz Intel Pentium processor; 256 Mbytes RAM.

Subsequent test phases increased the number of simulation entities and agent interactions. In the case of DCA testing (CAP is a specific kind of DCA), the number of air-to-air interactions was increased until the Soar agents' behaviors were determined to be invalid. Subject Matter Experts, or SMEs, made this determination based on visual perception of agent behavior and the update rates provided by PerfMETRICS. This established limits for the number of simulation entities, Soar agents, and their interactions that could be expected to not adversely affect the perception of valid simulation behavior. The final test phase involved the execution of different scenarios in a fully populated battlespace on the STOW WAN. These tests exposed the individual HLA federates (AirSF simulations) to the workload demands expected during the ACTD. Specifically, as many as 5,000 simulation entities and network traffic at the LAN interface averaging between 300 and 800 packets per second.

Using PerfMETRICS during these test phases helped determine the maximal number of Soar agents per workstation that still allowed the achievement of realistic behaviors during the ACTD. This allowed the AirSF test director to make decisions regarding scenario workload and configuration. Using the performance information described above and estimates about the expected number of FWA sorties (missions) during the STOW ACTD exercise, the test director was also able to provide an estimate of the number of AirSF simulation engines (Intel-based PCs) required to meet the exercise objectives. As mentioned, DIS/HLA simulation training exercises are to some extent pre-scripted with well-defined scenarios and objectives. For the AirSF component, SMEs estimated the Air Tasking Order (ATO) that lists missions to be executed, would require a maximum number (at any one point in time) of the following basic mission types: DCA (includes CAP) – 3 missions (12 agents), Strike (includes CAS) - 10 missions (40 agents), and Intel (includes ESM) - 5 missions (10 agents). The number of required engines was then estimated as $N = \sum_{i=1}^{m} \lceil (E_m * S_m) / P_m \rceil$ where N is the number of simulation engines, m is a specific mission type, E_m is the number of entities for mission type m, S_m is the number of scenarios of mission type m, and P_m is the number agents per engine for mission type m (derived using PerfMETRICS).

5.1.3 Resource Monitoring

Based on the process discussed in the previous section, it was estimated the AirSF component for the STOW ACTD would require (at any one period) 26 simulation engines, 3 for DCA, 20 for Strike, and 3 for Intel. The WISSARD facility at NAS Oceana, Virginia Beach, VA (the air simulation component of the STOW ACTD) was issued 36 workstations designated as simulation engines. Figure 5.5 shows the actual utilization of the engines during the STOW ACTD (also derived using data gathered with PerfMETRICS). The median values for backend (simulation engine) utilization over the three days were 72, 75, and 61 percent, respectively. The original estimate of 26 simulation engines is 72% of the 36 issued for AirSF simulations, corresponding closely with the actual usage.

It is important to note that the median values reported above reflect the average time spent executing Soar scenarios. An additional factor can be considered and includes the number of simulation engines that are at any point in time either in a simulation startup mode or in a quiescent mode waiting for the Soar agents to begin their missions. Interestingly enough this factor is representative of overlaps in real world FWA flight operations. When this factor is taken into account the actual backend utilization during the ACTD approaches 100%.

Figure 5.6 provides the average entity state update rates for the principal Soar mission types discussed throughout this paper. It shows that the prescribed update rates were probably met for most of the Soar missions except in cases where more agents were loaded onto a single workstation than were deemed reasonable by the pre-exercise tests.

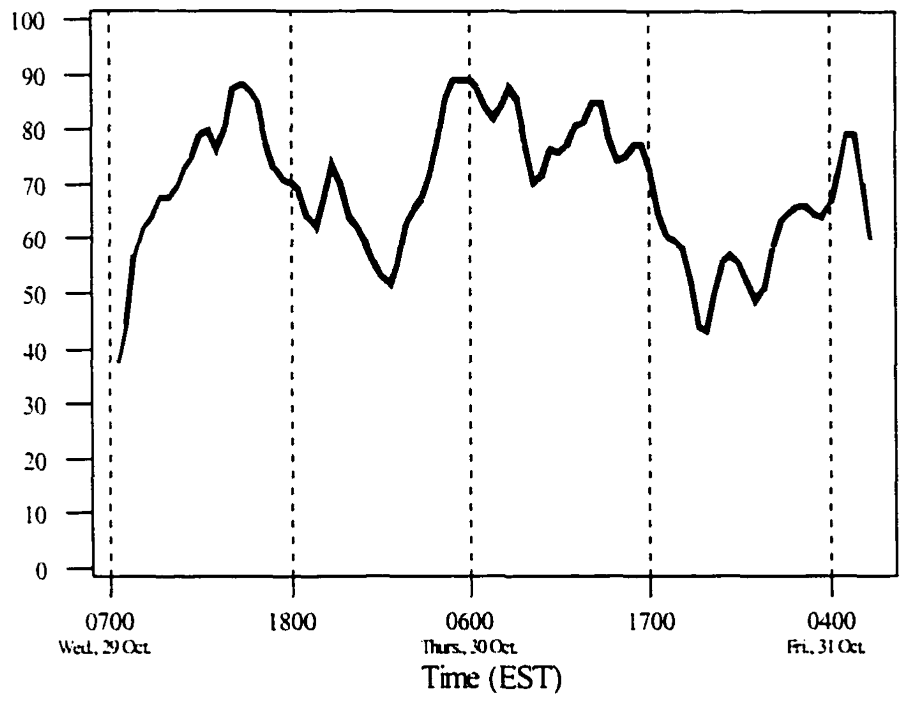


Fig. 18. STOW ACTD AirSF simulation engine utilization.

Using PerfMETRICS provided valuable performance information to model developers and testers. It helped them make reasonable assessments about the capabilities of existing hardware to provide credible entity-on-entity level simulation to augment operational testing. By gradually introducing critical factors in a controlled test environment and using PerfMETRICS to measure their impact on performance, scenario design and configuration planning for the STOW ACTD became a more quantifiable process. The test director and site manager were provided with information needed to make tradeoffs in hardware, software, and personnel requirements.

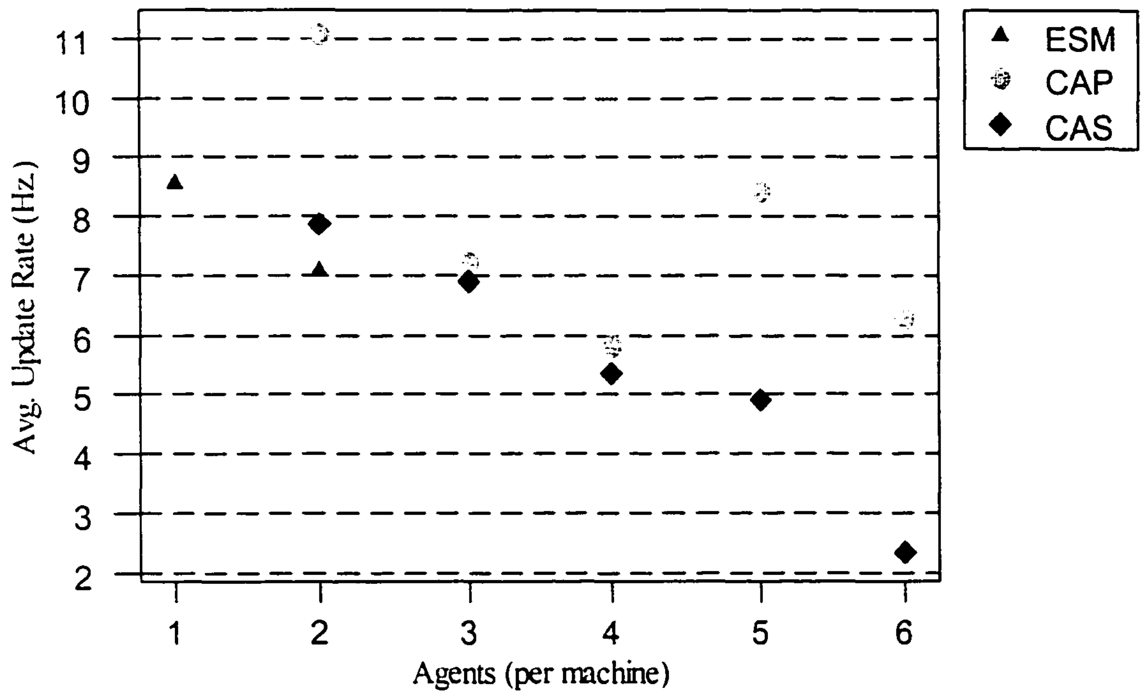


Fig. 19. STOW ACTD AirSF entity state update rates by mission type.

The DARPA STOW ACTD provided an excellent opportunity to use the framework presented in this thesis. This use case was the first time PerfMETRICS was used to support a real-world application. Performance information was successfully collected and used to support decision-makers during model design and testing, and during scenario configuration and planning. It was also used during the simulation exercise to provide overall monitoring of the resources in the air component of the synthetic environment.

5.2 U.S. Navy Battle Force Tactical Trainer (BFTT) Air Management Node (AMN)

The U.S. Navy has developed the Battle Force Tactical Training (BFTT) system to provide shipboard training across the full spectrum of mission scenarios from unit level, small team training (Tier I), to theater level joint training exercises (Tier III). Since peacetime constraints have limited the funds available for adequate mission training using live forces to maintain an adequate level of military readiness, the DoD is exploring the increased use of simulation to augment live training. The principal technical objective of BFTT is to create a network of coordinated training using ADS to stimulate shipboard sensors. The BFTT system provides immediate feedback regarding the performance of trainee(s). The implementation of BFTT consists of a collection of hardware and software components used for training scenario generation, stimulation/simulation control, data collection, and performance monitoring. Shipboard and shore-based BFTT networks may be interconnected to provide a larger, WAN-based synthetic battlespace.

An important BFTT training capability is Air Traffic Control (ATC). The BFTT Air Management Node (AMN) is designed as a training tool to help keep air controllers proficient with ATC terminology and control procedures. Principal goals in developing the AMN are to improve upon the ATC training capabilities of the BFTT Combat Simulation Test System (CSTS), including an improved HCI and enhanced simulation and modeling capabilities (i.e., fidelity). An additional objective is to initiate a migration path for BFTT using the Defense Modeling and Simulation Organization's (DMSO) High Level Architecture (HLA). To achieve these goals, the Naval Sea Systems Command's (NAVSEA) Performance Monitoring, Training, and Assessment Program Office (PMS430) proposed the use of simulation technologies developed during the STOW ACTD. Various technical challenges existed regarding the integration of STOW technology with the BFTT system but doing so would provide a more robust and realistic synthetic environment capable of supporting U.S. Navy training requirements.

The AMN architecture and design (both hardware and software) is influenced by many factors, among them the physical and environmental shipboard constraints found on U.S. Navy combat vessels. Given these constraints, the actual AMN implementation was limited to a total of nine processor boards to service all the application needs of the AMN, including simulation engines, Human Computer Interface (HCI) applications, an HLA⇔DIS interface to integrate the AMN with the existing shipboard training simulation network, and a special interface application for communicating and sharing data with shipboard consoles used to control the simulation training environment. The

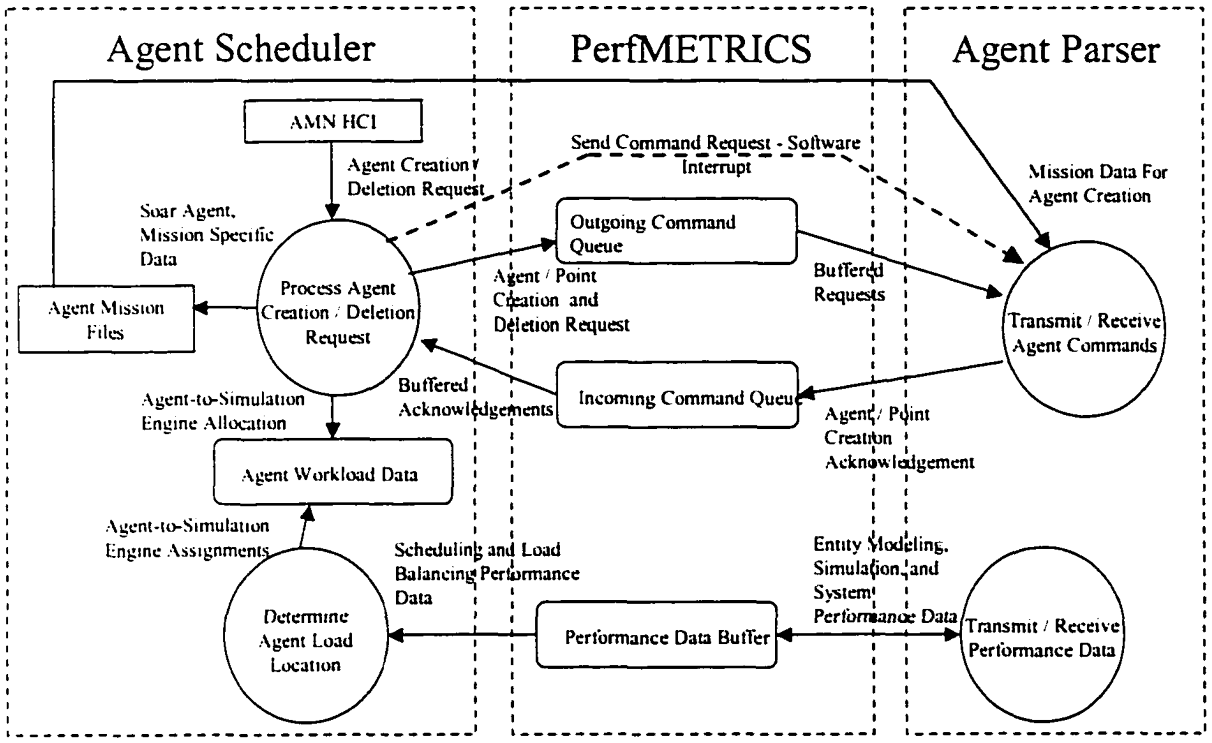

Agent Scheduler
PerfMETRICS
Agent Parser
AMN HCI
Agent Creation / Deletion Request
Soar Agent, Mission Specific Data
Send Command Request - Software Interrupt
Mission Data For Agent Creation
Outgoing Command Queue
Agent Mission Files
Process Agent Creation / Deletion Request
Agent / Point Creation and Deletion Request
Buffered Requests
Transmit / Receive Agent Commands
Buffered Acknowledgements
Incoming Command Queue
Agent-to-Simulation Engine Allocation
Agent / Point Creation Acknowledgement
Agent Workload Data
Agent-to-Simulation Engine Assignments
Scheduling and Load Balancing Performance Data
Entity Modeling, Simulation, and System Performance Data
Determine Agent Load Location
Performance Data Buffer
Transmit / Receive Performance Data

Fig. 20. BFTT AMN dynamic load scheduling using PerfMETRICS.

final design consists of only four processor boards designated as simulation engines and creates performance issues related to hardware/software capabilities meeting processing requirements of a highly dynamic AMN scenario workload (based on TacAir-Soar technology). The run-time architecture of the AMN is complicated by requirements to support dynamic creation, deletion, and control of Tac-Air Soar agents. Operator control of the agents is via the AMN HCI, displayed as a GUI at the BFTT Operator Console (BOPC). What is needed is the ability to make good use of the processing cycles on the available processors (the 4 simulation engines). One technique to achieve this is *static load scheduling* based on the existing processor and memory workload at any given instant in time.

PerfMETRICS was selected to provide the AMN load scheduling algorithms with requisite performance data. Using the existing communication infrastructure provided by

PerfMETRICS to manage the agents simplified the AMN design and reduced the amount of new code development required to implement dynamic agent creation. This was an important consideration given the strict AMN development schedule. Figure 5.7 shows a Data Flow Diagram (DFD) of the AMN dynamic agent control architecture. The PerfMETRICS collection daemon was modified to buffer application requests to create and delete TacAir-Soar agents. Run-time performance data from the application related to entity update rates, number of existing entities, and other low-level operating system and hardware performance is passed into the agent scheduler process residing in the AMN HCI process. Initial work was done to develop different scheduling algorithms. Development and test scheduling constraints, however, resulted in a simple round-robin scheduling algorithm being used in the fielded system. The data infrastructure to support different scheduling algorithms was kept in the source code and can be used to provide performance information for future, more sophisticated scheduling algorithms.

5.3 Aviation Combined Arms Tactical Trainer–Aviation (AVCATT-A)

This case study examines the use of PerfMETRICS to monitor, collect, and analyze performance information to make assessments about system design and procurement for the U.S. Army's Aviation Combined Arms Tactical Trainer-Aviation (AVCATT-A) trainer, a re-configurable manned simulator system. It is a dynamic, alternative instructional concept to train and rehearse using networked simulations and simulators in a collective and combined arms simulated battlefield environment. The systems principal objective is to provide unit-level proficiency training for rotary wing (helicopter) aircraft. The AVCATT-A system is currently under development and will provide a fair fight, realistic, high intensity, task-loaded combat environment composed of attack, reconnaissance, cargo, and utility aircraft platforms; SAF workstations; an After Action Review (AAR) capability; a Battlemaster Control (BMC) console; and workstations for ground maneuver, Fire Support (FS), CAS, logistics, battle command, and engineer role players. The benchmark testing presented in this use case was intended to help decision makers assess the use of different behavioral modeling technologies (i.e., TacAir-Soar, SAF Taskframes), their impact on run-time resource requirements and performance, and their relationship to the expected training scenario workload.

The discussion and data presented thus far clarify the fact that in typical DIS and HLA simulation environments, large entity counts require significant computer and network resources such that the simulation workload and its impact on run-time performance must be carefully considered in system development. Computer and network resources must be determined based on workload requirements of representative exercise scenarios identified by subject matter experts. As military programs such as AVCATT-A continue to mature, users will demand increasingly complex training scenarios; therefore it is important to understand the simulation resource requirements. To support AVCATT-A system design and procurement and to gain insight into the problem of defining resource requirements, benchmark testing was done using semi-automated and fully automated Joint Semi-Automated Forces (JSAF) fixed and rotary-wing aircraft representations.[11]

The goals of the test were to provide estimates of the number of TacAir-Soar agents and SAF taskframe-based entities that can effectively be simulated on a specified computer configuration. Objectives included: 1) creating a synthetic battlespace with representative ground and air forces, 2) creating and executing SAF scenarios to generate realistic workloads on the simulation engines, and 3) measuring the run-time performance of the simulations while executing the SAF scenarios. Initially, the aircraft types were to include Fixed-Wing Aircraft (FWA) and Rotary-Wing Aircraft (RWA). However the results presented here only include FWA due to the lack of availability of Soar-based RWA at the time the benchmarking was performed.

The tests consisted of populating the synthetic environment with up to 532 ground vehicles placed in a standard JSAF terrain representation. The ground vehicles consisted

[11] JSAF is the new name for the CGF application (SAF-based) developed during the STOW ACTD. JSAF functionality continues to evolve and is used to support various DoD M&S domains.

of Friendly (Blue) and Opposition Forces (Red). For the TacAir-Soar testing, two series of test were run, one with the 532 ground vehicles and a second series with 270 ground vehicles. During the tests, some of the ground forces were given simple missions involving entity movement. The computing platforms used to test the FWA were 400 MHz Pentium II-based personal computers configured with 384 Mbytes of RAM and 700 Mbytes of swap space. The computers used to generate the ground forces were 200 MHz PII-based PCs with 256 Mbytes of RAM. The computers were connected using 10base-T Ethernet in a network topology designed to optimize network traffic in a WAN/LAN HLA environment. The JSAF applications used for the tests were run in HLA mode (as opposed to DIS mode) using the STOW RTI-s, the RTI implementation specific to the STOW application environment.

5.3.1 Taskframe Testing

The capacity tests for taskframe-based FWA consisted of a SAF scenario with fifty taskframe entities of different aircraft types. The aircraft were assigned missions normally associated with their aircraft type and included tanking, EW, DCA, CAS, etc., the goal being to provide realistic (representative) interactions among ground and air entities. Initially, all fifty aircraft were loaded, but were idle in the battlespace. This means the simulation was updating the state information for each entity. Since the entities were idle, the workload generated by each was minimal. At periodic intervals each aircraft (or section of aircraft) initiated its mission, increasing the simulation workload on the test platform. After all entities were launched, the simulation was allowed to run in a steady state for approximately ten minutes. The taskframe entities were then deleted from the simulation at regular intervals until the FWA entity count was zero. The total time for a single execution was approximately 60 minutes. This test was repeated five times and the results recorded.

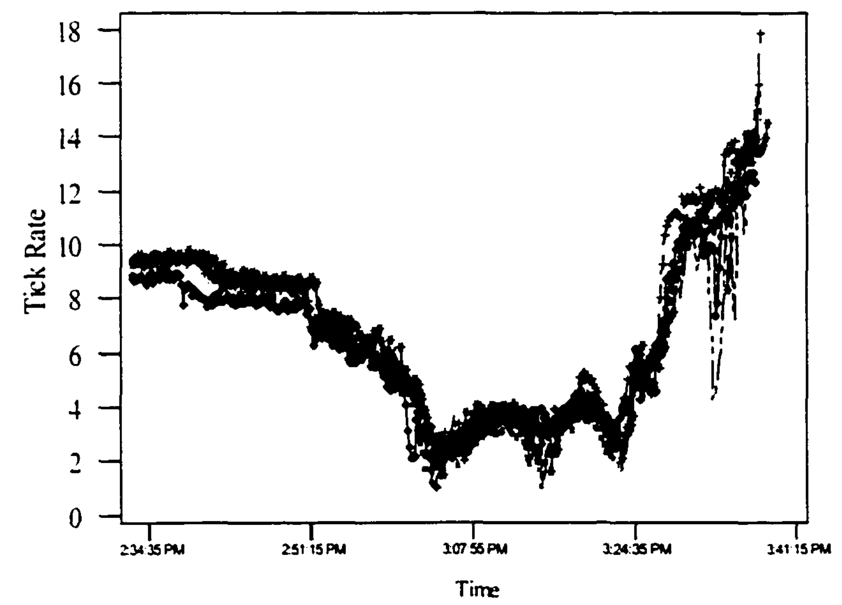


Fig. 21. Entity state update (tick) rate for FWA taskframe test.

The results of the testing indicate that between 40 and 50 taskframe entities (FWA) can be simulated on a single JSAF simulation engine operating as a backend (i.e., no GUI). Performance data from the five executions were analyzed using Analysis of Variance (ANOVA) to determine the mean tick rate and entity state update performance of the simulation for the given workload. For the purpose of the results presented here, *Tick rate* is defined as the number of times per second (Hz) that an entity has its state information updated. *Entity State Update Performance* is defined as the percentage of the total number of entities that are meeting their prescribed update rates over a window of time: in this case the prescribed update rate is 2 Hz and the sampling window is 60 seconds. For the set of execution samples, ANOVA assumptions were verified by examining the residuals output. In some cases the sampling assumptions were marginally acceptable. More important, however, is the practical significance of the observations presented in Figure 5.8 Specifically for all five samples (executions), the observed entity tick rates were similar.

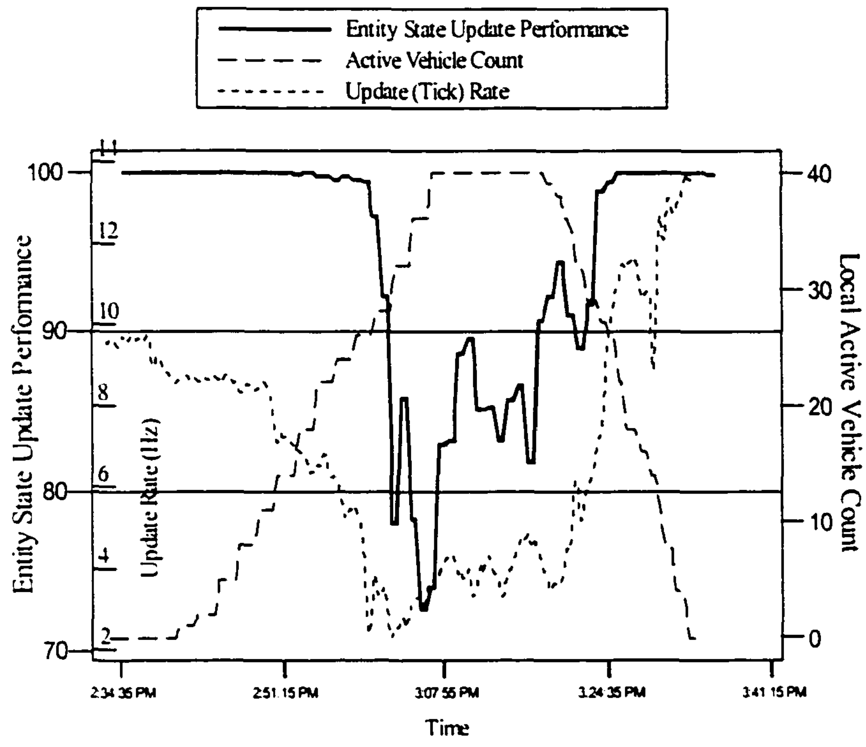


Fig. 22. Entity state update performance and update (tick) rate for a sample FWA taskframe execution.

Figure 5.9 shows the entity tick rates and the entity state update performance for one of the executions. Active entity count is shown along the right-hand Y-axis. Entities were considered active while executing their missions. Note that the active entity count only went to forty for the tests because ten of the fifty total aircraft were RWA that sat idle because they could not be assigned missions. These idle RWAs however, were still ticked and therefore did contribute to the simulation workload. The large drop in update (tick) rate and the corresponding drop in the entity state update performance coincide with the peak levels of interaction and aircraft activity during run-time. The data shows that, assuming the highest levels of entity activity and interaction, it is possible to simulate as many as forty active FWA (and additionally tick ten low activity aircraft) and still maintain acceptable update rates (4 HZ) and overall entity state update performance (ninety percent). The statistical analysis indicates that, with 95 percent certainty, one can

TABLE 6
Update Rate and Entity State Update Performance For TacAir-Soar Scenarios

	532 Remote Ground Vehicles				260 Remote Ground Vehicles			
	4 agents	6 agents	8 agents	10 agents	4 agents	6 agents	8 agents	10 agents
Update (Tick) Rate	5.7 Hz	4.0 Hz	2.6 Hz	1.7 Hz	5.3 Hz	3.8 Hz	3.1 Hz	2.3 Hz
Entity State Update Performance	99.8 %	89.3 %	73.0 %	37.0 %	99.8 %	96.3 %	82.4 %	61.5 %

expect an average tick rate of between 6.6 and 6.8 Hz. The average entity state update performance will be between 92.64 and 93.65 percent.

5.3.2 TacAir-Soar Testing

The capacity tests for the TacAir-Soar agents consisted of scenarios with four, six, eight, and ten agents per machine. All aircraft were initialized to execute Strike missions (i.e., attacking ground targets) since these missions are known to generate the greatest simulation/Soar workload for a given entity count. The aircraft (F-16Cs with laser-guided bombs) were placed at initial points approximately ten minutes (real-time) from the targets (ground OPFOR). They would fly to the targets and after dropping their ordnance, would egress back to the initial point. As mentioned, for the TacAir-Soar testing, two series were run with the same scenario just described. The first series consisted of 532 ground vehicles and the second series consisted of 260 ground vehicles. The objective of running the second series was to assess the impact of the ground entity count on simulation/Soar performance. The time for a single execution was fifteen minutes and each execution was repeated five times for each test series.

The results of the TacAir-Soar testing are presented in Table 5.2. There appeared to be a marginal improvement in run-time performance for the case with fewer ground vehicles;

For 532 ground vehicles, six TacAir-Soar agents per simulation engine appear to be the optimal number given the compute resources employed. For the case with 260 ground vehicles, six to eight agents can be run on the same simulation engine. It is important to note the agents still continue to run their missions despite overloading and that in some overloaded cases agent behavior may still be valid.

The problem is we cannot be confident in the validity of the interactions because of the possibility of reordering and delays in simulation. Although anecdotal in nature, observations made during the tests did indicate better simulation and agent behavior during the test series with 260 ground vehicles and it appeared that the Soar agents actually achieved a higher ratio of kills when delivering ordnance. These observations, however, were not quantified. Performance data from the five executions of each scenario, from both test series, were analyzed using ANOVA to determine the mean tick rate and entity state update performance of the simulation for the given workload. For the set of execution samples, ANOVA assumptions were verified by examining the residuals output. In some cases the sampling assumptions were marginally acceptable. As shown during the discussion of the FWA taskframe test results, Figure 5.10 illustrates how the observed entity tick rates are strongly correlated among sample executions. This particular plot shows the execution samples for eight agents with 532 remote ground vehicles.

Figure 5.11 shows a time series of update rates of the Soar agents from one execution of each test with 260 remote ground vehicles. The plotted data are actually the moving averages of the update rates over the duration of the execution. The pronounced drop in the tick rate is indicative of the final approach after acquiring the target, the release of the ordnance, followed by the start of the egress from the target area (where the tick rate starts to raise).

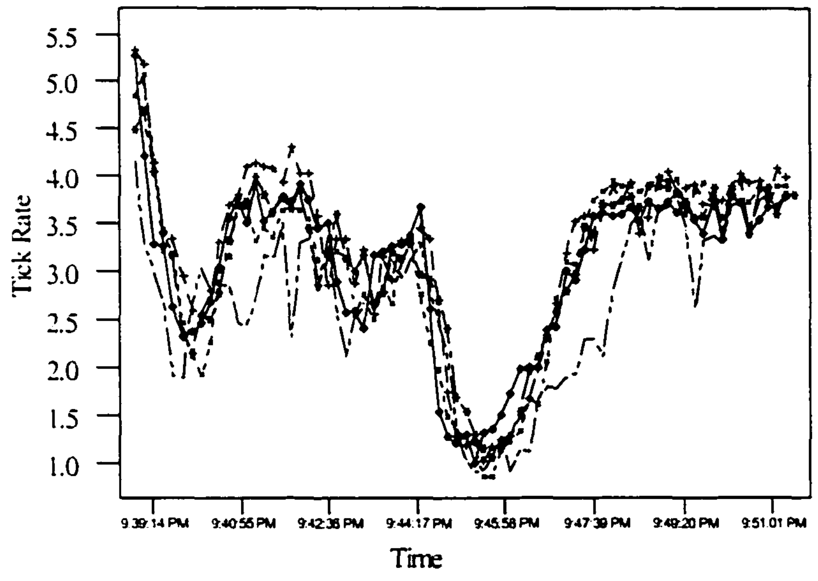


Fig. 23. Entity state update (tick) rate for 8 TacAir-Soar agents; 532 remotes ground vehicles.

The results of the study clearly show the increased computational costs associated with more sophisticated Soar mission behavior modeling. But for training systems with constrained numbers of operators/trainers (such as in AVCATT-A), fully autonomous behavior modeling using technologies such as TacAir-Soar can significantly reduce operator workload. This fact becomes even more critical given the goals to create larger and more realistic training exercises, translating to higher entity counts and increasingly complex simulation environments. AVCATT-A can benefit from both behavioral modeling technologies by using benchmark studies as presented here, to determine an appropriate ratio of fully autonomous and semi-autonomous entities during scenario and system design. TacAir-Soar intelligent agents can generate highly realistic RWA and FWA representations while significantly reducing the required number of simulation operators to meet exercise objectives. Less costly taskframe-based entities can then be used to provide greater density to the synthetic battlespace, enhancing the realism of the synthetic environment.

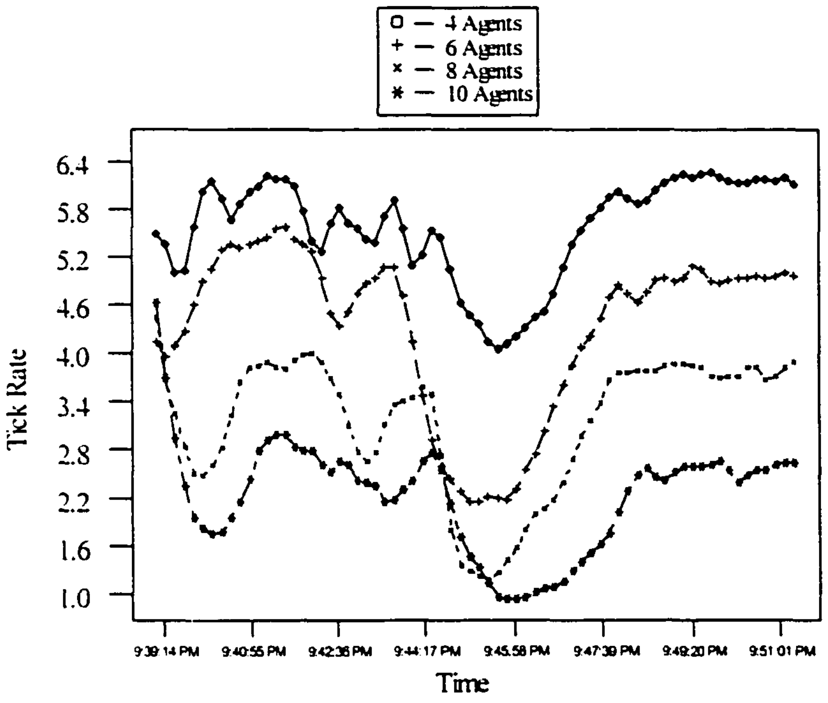


Fig. 24. Entity state update (tick) rate for TacAir-Soar agents; 260 remote ground vehicles.

The benchmark testing results obtained during this use case provided useful information to decision-makers trying to make reasonable assessments about the capabilities of existing hardware and software to provide credible entity-on-entity level simulation to augment the AVCATT-A training system. By varying critical factors in a controlled test environment and measuring their impact on performance, scenario design and configuration planning for CGF becomes a more quantifiable process.

5.4 U.S Air Force Distributed Mission Training (DMT)

The last use case for the research presented in this thesis is to support assessments related to technology insertion, specifically to understand the impact of using STOW technology to link virtual cockpit (human-in-loop) simulators. The U.S. Air Force is currently defining requirements for a new training system and plans to use emerging distributed simulation technologies to enhance the effectiveness of the training environment. The U.S. Air Force has previously relied on aircraft as the primary mechanism for mission training. Peacetime constraints have now limited the funds available for adequate mission training using real aircraft. To maintain an adequate level of military readiness, the DoD is exploring the increased use of simulation to support operational training, such as the Air Force's Distributed Mission Training (DMT) program. DMT is a shared training environment comprised of live, virtual (manned), and constructive (computer generated) simulations. The principal technical objective of DMT is to create high fidelity manned simulators networked with other air, ground, and sea forces in a realistic synthetic battlespace. The simulators will support the full spectrum of training from unit-level, small team training, to theater-level joint training exercises, and can potentially be employed to support analyses, and test and evaluation.

5.4.1 Technology Insertion

Following the successful completion of the STOW ACTD, DARPA initiated follow-on work to enhance and improve the SAF implementations, increase overall STOW system performance, and transition STOW technology to other programs. One such effort was the *Distributed Mission Training (DMT) Experiment.* Various technical challenges exist regarding the integration of constructive, virtual, and live simulations intended for use in the DMT program and were discussed in Section 1. One technical issue is interoperability among DIS- and HLA-based systems. The key implementation component to integrate STOW technology with existing DIS-based virtual cockpit simulators is an HLA Interface. DARPA developed the interface to share virtual and constructive simulation data; translating between the HLA/RTI-based data packets associated with the STOW synthetic battlespace and the DIS-based data packets used by the Air Force Research Laboratory (AFRL) virtual simulators located in Mesa, Arizona.

A primary goal of the DMT Experiment was to characterize the impact of integrating STOW technology (i.e., HLA-based synthetic forces and synthetic environment) and a DIS-based virtual training environment, specifically, the throughput and latency as seen by the virtual component of simulation state data generated by the synthetic forces and synthetic environment components of the STOW federation. The principal performance objective of the integration was to provide federate state data to the virtual simulation hosts at a sufficient rate to maintain the quality of the training experience. Assessing the success or failure of meeting this objective required evaluating run-time performance by identifying the system's shared hardware and software resources, understanding the utilization and contention for these resources, and quantifying the delays imposed by using these resources.

The data update rate requirements for the virtual simulator's image generator had an upper bound estimated to be 60 Hz, however, the data rate at the virtual simulator's network interface was significantly less than the image generator's access. Thus, to meet the performance objectives, HLA Interface updates needed to occur at a rate that was consistent with the data requirements for the virtual simulation host. The performance criteria for this evaluation required measuring the speed and utilization of the DIS⇔HLA data interface. Metrics were latencies, throughputs, and effective bandwidth at various levels in the protocol (communications) stack. End-to-end performance was important and timing data was aggregated to provide a measure of the overall communications performance. The principal mechanism chosen to support this performance analysis was PerfMETRICS. Its functionality was extended to provide remote control of all HLA interfaces and display the relevant performance information.

5.4.2 Assessing the Performance Impact – High-level

Original test plans were extensive but contention for the virtual cockpits by other tasks within the DMT Experiment and other programs required restricting the test time. Two test scenarios were designed to characterize latencies through the HLA Interface as a function of workload (i.e., data packet throughputs, the number of virtual cockpits, and the number of Computer-Generated Forces, or CGF). The latency measurements include transport delays, delay across the RTI (HLA publications/subscriptions), data packet translation delays, and end-to-end delay (as seen at the HLA Interface).

The first scenario was designed to capture data throughput rates representative of different levels of pilot activity during air-to-air engagements. Specifically, low, medium, and high-levels of maneuvering providing different rates of changing aircraft position and orientation. The second scenario was designed to observe the impact of ground CGF and STOW synthetic natural environments on the relevant performance metrics. Ground CGF consisted of Blue and Red forces laid down in a region of high-resolution terrain. Buildings, tank ditches, and other STOW dynamic terrain objects were also used to populate the synthetic battlespace. The test scenario consisted of sending TacAir-Soar agents on a strike mission and having the virtual cockpits 'chase' the agents to observe the environmental enhancements provided by STOW SE. The premise of the scenario design was that the SE would induce the additional processing and communications overheads desired for this performance study.

The PerfMETRICS monitoring system was extended to specifically measure network latencies and data throughputs for the DMT Experiment and also provides feedback to and control of monitored applications. PerfMETRICS was used to gather all data except the LAN and WAN transport delays and the bandwidth utilization on the HLA network. For the transport delay measurement, a program was implemented using an *echo-based* measurement mechanism representative of multi-cast communications up to the point where the RTI receives data from a multicast port. Two instances of the program were run, one instance collecting WAN traffic from virtual cockpit simulators located at the Theater Air Command and Control Simulation Facility (TACCSF) in Albuquerque, New Mexico, and another collecting LAN traffic at AFRL. Bandwidth utilization on the HLA network was captured using a PC-based network analyzer.

Results from data collection at one of the HLA Interfaces during the air-to-air tests are depicted in Figures 5.12 – 5.15. Note that data for all HLA Interfaces were similar, as was expected. The data presented are from the third air-to-air test where pilot activity was

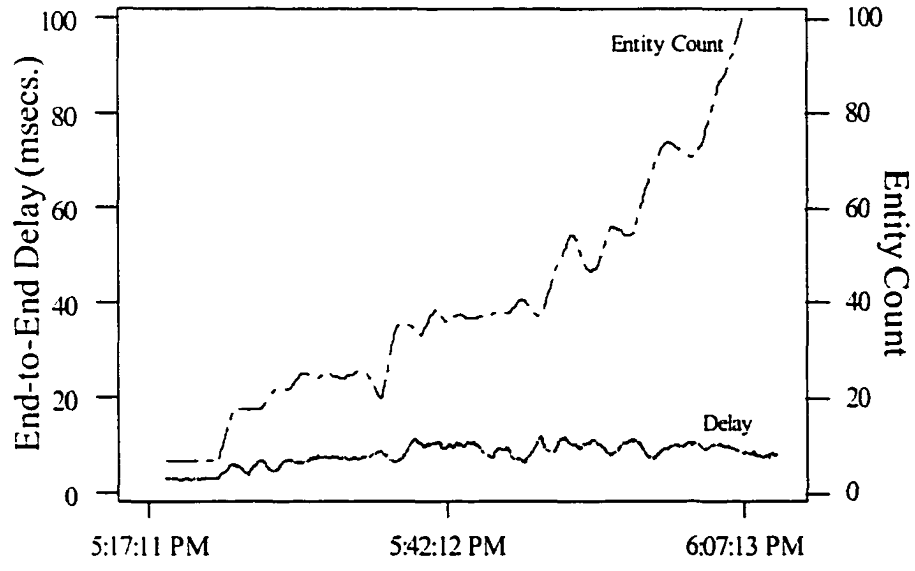
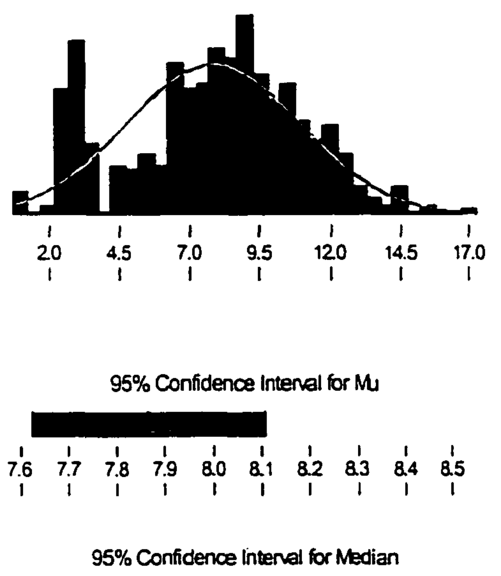


Fig. 25. Latency vs. entity count (Air2Air).



```
Anderson-Darling Normality Test
  A-Squared:         5.946
  P-Value:           0.000

  Mean               7.86688
  StDev              3.08215
  Variance           9.49966
  Skewness          -2.1E-01
  Kurtosis          -5.4E-01
  N                  632

  Minimum            0.7600
  1st Quartile       6.1250
  Median             8.2300
  3rd Quartile       9.9075
  Maximum           16.9300

95% Confidence Interval for Mu
  7.6261             8.1076
95% Confidence Interval for Sigma
  2.9211             3.2622
95% Confidence Interval for Median
  7.9800             8.5025
```

Fig. 26. Latency description (Air2Air).

representative of close-in combat such that the aircraft exhibits rapid changes in position and orientation. Note that this level of activity was not sustained throughout the test. The data as presented in Figure 5.12 indicates that, for the given scenario and workload (up to 102 entities), end-to-end delays as seen by the interfaces are loosely correlated with entity

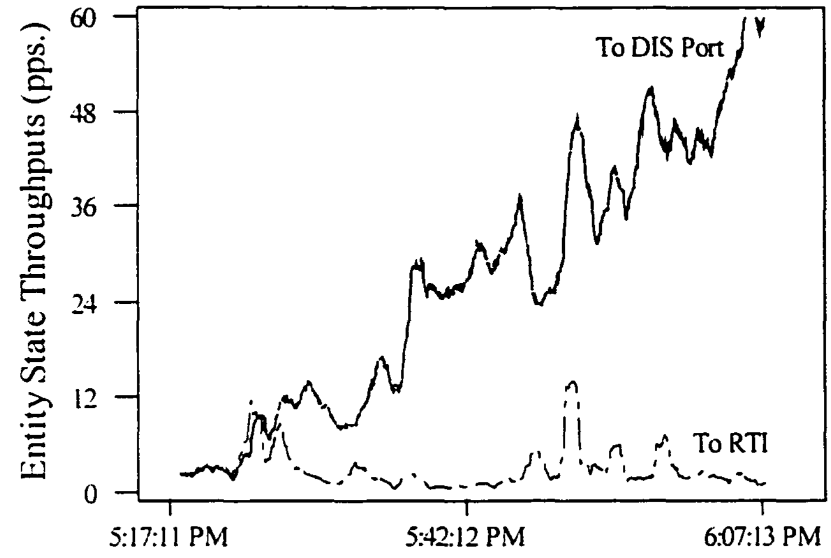
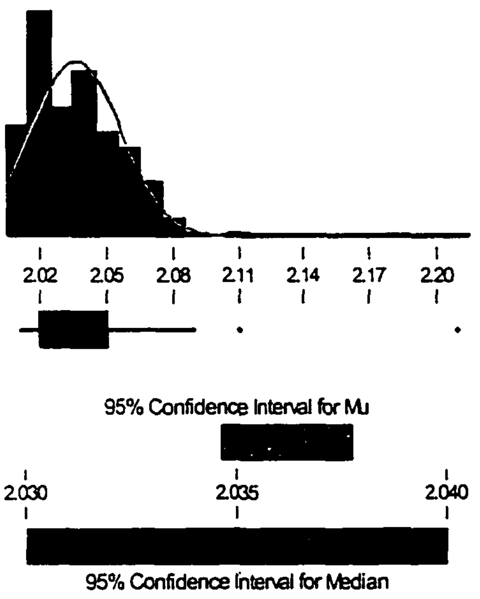


Fig. 27. Entity state throughputs (Air2Air).

count. Figure 5.13 provides descriptive measures of end-to-end latency as defined for this study, namely the time it takes a packet to be transmitted between the DIS ports of any two HLA interfaces linking the associated virtual cockpits. One significant observation of the data in this figure is the deviation in latencies, where over fifty percent of the values range between six and ten milliseconds. This is inconsistent with preliminary studies where deviations about the mean latency were relatively 'tight'. In the broader scope, the update rates provided to the simulators were sufficient for providing the pilots with a good visual perception of the synthetic airspace.

The data presented in Figure 5.14 and 5.15 are useful for examining interface performance. For the given scenarios, entity state data dominated the throughput at the HLA Interfaces. Figure 5.14 shows the expected increase in throughput as a result of increasing numbers of entities. The throughput from the cockpit is fairly constant; the plot shows correlation between the DIS and RTI throughputs (observed as spikes in the plot's moving averages). This is most likely an artifact of the pilots reacting to each



Anderson-Darling Normality Test

A-Squared:	14.378
P-Value:	0.000
Mean	2.03620
StDev	0.02040
Variance	4.16E-04
Skewness	1.45036
Kurtosis	7.53854
N	632
Minimum	2.01000
1st Quartile	2.02000
Median	2.03000
3rd Quartile	2.05000
Maximum	2.21000

95% Confidence Interval for Mu

2.03461 2.03780

95% Confidence Interval for Sigma

0.01933 0.02159

95% Confidence Interval for Median

2.03000 2.04000

Fig. 28. Avg. access to DIS port (Air2Air).

other's maneuvers, creating an increased rate of change in aircraft orientation and position and resulting in increases in the entity state outputs at both interfaces.

The significant point about this is the run-time performance of the HLA Interface did not degrade as throughputs increased during these tests. The current implementation of the HLA Interface is configured to read the DIS UDP port at a 500 Hz rate (every 2 msecs.). Figure 5.15 shows the average access delay for reading the port, which deviates only on the order of microseconds despite an increasing load of traffic through the interface. This indicates that for the given workload the HLA Interface is able to complete all other processing and return to read its DIS port within microseconds of its scheduled time.

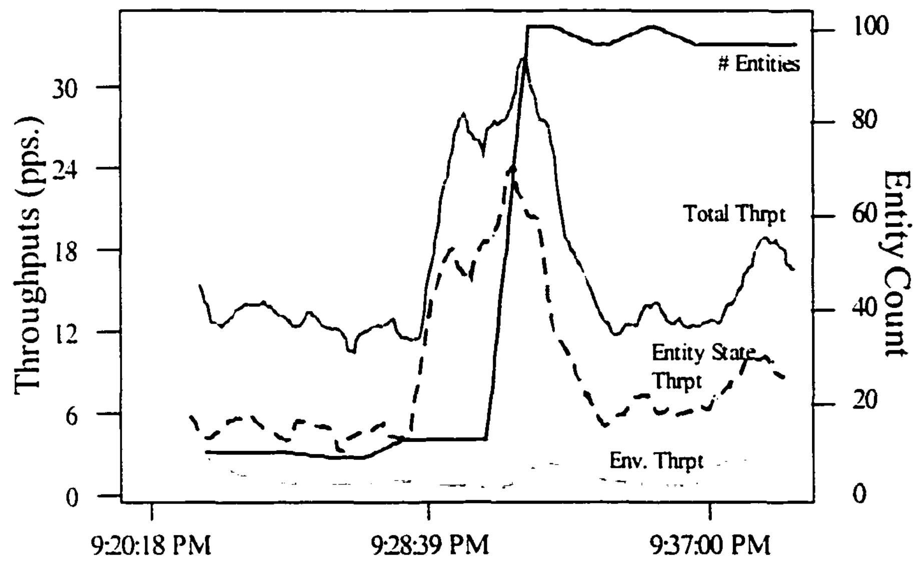
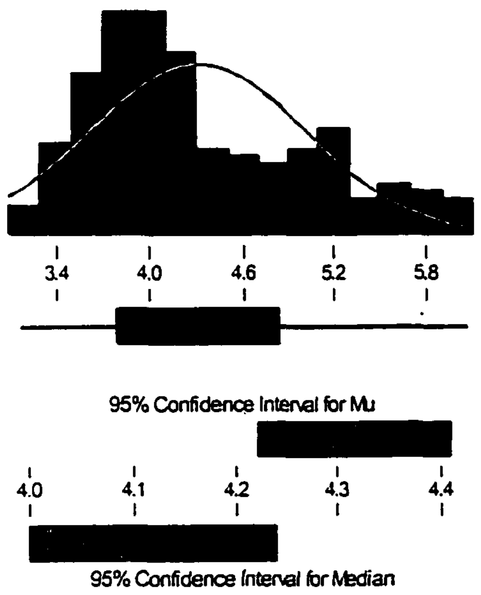


Fig. 29. Throughputs vs. entity count (Air2Grnd).



Anderson-Darling Normality Test

A-Squared:	5.031
P-Value:	0.000
Mean	4.31514
StDev	0.69980
Variance	0.489725
Skewness	0.730218
Kurtosis	-3.9E-01
N	213
Minimum	3.17600
1st Quartile	3.78800
Median	4.15400
3rd Quartile	4.82867
Maximum	6.06333

95% Confidence Interval for Mu

4.22062	4.40966

95% Confidence Interval for Sigma

0.63906	0.77341

95% Confidence Interval for Mediar

3.99926	4.23861

Fig. 30. Latency description (Air2Grnd).

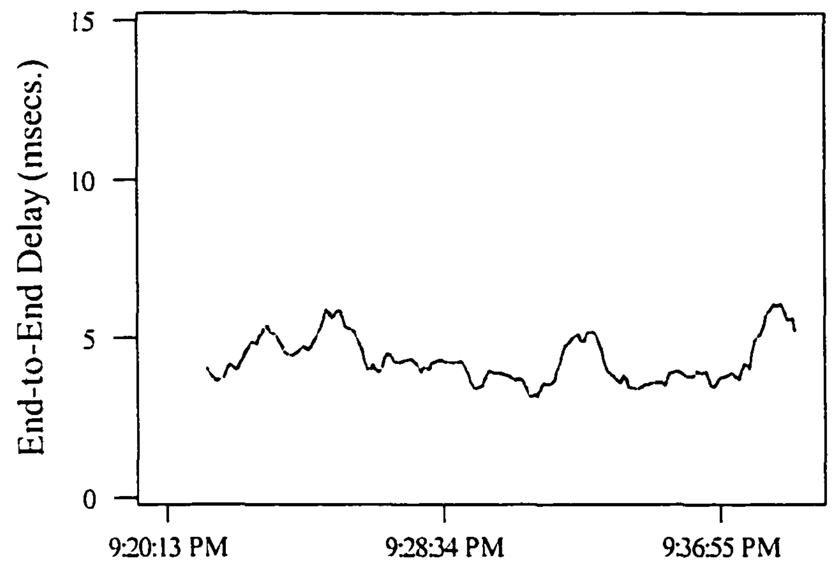


Fig. 31. End-to-end latency (Air2Grnd).

Figure 5.16 shows data gathered from Viper 3 (one of the F-16C virtual cockpits) during the last air-to-ground test. This test consisted of a workload generated by seven TacAir-Soar agents, two virtual cockpits, over ninety ground-based CGF companies of Red and Blue tanks, and SE data that included dynamic terrain (buildings, road craters, etc.), diurnal effects (night and day changes), smoke plumes, and road craters. The data shows total, Entity State, and environmental PDU throughputs to the HLA Interface DIS port. Additionally, the plot shows the influence of the STOW CGF. The end-to-end latency, as seen by the interfaces, is better than the air-to-air test. Measures are presented in Figure 5.17 and 5.18. The data show a mean end-to-end latency of 4.3 milliseconds and ranges between three and six milliseconds. It is possible that the lower latency in this test could be attributed to the lack of additional traffic from TACCSF, which was also conducting tests during the air-to-air tests.

5.4.3 Assessing the Performance Impact – Low-level

LAN transport delays are presented in Table 5.3. The results reinforce the point that in a LAN environment most of the data latency is introduced at the application level. For

STOW federates that means processing and queuing delays in the RTI and application code. For the HLA Interface, that predominately means processing and queuing delays within the RTI since processing time associated with the data translation is on the order of microseconds. Transport delays for the network connection with TACCSF are reported in Table 5.4. As expected, the 'time of flight' on the WAN is significantly greater than on the LAN, averaging around twelve milliseconds during all the tests. TACCSF was simulating an AWACS aircraft and only reported data during the air-to-air test. The average latency across the RTI was 32.4 milliseconds. However, this was derived from a very small sample set during one test and should not be used as an estimate of expected application latencies given workloads similar to these tests.

Table 5.5 presents the bandwidth utilization on the HLA network during the four test scenarios. The significant point is that the scenarios used in this study did not generate traffic that came even close to taxing the 10 Mbps network; the air-to-air scenarios exhibiting close-in combat maneuvering generated bandwidth that approached the capacity of the T1 data link used to connect sites on the WAN (ARFL and TACCSF). The higher numbers associated with the air-to-air test are most likely a result of the additional test traffic generated from TACCSF and the higher entity state throughputs associated with the virtual cockpits and CGF.

One of the primary concerns associated with this use case was the performance of the HLA Interface, since it had stringent requirements for providing state data at a rate that would not adversely impact the visual systems used in the virtual cockpits. The data presented is significant because of widespread skepticism about integrating HLA and the RTI with virtual cockpits having stringent real-time processing requirements (driven by the man-in-loop visualization requirements). The data gathered during the study show that even in the presence of increasing simulation workloads and network traffic, the HLA Interface can adequately translate and transmit STOW/HLA data to the virtual cockpits. Data translation times are on the order of microseconds, and the interface implementation reduces latency by servicing its DIS and RTI ports at a 500 Hz rate.

TABLE 7
LAN Transport Delays (msecs.) – HLA Network

Test	Avg.	Min.	Max.
Air2Air -High Movement	.22	.20	.54
Air2Grnd -no CGF, no SE	.22	.21	.37
Air2Grnd -no CGF, SE	.27	.21	1.84
Air2Grnd -CGF, SE	.32	.21	3.6

TABLE 8
WAN Transport Delays (msecs.) – HLA Network

Test	Avg.	Min.	Max.
Air2Air - High Movement	12.99	11.01	35.51
Air2Grnd - no CGF, no SE	12.02	11.08	15.93
Air2Grnd - no CGF, SE	12.94	11.15	39.91
Air2Grnd - CGF, SE	13.15	11.10	25.43

TABLE 9
Bandwidth Utilization – HLA Network

Test	Percent Usage (%)
Air2Air – High Movement	11.81
Air2Grnd – no CGF, no SE	3.37
Air2Grnd – no CGF, SE	2.62
Air2Grnd – CGF, SE	5.4

The performance monitoring and measurement framework presented in this thesis was successfully used as a basis to develop the test and measurement methodology for the DMT Experiment. The raw performance data was presented at a level of abstraction sufficient for detailed analysis of resource utilization to relate federation performance to scenario workload so as to be meaningful to persons making decisions about the suitability of STOW technology for integration with real-time virtual cockpit simulators.

5.5 Use Case Summary

This section has presented a series of case studies, each highlighting the use of the PerfMETRICS monitoring system to gather data meaningful to different decision-makers for different aspects of a distributed simulation environments including model development, exercise configuration and control, resource utilization assessments, and capacity planning. The monitoring has included simulations and other applications that integrate live, virtual, and constructive (simulation) components. The variety of relevant performance information includes system level (hardware and operating system) performance metrics, performance measures associated with simulation infrastructure, modeling performance, and networking performance metrics. The execution environments include training, operational, analysis, and acquisition for the U.S. Navy, U.S. Army, and U.S. Air Force.

Table 5.6 provides an abbreviated list of the performance metrics used to support performance analysis and assessments during these case studies. The application of the framework presented in Section 3 and the PerfMETRICS monitoring system described in Section 4 support the performance analysis activities during these case studies and also support the conclusions for this thesis research in the next section.

TABLE 10
Use Case Metrics Summary

MODEL DESIGN AND TESTING

1) IP PPS as F(time)
2) Remote Vehicle Count by Mission Type
3) Remote Radio Count by Mission Type
4) RTI Tick Processing by Mission Type
5) Correlation Values for above

SCENARIO CONFIGURATION

6) Available workstations
7) Mission types
8) Entity count
9) Number of Missions

RESOURCE MONITORING

10) Average Utilization of available simulation engines
11) Average Entity Update Rate by Mission Type

LOAD SCHEDULING

12) Entity Update Rate
13) Entity Count
14) Memory Utilization

TECHNOLOGY ASSESSMENT & CAPACITY PLANNING

15) Local Entity Count by Behavioral Model Type
16) Remote Entity Count
17) Entity State Update Performance by Behavioral Model Type
18) Entity Update (Tick) Rate by Behavioral Model Type

TECHNOLGY INSERTION IMPACT

19) Entity Count as F(time) by Mission Type
20) End-to-end delay as F(time) by Mission Type
21) Average Latency and Variance as F(time) by Mission Type
22) Entity State Throughput as F(time) by Mission Type
23) Average Network Access Rate
24) Total Throughput
25) Environmental Throughput
26) LAN Transport Delays
27) WAN Transport Delays
28) Bandwidth Utilization

SECTION 6

CONCLUSIONS

The origins of this research stem from the practical need to understand the run-time behavior of an inherently complex distributed computing domain, specifically that of advanced distributed simulations. The complexity exists due to the proliferation of enabling technologies used in ADS environments, the abstract nature of modeling real-world environments, and the interaction of various factors that affect the run-time behavior of distributed computing applications. Different persons associated with the design, configuration, and control of distributed simulations need to understand the impact of decisions made regarding the allocation and use of the various logical and physical resources comprising the distributed run-time environment. Characterizing run-time behavior is a key aspect to providing decision-makers with an understanding of technology factors contributing to performance bottlenecks in these environments, and providing a mechanism to monitor and collect run-time performance data supports meaningful assessments about the degree to which simulation objectives are achieved.

The original proposal for this research maintained a hypothesis that there exists a generalized framework for performance analysis of distributed simulations. The objectives intended to define an abstract representation of all distributed simulations and to present a series of formal techniques that support a direct mapping from the performance abstraction to a set of performance metrics. In reality, this proposition far exceeds the scope of this thesis research. Although such an abstraction may in fact exist, the diversity of application domains for distributed simulations and an almost infinite number of goals for the evaluation of run-time performance make the proof of this hypothesis a daunting task. The specification of a credible, generalized abstraction of distributed simulation performance and the definition of a robust set of related performance metrics would be a difficult goal to achieve in any time period.

However, one achievable aspect of the original proposal was the use of DIS and HLA simulation environments as a case study for demonstrating the utility of a framework for the analysis of distributed simulations. Initial research, focused on the original thesis objectives, quickly led to the realization that DIS and HLA simulations were sufficiently complex M&S environments and provided boundary conditions for a practical specification of a performance analysis framework. The well-defined and formalized DoD functional areas that use M&S applications and the common set of real-world representations manifesting themselves within DIS and HLA simulations delineate the characterization of distributed simulation performance to those run-time performance factors related to the simulation of real-world objects and some space/time analog of the real-world environments in which they interact. The description of real-world objects, object interactions, and environments is meaningful to persons making decisions about the design, configuration, and control of DIS and HLA simulations. These three activities and the recurring focus on objects and their interactions provide the basis for proposing an architectural framework for the performance analysis of DIS and HLA simulations. The research objectives presented in Section 1 of this thesis are modified from the original proposal only in the context of constraining the definition of a performance analysis framework to its application in a M&S domain that demands practical solutions to time-critical problems.

The use cases presented in Section 5 illustrate the utility of a unified framework (defined in Section 3) to provide a cohesive process for DIS and HLA performance analysis; specifically to be able to: 1) delineate the system boundaries based on the execution domain (e.g., training, analysis) and analysis objectives (e.g., model design, exercise control), 2) select the appropriate performance metrics, 3) consider the impact of different workloads (scenarios) on run-time performance, and 4) analyze, interpret, and present the performance information to decision-makers. The remainder of this section provides an evaluation of the work accomplished during this thesis research as it relates to the research objectives presented in Section 1.3 and concludes with a discussion of its significance and the future direction in which this research should move.

6.1 Evaluation

Objective 1. Define a framework useful for characterizing DIS/HLA simulation performance. The framework shall include a conceptualized view of performance in the context of DIS and HLA simulations, and a taxonomy of performance measures useful to different decision-makers involved with the DIS / HLA life-cycle.

The framework defined in Section 3 and represented in Figure 3.4 provides a unified architecture (conceptualized view) for performance analysis of DIS and HLA simulations. The framework considers relevant execution domains (training, analysis, acquisition, and operational) as outlined in the U.S. DoD Modeling and Simulation Master Plan which employ, or are candidates for employing DIS and HLA-based simulation technologies. Although not explicitly shown in the framework diagram, the manner in which simulation technologies are applied within each domain manifests itself as an actual scenario workload, another significant aspect affecting run-time performance and explicitly considered in the analysis framework. Refinement of the framework as it applies to each domain results in a definition of the principal activities associated with any simulation life-cycle, specifically the design (and development), the configuration (and planning), and the run-time control (and monitoring) of the simulation environment.[12]

The framework considers performance measures relevant to DIS and HLA-based environments based on the physical and logical resources and services used to implement a distributed simulation environment. The performance measures are categorized by architectural layers (abstractions) of different simulation components and result in the definition of system level metrics to capture lower levels of system performance (network and operating system), simulation infrastructure metrics to capture overheads associated

[12] Requirements and project management activities can also be considered principal activities in a simulation life-cycle (or more generally any software development life-cycle), but they are not considered in the framework because of their obscure, indirect influence on run-time performance of a simulation environment.

with executing simulation models (the implementation costs of using simulation), and model performance metrics (object-based physical and behavioral representations) to characterize the run-time performance of the different model implementations. The framework establishes the linkage between the identified performance measures and their intended use (analysis to support design, configuration, and control activities) by defining an interface (the PerfMETRICS implementation in the context of this research) to provide a conduit for run-time acquisition of the relevant performance metrics and their aggregation into meaningful information for decision-makers.

Regarding framework utility, DIS are based on an IEEE standard that specifically embeds the notion of physical and behavioral representations into an application-level protocol. This facilitates the applicability of the framework's taxonomy of performance measures to all DIS-based environments and when considered in conjunction with a common set of analysis objectives (i.e., supporting design, configuration, and/or control activities), provides a flexible architecture for performance analysis. The HLA embodies the latest evolution of distributed simulation protocols (within the U.S. DoD) and a generally accepted view is HLA is a surrogate for legacy and new simulation environments that would otherwise be using DIS. As such, the objectives for performance analysis between DIS-based and HLA-based environments will be similar, except for additional requirements to understand the costs associated with the HLA RTI and the intrinsic services it provides for managing the distributed simulation environment. This fact suggests the framework is also generally applicable to HLA environments. Finally, from a pragmatic standpoint, and perhaps most significantly, the conceptual soundness of this framework is demonstrated by its continued acceptance and use to support real-world DoD M&S-based exercises; providing a standard set of DIS and HLA performance measures that are relevant and useful for analysis in different simulation domains, examples of which have been presented as use cases (that span a four year duration) in Section 5 of this thesis report. Table 6.1 summarizes the programs and applications that have utilized the framework, and specifically PerfMETRICS, to support modeling and simulation design, configuration, and control activities.

TABLE 11
Summary of Framework and PerfMETRICS Utilization

DARPA STOW ACTD	(U.S. DoD R&D program)
JSAF - Air Component	
JSAF - Marine Corp Component	
DMT Experiment	(U.S. Air Force - Constructive and Virtual Environment)
JSAF	
HLA⇔DIS Interface	
BFTT Air Management Node	(U.S. Navy - Constructive and Live C4I Environment)
JSAF	
HCI (Human Computer Interface process)	
HLA⇔DIS Interface	
AVCATT-A	(U.S. Army - Constructive and Virtual Environment)
JSAF	
ModSAF	
OTBSAF	

Objective 2. Develop a measurement, monitoring, and analysis infrastructure useful for supporting DIS and HLA simulation performance analysis.

This thesis research has resulted in the design and development of the PerfMETRICS monitoring system. PerfMETRICS satisfies basic requirements for software-based monitoring in a real-time distributed simulation environment. It provides centralized or distributed monitoring of networked simulation engines, the workstations hosting those simulations, and the specific models executing inside the simulation engines. It supports data collection and provides mechanisms to support data analysis, and data presentation. The PerfMETRICS communications infrastructure also supports dynamic control of the monitoring environment, error notification and logging, and execution control of applications (including simulations) instrumented with the PerfMETRICS monitoring code.

From the beginning, the PerfMETRICS design included the same basic taxonomy of performance measures presented in this thesis. The architecture has proven to be extensible and the implementation robust enough to meet changing analysis requirements for programs such as the STOW ACTD, the DMT Experiment, BFTT AMN, and

AVCATT-A. Since the collection daemon is loosely-coupled with the monitored applications, it does not have an architectural dependency meaning that PerfMETRICS is readily amenable to monitoring DIS or HLA-based applications, or both within the same exercise (when a DIS⇔HLA Interface is present to support this configuration).

The PerfMETRICS monitoring system does not contain sophisticated data analysis mechanisms or data visualization tools, and in itself cannot be considered a decision-support tool. It is best viewed as a "lightweight", yet robust monitoring and data collection tool sufficient for providing timely performance information to persons trying to make decisions about how to design and implement simulation models, configure the simulations and associated scenarios, and monitor and control the run-time execution environment. PerfMETRICS is but one alternative to support DIS and HLA run-time performance analysis but it proved to be very effective in its application to the "real" environments presented in the use cases. Research results from its use are not only reported in this thesis, but have also been disseminated to a broad simulation development community via conference publications, after-action reviews, and technical interchange meetings.

Objective 3. Relate the costs of obtaining the performance information for use in both dynamic and static performance analyses in terms of the intrusiveness of run-time monitoring and measurements of DIS/HLA simulations.

Discussing the intrusiveness of run-time monitoring really means assessing the cost of data collection weighed against the value of that data and the possibility that data collection will change model or simulation behavior. Thus a real need exists to understand monitoring and measurement overheads so appropriate choices can be made. Quantitative studies were initiated early on during PerfMETRICS development. Native UNIX code profilers were used as a means to measure the impact of specific instrumentation strategies. Performing these studies throughout the development evolution resulted in the recognition of three principle factors when making tradeoffs in measurement and monitoring costs and the value (essentialness) of data. They are: *data* *granularity*, *data rate*, and *data volume*. An example of collecting data at different granularity levels is to gather performance data on a specific class of entity as opposed to every instance of an entity in a specific class. This proved to be the case during PerfMETRICS development; not only were the run-time overheads of tracking and buffering data for individual entities too great, but user requirements emphasized understanding the impact of certain entity types and their missions on overall scenario performance. This resulted in the implementation of a more course-grained view of entity-based performance data, specifically aggregated by entity class (e.g., air, ground, water).

Data rate impacts, in some cases, can be significant. Instrumented models that exhibit low-processing, high-update rates can impose greater monitoring overheads than a model that is more compute-bound and whose update rate is much less frequent. This was observed during the DARPA STOW development effort. Simulations modeling fewer numbers of fixed-wing aircraft (compute/memory bound model updates) exhibited lower monitoring costs (intrusiveness) than simulations modeling large numbers of tanks (simpler models but having more updates). The monitoring costs were greater due to the frequency each entity was updated (and the associated instrumentation code was executed). Data volume must also be considered because of the potential processing, memory, and bandwidth (communications) requirements.

The significance of the performance impact of these factors is highly dependent upon the value of the data as required for analysis. The intrusiveness of the instrumentation code is also dependent upon the performance of the underlying hardware so improved processor and memory performance may actually decrease the overall intrusiveness of the monitoring process for some instrumentation profiles. Intrusiveness is still primarily a function of data granularity, the frequency that instrumentation code is executed (data collection rate), and the amount of data that is collected (data volume).

Objective 4. Provide a baseline of practical experiences for future work related to performance measurement and monitoring for the design, configuration, and control of DIS and HLA simulations.

Regarding related efforts early on in this research, observations indicate there was not an emphasis on establishing an infrastructure for defining and monitoring higher-level (application) performance measures. Most efforts mainly focused on building a communications infrastructure to gather operating system-level metrics that don't necessarily describe the impact of modeling or scenario design decisions. More recent efforts within the DMSO organization are attempting to bring together a framework for estimating the performance requirements of an HLA Federation [71,72]. It does consider model components of the simulation environment (objects and their expected interactions) but is not intended for use beyond HLA and takes an approach that is benchmark-centric rather than emphasizing the run-time monitoring and data collection aspects of data analysis to support design, development, configuration, and control activities. This effort's focus is on the estimation of performance and resource planning.

The significant aspect of this thesis research is it provides a more generalized definition of performance for DIS and HLA-based simulation environments. The use cases provided practical experience in real world programs, established a baseline of performance measures useful for characterizing the run-time performance of a diverse set of modeling and simulation applications, and identified some of the limitations of the current performance monitoring implementations (including PerfMETRICS). The lessons learned are useful for extending or generalizing the framework, and enhancing or developing a new data collection methodology.

6.2 Evaluation Summary

The evaluation of research objectives has shown the utility and breadth of application of the performance framework and its effectiveness at providing meaningful and useful information to persons making decisions about the design, configuration, and control of

DIS and HLA simulation environments. The positive impact of using the framework (specifically the performance measures and PerfMETRICS) on real-world programs is evident as users were able to authoritatively (quantitatively) answer specific questions regarding model design, workload partitioning, and resource utilization (STOW ACTD and AVCATT-A). The taxonomy of performance measures appeared robust as witnessed in the use of PerfMETRICS in an "active" vice "passive" mode of operation, providing dynamic load scheduling of simulation workloads (BFTT AMN). The data collection methodology and implementation proved to be flexible in the presence of evolving requirements to monitor different kinds of applications (e.g., DIS⇔HLA Interface) and support assessments regarding the impact of changing technologies (DMT Experiment).

From its initial (and successful) use during the STOW ACTD, the defined performance measures and PerfMETRICS infrastructure have demonstrated the added value of making performance monitoring and data collection an integral component of any DIS and HLA-based architecture. It is entirely too difficult to anticipate run-time performance problems due to what is typically a complex set of objects and interactions and increasing demands on the scale (size) of the distributed simulation environments. The unified framework presented in this thesis provides the mechanism to support near-real time assessments regarding the impact of certain modeling and simulation design and configuration alternatives.

6.3 Practical Significance and Contribution

This section of conclusions can best be summarized by saying the framework presented in this thesis research is based on practical experiences gained while participating in the real-world programs presented as use cases in this research. The use cases are associated with performance of DIS or HLA-based (or both) simulations, however, this does not preclude the practical application of the techniques, tools, and analysis objectives to other distributed simulation environments. Recurring objectives appeared in each use case based on the need to understand the performance impact of specific scenario workloads (objects, object interactions, and synthetic environments). The recurring analysis objectives and the ability to meet those objectives for each of the use cases provide a credible basis for assessing the framework. The taxonomy seems robust in its coverage of relevant performance measures for each use case. This fact highlights a significant contribution of this research, namely it provides a framework to help decision-makers build a performance monitoring and analysis infrastructure into the overall design and configuration of a simulation environment. Not considering the simulation-centric performance measures, the framework in general is reusable (as is the PerfMETRICS implementation) and as such could be considered useful in any distributed computing environment where understanding run-time complexity and the problems it creates is desirable.

The fact that the use cases occurred over a period of four years, and the fact that PerfMETRICS was selected as integral part of the system development and analysis within these programs is a testament to the reasonableness of the system design and it adequacy to meet changing analysis objectives by providing a flexible and extensible implementation. Over the life of the current PerfMETRICS implementation, it has been interesting to see other monitoring systems within the same community end up being re-architected so as to be similar in design to PerfMETRICS. This is in large part due to the simple and clean approach to implementing a daemon-based architecture with simple, minimally-intrusive, and well-defined interfaces with the application being monitored.

6.4 Future Research

Coinciding with the successful achievement of thesis objectives comes the realization that opportunities exist to extend this research and its application to DIS and HLA-based simulation environments, as well as other distributed computing environments. Future research and development activities include: 1) enhancements or extensions to the existing PerfMETRICS implementation, 2) improvements to the utility of the proposed framework, and 3) continuing to actively employ the use of the framework and monitoring infrastructure in existing or future programs. Useful modifications to PerfMETRICS include:

- **Variable-length data packets**; PerfMETRICS uses a statically defined protocol packet for sharing performance data. Currently, this packet specification must be changed on a "per application" basis depending on the kind of performance data desired. This also requires modifications to the collection code that must log or display the relevant performance information. Although this will add processing overheads to the collection daemon packet processing routine, the benefits in terms of system flexibility/extensibility and packet bandwidth reduction are expected to outweigh the processing costs.

- **Dynamic selection of monitored, collected, and displayed performance measures**; Data collection and presentation requirements may change in between or during the execution of a distributed simulation exercise depending on the type of analysis activity. The ability to specify and filter information that is collected and displayed is desirable. This makes data analysis and interpretation easier as well as reducing bandwidth requirements related to PerfMETRICS monitoring and collection processing, network transmission, and data logging.

- **More extensive run-time visualization/graphics**; PerfMETRICS currently provides the capability to concurrently display performance data in a tabular format for multiple-applications. An earlier SGI-based version of the PerfMETRICS GUI provided a time-series plot capability allowing a person to see the correlation between selected performance measures as a function of time. This capability needs to be re-implemented. Additional graphical displays could include 3-dimensional presentations to support more complex, multi-variate analysis methods.

- **Dynamic load modules**; Incorporating this functionality into PerfMETRICS supports run-time monitoring of DIS, HLA, or both protocol environments; the objective being to structure the implementation to be more of a "composable" system. An example is to load a Federation Management module to allow the PerfMETRICS collection daemon to collect federation Management Object Module (MOM) data. Implementing dynamic load modules also supports the integration of relevant data for monitoring different application types reporting different performance metrics. An added benefit is the possibility of smaller PerfMETRICS binaries and associated working sets since the related processes would only have memory requirements for the code that is actually used for a specific monitoring task.

- **Automated and less intrusive instrumentation techniques**; Its not clear that hand-instrumented application code is not necessary for analysis activity involving the use of PerfMETRICS. However, it is possible to implement automatic code generation for some instrumentation "hooks". Possibilities include count and timing data associated with the DIS and RTI communications infrastructure or other shared/reused software components that have well-defined interfaces. Modifications to PerfMETRICS to further reduce the intrusiveness of the measurements process include utilizing system time information stored in a shared memory block (maintained by xntp) and better use of inline code substitution techniques (macros) to reduce overheads of frequently called data collection functions.

- **Integrated pre-processing data analysis module**; The present implementation of PerfMETRICS requires performance data saved to a log file to be pre-processed with a separate application, before being imported and used in a post-exercise analysis mode. It would be better if this pre-processing function was tightly-coupled with the PerfMETRICS GUI to provide a more user friendly interface to select the data desired for analysis. Additionally, analysis tools could be instantiated using the PerfMETRICS GUI to provide a more cohesive analysis environment for the end users. This level of automation would also make the end-to-end monitoring, collection, and analysis process more suitable for real- and near real-time decision support functions during the conduct of a simulation exercise.

- **Reuse assessments of other monitoring and analysis components**; the objective being to increase the overall functionality of the PerfMETRICS monitoring, collection, and analysis environment. Any functional enhancements to the existing PerfMETRICS implementation in terms of code generation should be weighed against alternatives such as the reuse of other open source applications. Additional activity should include the migration (porting) of the PerfMETRICS implementation for use in a Win32 (Windows and NT) environment. It would be especially useful to be able to readily connect a Window-based laptop computer into any DIS / HLA environment configured to use PerfMETRICS.

- **Dynamic multi-level, multi-resolution modeling**; PerfMETRICS can be modified to supply relevant performance data and control information to support the dynamic instantiation of model representations at differing levels of fidelity. An example is to implement run-time switching between taskframe-based and TacAir-Soar-based behavioral representations for a specific entity. PerfMETRICS control packets could also be used to selectively "turn on" or "turn-off" different levels of detail in specific model representations. Included in these enhancements is the implementation of different load-scheduling algorithms based on different factors effecting run-time performance and the feedback mechanism already implemented in PerfMETRICS.

Improvements to the performance analysis framework include:

- **Specification of aggregate / global performance measures**; a more robust set of performance metrics could be identified to provide an aggregated view of simulation and system performance across the entire distributed simulation (WAN/LAN) environment. In general, a richer set of performance measures within the framework would be useful, the objective being to relate application and model-level complexity to run-time performance.

- **Specific techniques applicable to performance analysis**; Given the specification of additional performance measures, second-order analysis techniques could be applied, such as sensitivity analysis characterizing the global distributed simulation environment given some increase in the scenario workload. Additional methodologies and mechanisms could be used to perturb the system and understand the global impact (stability, reliability, validity) of variations / fluctuations in some subset of distributed simulation applications.

- **Complexity/scalability classifications**; It would be useful to be able to provide reasonable estimates of expected performance based on the run-time interactions and compute resources used by any given scenario. This capability would be useful, for example, to provide "look-ahead" and allocate resources based on scenario dynamics.

- **Automated, traceable mapping function between the framework, the relevant performance measures, and the analysis objectives**; Right now, the process by which relevant performance metrics are identified and used to support data analysis is based on the experience level of the analyst and his ability to explicitly define the data requirements for any given analysis function. It would be very useful too provide the capability to automatically identify relevant performance measures (from the frameworks taxonomy) and relate those to specific kinds of DIS and HLA performance analysis objectives, for which they are suitable. This capability might be a good candidate for some kind of intelligent agents or other cognitive model.

- **A component of the framework to include the integration of live C4I systems;** Large-scale integration of real-world systems (e.g., C4I) with constructive simulation environments is and artifact of successes using DIS and HLA-based simulations and more stringent requirements for the use of modeling and simulation to augment the different DoD training, engineering, and analysis domains. C4I systems as used by the military have many unique communications and timing requirements. The semantic information associated with a C4I interface can significantly impact behavioral models inside the synthetic battlespace. Enhancements to the framework and its representation should account for the behavioral effects of C4I interfaces as well as the indirect impact on physical representations controlled by those behaviors.

- **Composable taxonomy of performance measures based on distributed application type and analysis requirements**; this desirable enhancement is based on the contention that the framework is extendible to non-DIS / HLA domains. The benefits would be manifested in a tool that supported the identification of performance measures and the automatic configuration of PerfMETRICS (composability) to monitor and collect performance data.

This section (and dissertation) concludes with a discussion related to opportunities for utilizing and extending this thesis research to support existing and future programs, as well as other non-DIS and HLA distributed computing environments. As previously stated, although some performance issues may be unique to distributed simulations, many are applicable to distributed computing applications in general and the expectation is that much of what has been presented here will be useful for performance analysis across a broader spectrum of application domains (as opposed to distributed simulation). The notion of performance bottlenecks associated with layered communications protocols, data translations, task scheduling, fault tolerance, process migration, object management, and other architectural technologies and mechanisms provides a natural overlay of other applications domains to the ones presented in this thesis.

Despite the maturation of distributed simulation architectures, most systems development efforts invariably appear to either make run-time performance an after thought in terms of system architecture or alternatively pursue the "holy grail" of performance analysis in the context of prediction. Additionally, programs frequently decouple performance analysis from configuration and control activities in terms of the overall system architecture, which necessarily seems to demarcate an important group of persons (configuration planners and exercise support personnel) from information they require to make effective decisions. This can result in redundant information flows when another mechanism is implemented to get the required data. PerfMETRICS provides a practical approach to run-time performance analysis. The framework and monitoring infrastructure represent reasonable tradeoffs in terms of valuable information, monitoring intrusiveness, and implementation complexity.

PerfMETRICS and the performance analysis framework embodied in its implementation are currently being considered for use in additional DoD programs. The framework represents the notion of a unified and integrated model of performance characterizations across most simulation life-cycle activities, most significantly the design, development, configuration, and control of distributed simulation environments. The following near-term DoD programs have been identified as potential candidates to benefit from the results of this thesis:

- **Warfighting Concepts to Future Weapon System Design (WARCON)**; A program with objectives to develop an integrated acquisition environment that couples examination of warfighting concepts and weapon system design. The system architecture includes the integration of engineering and operational simulations. The interoperability of disparate simulation architectures has an array of modeling and simulation issues that impact run-time performance.

- AVCATT-A; Preliminary work to support this program's contract proposal was presented in the use cases. This program has now started the design and implementation phases and as such has a need to understand the run-time performance impact of any design decisions. A requisite tool needs to provide low-level model analysis to understand the details of sub-model behaviors, the objective being able to assess overall model performance based on the run-time interactions of model and sub-model components.

- **NWDC / MBC**; has identified a desire to investigate the thesis framework and PerfMETRICS infrastructure to support run-time performance analysis in Maritime Battle Center M&S activities (e.g., Fleet Battle Experiments). Thesis results will be useful to establish system-level requirements for a unified approach to understand constructive simulation performance as well as understand the performance impact of integration of M&S components and live C4I systems.

REFERENCES

[1] G. Coulouris, J. Dollimore, T. Kindberg, *Distributed Systems, Concepts and Designs*. Addison Wesley, 1994.

[2] S. Mullender. *Distributed Systems*. ACM Press, 1993.

[3] T.L. Casavant, M. Singhal, *Readings in Distributed Computing Systems*. IEEE Computer Society Press, 1994.

[4] C.M. Krishna, K.G. Shin, *Real-Time Systems*. McGraw-Hill, 1997.

[5] S.H. Son, *Advances in Real-Time Systems*. Prentice Hall, 1995.

[6] K. Hwang, *Advanced Computer Architecture – Parallelism, Scalability, Programmability*. McGraw-Hill, 1993.

[7] M.L. Simmons, *Debugging and Performance Tuning for Parallel Computing Systems*. IEEE Computer Society Press, 1996.

[8] U.S. DoD Modeling and Simulation (M&S) Master Plan, October, 1995. http://www.dmso.mil/docslib/mspolicy/msmp/1095msmp/.

[9] W.P. Hughes, *Military Modeling for Decision Making*. Military Operations Research Society, 1997.

[10] A.R. Pope, D.C. Miller, "The SIMNET Communications Protocol For Distributed Simulation," *Proc. of the Sixth Annual Technology In Training and Education (TITE) Conference*, March 1988.

[11] D.C. Miller, A.R. Pope, R.M. Waters, "Long-Haul Networking of Simulators," *Proc. of the Tenth Interservice/Industry Training Systems Conference*, December 1989.

[12] D.C. Miller, "The SIMNET Architecture For Distributed Interactive Simulation," *Proc. of the Summer Computer Simulation Conference*, July 1991.

[13] E.P. Harvey, P.A. Bonanni, "Simulator Network Brief," *Presented to Fighter Wing One at NAS Oceana, Virginia*, November 1989.

[14] A.R. Pope, "The SIMNET Network and Protocols," BBN Report Number 7102. BBN Laboratories, July 1989.

[15] "IEEE Standard for Distributed Interactive Simulation – Application Protocols," *IEEE Standards Board*, Technical Report IEEE-Std-1278.1-1995, New York, 1995.

[16] "IEEE Standard for Distributed Interactive Simulation – Communication Services and Profiles," *IEEE Standards Board*, Technical Report IEEE-Std.1278.2-1995, New York, 1995.

[17] "Rationale Document: Entity Information and Entity Interaction in a Distributed Interactive Simulation," *Institute for Simulation and Training*, IST Report Number IST-PD-91-1. University of Central Florida, January 1992.

[18] M. Tambe, W.L. Johnson, R.M. Jones, F. Koss, J.E. Laird, P.S. Rosenbloom, K. Schwamb, "Intelligent Agents for Interactive Simulation Environments," *AI Magazine*, vol. 16, no. 1, pp. 15-39, 1995.

[19] J.E. Laird, K.J. Coulter, R.M. Jones, P.G. Kenny, F. Koss, P.E. Nielsen, "Intelligent Computer Generated Forces in Distributed Simulations: TacAir-Soar in STOW-97," *Proc. 1999 Spring Simulation Interoperability Workshop*, Paper No. 98S-SIW-212, March 1998.

[20] Aggregate Level Simulation Protocol Home Page. http://alsp.ie.org/alsp.

[21] "Aggregate Level Simulation Protocol (ALSP) Program Status and History," Technical Report MTR-93W0000079, The Mitre Corporation, March 1993.

[22] A.L. Wilson, R.M. Weatherly, "The Aggregate Level Simulation Protocol: An Evolving System," *Proc. 1994 Winter Simulation Conference*, December 1994.

[23] M.C. Fischer, "Aggregate Level Simulation Protocol (ALSP), Future Training with Distributed Interactive Simulations," *Proc. 1995 International Training Equipment Conference*, April 1995.

[24] J.S. Dahmann, R.M. Fujimoto, R.M. Weatherly, "The Department of Defense High Level Architecture," *Proc. 1997 Winter Simulation Conference*, December 1997.

[25] Defense Modeling and Simulation Office, http://www.dmso.mil.

[26] S.G. Purdy, R.D. Wuerfel, "A Comparison of HLA and DIS Real-Time Performance," *Proc. 1998 Spring Simulation Interoperability Workshop*, Paper No. 98S-SIW-042, March 1998.

[27] S. McGarry, "An Analysis of RTI-s Performance in the STOW 97 ACTD," *Proc. 1998 Spring Simulation Interoperability Workshop*, Paper No. 98S-SIW-229, March 1998.

[28] D.B. Cavitt, J. Bell, M. Checchio, C.M. Overstreet, K.J. Maly, "Performance Monitoring for the Design, Configuration, and Control of DIS/HLA Exercises," *Proc. 1998 Spring Simulation Interoperability Workshop*, Paper No. 98S-SIW-066, March 1998.

[29] L.D. Budge, R.A. Strini, R.W. Dehncke, J.A. Hunt, "Synthetic Theater of War (STOW) Overview," *Proc. 1998 Spring Simulation Interoperability Workshop*, Paper No. 98S-SIW-086, March 1998.

[30] G.E. Lukes, G. Goodman, "Synthetic Environments and Lessons Learned from the STOW-97 ACTD," *Proc. 1998 Spring Simulation Interoperability Workshop*, Paper No. 98S-SIW-097, March 1998.

[31] G.E. Lukes, P.A. Birkel, "Synthetic Environments – Final Technical Report," *Prepared for the Defense Advanced Research Projects Agency*, Document Control Number 98-5-3100, 1998.

[32] R. Cole, B. Root, L. O'Ferral, J. Tarr, "STOW Network Technologies and Operational Lessons Learned," *Proc. 1998 Spring Simulation Interoperability Workshop*, Paper No. 98S-SIW-103, March 1998.

[33] J. Tsai, S. Yang, *Monitoring and Debugging of Distributed Real-Time Systems*, IEEE Computer Society Press, 1996.

[34] B. Plattner, "Real-Time Execution Monitoring," *IEEE Trans. Software Eng.*, Vol. SE-10, No. 6, pp. 756-764, Nov. 1984.

[35] D.C. Marinescu, J.E. Lumpp, T.L. Casavant, H.J. Siegel, "Models for Monitoring and Debugging Tools for Parallel and Distributed Software", *J. Parallel and Distributed Computing*, Vol. 9, pp. 171-183, June 1990.

[36] S.E. Chodrow, F. Jahanian, M. Donner, "Run-Time Monitoring of Real-Time Systems", *Proc. Real-Time Systems Symp.*, pp. 74-83, IEEE Computer Society Press, 1991.

[37] J. Joyce, G. Lomow, K. Slind, B. Unger, "Monitoring Distributed Systems", *ACM Trans. Computer Systems*, Vol. 5, No. 2, pp. 121-150, May 1987.

[38] P.S. Dodd, C.V. Ravishankar, "Monitoring and Debugging Distributed Real-Time Programs", *Software-Practice and Experience*, Vol. 22, No. 10, pp.863-877, Oct. 1992.

[39] B.P. Miller, C. Macrander, S. Sechrest, "A Distributed Programs Monitor for Berkeley UNIX", *Software-Practice and Experience*, Vol. 16, no. 2, pp. 183-200, Feb. 1986.

[40] H. Tokuda, M. Kotera, C.W. Mercer, "A Real-Time Monitor for a Distributed Real-Time Operating System", *Proc. ACM Workshop Parallel and Distributed Debugging*, ACM Press, 1988.

[41] C.M. Pancake, M.L. Simmons, J.C. Yan, "Performance Evaluation Tools for Parallel and Distributed Systems," *Computer*, pp. 17-19, November 1995.

[42] M.T. Heath, A.D. Malony, D.T. Rover, "The Visual Display of Parallel Performance Data", *Computer*, pp. 21 – 28, November 1995.

[43] S. Fickas, M.S. Feather, "Requirements Monitoring in Dynamic Environments," *Proc. of the Second IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, March 1995.

[44] H. Jakela, "Performance Visualization of a Distributed System: A Case Study," *Computer*, pp. 30 - 36. November 1995.

[45] B.P. Miller et.al., "The Paradyn Parallel Performance Measurement Tool," *Computer*, pp. 37 - 46, November 1995.

[46] P.M. Dickens, P. Heidelberger, D.M. Nicol, "Timing Simulation of Paragon Codes Using Workstation Clusters," *Proc. of the 1994 Winter Simulation Conference*, pp. 1347-1353, December 1994.

[47] P.M. Dickens, P.Heidelberger, D.M. Nicol, "Parallelized Direct Execution Simulation of Message-Passing Parallel Programs," Technical Report 94-50, *ICASE*, July 1994.

[48] S.R. Sarukkai, P. Mehra, "Automated Scalability Analysis of Message-Passing Parallel Programs", *IEEE Parallel & Distributed Technology*, pp. 21-32, Winter 1995.

[49] R.M. Fujimoto, "Parallel Discrete Event Simulation," *Communications of the ACM*, Vol. 33, No. 10, pp. 31-53, October 1990.

[50] R.M. Fujimoto, "Performance of Time Warp under synthetic workloads," *Proc. of the SCS Multiconference On Distributed Simulation*, Vol. 22, pp 23-28, January 1990.

[51] B. Falsafi, D.A. Wood, "Cost/Performance of a Parallel Computer Simulator," *Proc. of the 8th Workshop on Parallel and Distributed Simulation*, pp. 173-181, July 1994.

[52] C.D. Carothers, B. Topol, R.M. Fujimoto, J.T. Stasko, V. Sunderam, Technical Brief presented at 1997 Winter Simulation Conference, December 1997.

[53] R. Vrablik, W. Richardson, "Benchmarking and Optimization of ModSAF," *Modular Semi-Automated Forces: Recent and Historical Publications*, LADS Document Number 94007 v. 1.0, May 1994.

[54] E. White, "ModSAF 1.4 Reverse Engineering Report," *Applied Research Laboratories*, The University of Texas at Austin, April 1995.

[55] J.E. Smith, L.C. Schuette, K.L. Russo, D. Crepeau, "Rational Characterization Of The Performance Of Distributed Synthetic Forces," *Proc. of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pp. 665-673, March 1996.

[56] R.K. Guha, M.A. Bassiouni, "A Framework for Modeling High Level Architecture (HLA) Using Petri Nets," *Proc. of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pp. 515-521, March 1996.

[57] S. Srinivasan, R.F. Reynolds, "Performance Modeling for the High Level Architecture," *Proc. of the 15th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pp. 951-959, March 1996.

[58] S. Srinivasan, R.F. Reynolds, "Communications, Data Distribution and Other Goodies in the HLA Performance Model," *Proc. 1997 Spring Simulation Interoperability Workshop*. Paper Number 97S-SIW-050, April 1997.

[59] R.R. Nair, D.N. McGregor, B.J. Root, "Data Collection Using SNMP In The DIS Environment," *Proc. of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pp. 361-371, March 1996.

[60] W.P. Sudnikovich, A. Huynh, J. Pasirstein, R. Wood, W. Walker, "A Proposal For Simulation Performance PDUS," *Proc. of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pp. 951-959, March 1996.

[61] J.F. Stender, "Simulation Management for Large Exercises", *Proc. of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pp. 893-896, March 1996..

[62] "IEEE Recommended Practice for Distributed Interactive Simulation - Exercise Management and Feedback (Working Draft Proposal)," *IEEE Standards Board*, IEEE-Std-1278.3-1995, Washington, D.C., 1995.

[63] B. Butler, "Changes To The Standards Document: Recommended Practice For Distributed Interactive Simulation - Exercise Management And Feedback," *Proc. of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pp. 523-534, March 1996.

[64] J. Swauger, "Desired Capabilities of DIS Exercise Support And Feedback Tools," *Proc. of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pp. 639-645, March 1996.

[65] T.J. Mowbray, R. Zahavi, *The Essential CORBA, Systems Integration Using Distributed Objects*, John Wiley & Sons, Inc., New York, 1995.

[66] K. Hunt, J.S. Dahmann, R. Lutz, J. Sheehan, "Planning For The Evolution of Automated Tools In HLA," *Proc. 1997 Spring Simulation Interoperability Workshop*, Paper Number 97S-SIW-067, April 1997.

[67] S.G. Purdy, R.D. Wuerfel, "A Comparison Of HLA And DIS Real-Time Performance," *Proc. 1998 Spring Simulation Interoperability Workshop*, Paper Number 98S-SIW-042, April 1998.

[68] R.D. Wuerfel, R. Johnston, "Real-Time Performance Of RTI Version 1.3," *Proc. 1998 Fall Simulation Interoperability Workshop*, Paper Number 98F-SIW-125, Sept. 1998.

[69] R.D. Wuerful, J.S. Olszewski, "Defining RTI Performance," *Proc. Spring Simulation Interoperability Workshop*. Paper Number 99S-SIW-100, April 1999.

[70] R.D. Wuerful, J.S. Olszewski, "An RTI Performance Testing Framework," *Proc. Spring Simulation Interoperability Workshop*, Paper Number 99F-SIW-127, Sept. 1999.

[71] R. Richardson, J.S. Dahmann, R. Weatherly, R. Briggs, "High Level Architecture Performance Framework," *ITEC presentation brief*, April 1998.

[72] "Federation Execution Planner's Workbook, Version 1.3 User's Guide," Defense Modeling and Simulation Agency. 1999.

[73] P.H. Winston, *Artificial Intelligence*, Addison Wesley, 1992.

[74] "Standard for Integration Definition for Function Modeling (IDEF0)," *Nat'l Bureau of Standards*, December 1993.

[75] A. Ceranowicz, "STOW 97-99," Presented at the ALL-STOW Conference. June 1998.

[76] J.O. Calvin, C. J. Chiang, S.M. McGarry, S.J.Rak, and D.J. Van Hook, "Design, Implementation, and Performance of the STOW RTI Prototype (RTI-s)," *Proc. 1997 Spring Simulation Interoperability Workshop*. Paper Number 97S-SIW-019, April 1997.

[77] B.A. Shirazi, A.R. Hurson, K.M. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos, 1995.

APPENDIX A

PERFMETRICS GUI USER'S GUIDE AND DATA DICTIONARY

Introduction

This user's guide provides an overview of the PerfMETRICS monitoring system and how to use it to monitor DIS/HLA-based simulations. PerfMETRICS detects, collects, and displays high-level performance metrics that describe behavior of the physical and logical resources and services used in the design and implementation of DIS and High Level Architecture (HLA) simulations. Run-time and post-exercise feedback of performance information can provide meaningful information as a guide in making decisions about the configuration and control of the available hardware and software resources; the goal is to provide information needed by exercise planners and managers. This performance information can also be used to support modeling and simulation requirements and design.

Quick Startup Procedures

1) Start all simulations required for performance monitoring. Note that the simulations must be run either as root or as the same user (uid or effective uid) as the PerfMETRICS collection daemon running on that workstation (safuser in the WISSARD lab).

2) Login as safuser on the workstation designated as the PerfMETRICS monitoring station (perfmon in the WISSARD lab).

3) Select the PerfMETRICS button on the icon bar of the window manager.

4) View the desired simulation engine by pressing the right hand mouse button and selecting the appropriate workstation.

Software and Hardware Requirements

The current PerfMETRICS implementation collects and reports performance information from hand-instrumented, ModSAF-based simulations. The system has only been tested on Silicon Graphics (SGI) and Linux-based Workstations. Note that the system level information is not currently available on SGI Workstations. To compile the PerfMETRICS instrumentation code, the PERFMETRICS compiler directive must be used during the build process and the functions provided in the libodumetrics software library must be linked into the simulation executable.

The PerfMETRICS collection daemon is written in ANSI C. The native SGI and Linux C-compilers are suitable for compiling the collection daemon. The simulation engines and the monitoring control station must be configured for UNIX System V IPC and Multicast/IP networking. As mentioned in the Quick Start section, the simulation process and the collection daemon process must have either the same uid or effective uid to properly communicate.

The PerfMETRICS Graphical User Interface (GUI) is a Motif-based application that currently requires the SGI Viewkit libraries. As such the GUI must be run on an SGI Workstation with Viewkit support. It may be displayed remotely on non-SGI systems by setting the appropriate display environment variable (DISPLAY).

Using The PerfMETRICS GUI

1. Starting the PerfMETRICS GUI. See quick start procedure

2.The User Interface Use the right mouse button to select a specific machine for viewing.

Admin Button

New View to create another window; this is useful when trying to observe or compare performance data from two or more simulations.

Data Logging to start and stop saving the data to a file.

Exit to quit PerfMETRICS; ALWAYS USE THIS BUTTON to quit PerfMETRICS instead of killing from the window manager.

Config Button: Unused at this point

Views Button: Toggles between global, local, and monitoring views; local view is currently the only view that display data.

Plot Button: Opens up a plotting window to display X-Y plots of selected performance variables. The most useful at this time is comparing "Slack in Tick Rate" and "Entity Update Performance". To do this:

- Open the Selection Button.
- Select Preferences.
- Check the box beside these variables.
- Select Apply, then Dismiss.

3. Interpreting the Data : See Data Dictionary table; Adjust the window panes to view appropriate entity, simulation, and system level performance data.

GUI Data Dictionary

The following table describes the tabular data presentation of the PerfMETRICS GUI. Note that PerfMETRICS monitors and collects more data than actually displayed on the GUI.

Item	Description	Units
ENTITY INFORMATION		
ENTITY COUNTS		
Locals	The number of local vehicles (being simulated on a workstation)	Integer count
Remotes	The number of remote vehicles (whose entity state is seen by a workstation)	Integer count
Other	The number of all other local and remote entity types (e.g., aggregates, radio, etc.)	Integer count
PHASE PROCESSING TIMING		
PDU In	The relative time spent processing incoming PDUs relevant for entity state data: over the past 10 seconds (% interval), relative to the total time spent ticking entities (% total tick), and relative to the entire simulation time (% total sim).	Percent (%)
PDU Out	The relative time spent processing outgoing PDUs relevant for entity state data: over the past 10 seconds (% interval), relative to the total time spent ticking entities (% total tick), and relative to the entire simulation time (% total sim).	Percent (%)
Hull	The relative time spent processing entity kinimatics (e.g., flight dynamics model): over the past 10 seconds (% interval), relative to the total time spent ticking entities (% total tick), and relative to the entire simulation time (% total sim).	Percent (%)
Turret	The relative time spent processing an articulated part (turret): over the past 10 seconds (% interval), relative to the total time spent ticking entities (% total tick), and relative to the entire simulation time (% total sim). Note this is only relevant for tanks and other ground vehicles containing turrets.	Percent (%)
Gun	The relative time spent processing entity weapon systems: over the past 10 seconds (% interval), relative to the total time spent ticking entities (% total tick), and relative to the entire simulation time (% total sim).	Percent (%)
Sensor	The relative time spent processing entity sensor systems: over the past 10 seconds (% interval), relative to the total time spent ticking entities (% total tick), and relative to the entire simulation time (% total sim).	Percent (%)
Tasking	The relative time spent processing entity behaviors: over the past 10 seconds (% interval), relative to the total time spent ticking entities (% total tick), and relative to the entire simulation time (% total sim). Note that for SOAR agents this is the behavioral processing times and for SAFOR this is the time spent executing task (taskframes).	Percent (%)
GUI	The relative time spent processing entity graphical display logic: over the past 10 seconds (% interval), relative to the total time spent ticking entities (% total tick), and relative to the entire simulation time (% total sim).	Percent (%)

SIMULATION INFORMATION		
Slack In Tick Rate	The time within a 500 millisecond window that is available after processing all entities.	Milliseconds
Entity Update Performance	The relative number of entities meeting their prescribed update rate (2 Hz.) over some specified time period (60 second default in ModSAF)	Percent (%)
Update Rates	The current rate that entity state is being updated (ticked): for local vehicles (local veh.), for remote vehicles (remote veh.), and all other vehicles (other). Note that this corresponds with what is referred to as the "SAF frame rate".	Cycles / Sec. (Hz)
Entity Tick Time	The time spent updating (ticking) entity state relative to the total simulation time: over the past 10 seconds (% interval), and since simulation startup (% total sim).	Percent (%)
Idle Scheduler	The scheduler's idle time (spin time in the case of ModSAF) relative to total simulation time: over the past 10 seconds (% interval), and since simulation startup (% total sim).	Percent (%)
RTI Tick Time	The time spent in the RTI (ticking) relative to total simulation time: over the past 10 seconds (% interval), and since simulation startup (% total sim).	Percent (%)
Real Time	Wall clock time since simulation startup.	Seconds
User Time	Time spent executing simulation code.	Seconds
System Time	Time spent in operating system code on behalf of the simulation process.	Seconds
Mean Soar Decision Time	Average time for a single Soar decision cycle (mean value over a ten-second interval).	Milliseconds
Mean RTI Processing Time	Average time for a single RTI tck (mean value over a 10 second interval	Milliseconds
SYSTEM INFORMATION		
CPU	Processor type: Operating System	
User	Relative time spent executing user processes since system boot up.	Percent (%)
System (Kernel)	Relative time spent executing operating system code since system boot up.	Percent (%)
Idle	Relative time spent idle since system boot up.	Percent (%)
MEMORY		
Total Physical	Total amount of memory configured on workstation (simulation engine).	Kilobytes (kB)
Used	Total amount of memory allocated by operating system to processes (including the simulation).	Kilobytes (kB)
Free	Total amount of available memory	Kilobytes (kB)
Total Swap	Total amount of swap space configured on workstation (simulation engine).	Kilobytes (kB)
Used	Total amount of swap space used by the operating system	Kilobytes (kB)

Free	Total amount of free swap space	Kilobytes (kB)
Page In	The operating system page rate over the past 10 seconds (newly allocated pages). Note this is the operating system's activity on behalf of a specific process.	Pages / Second
Page Out	The operating system page rate over the past 10 seconds (pages that are freed by writing data out). Note this is the operating system's activity on behalf of a specific process.	Pages / Second
Swap In	The operating system swap rate bringing processes in for execution. Note this is the operating system's activity in order to provide fair CPU time to all processes.	Pages / Second
Swap Out	The operating system swap rate taking processes out of exeuction. Note this is the operating system's activity in order to provide fair CPU time to all processes.	Pages / Second
NETWORK		
Packets Received	IP packet receive rate for the last 10 seconds	Packets / Second
Receive Errors	IP packet receive errors for the last 10 seconds	Packets / Second
Packets Sent	IP packet send rate for the last 10 seconds	Packets / Second
Send Errors	IP Packet send errors for the last 10 seconds	Packets / Second

APPENDIX B

FUTURE WORK – RESEARCH AND DEVELOPMENT ALTERNATIVES

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Variable-length data packets		X		X	X	X
Dynamic selection of monitored, collected, and displayed performance measures		X		X	X	X
More extensive run-time visualization / graphics			X	X	X	X
Dynamic load modules	X			X	X	X
Automated and less intrusive instrumentation techniques	X			X	X	X
Integrated pre-processing data analysis module		X		X		X
Reuse assessments of other monitoring and analysis components		X				
Dynamic multi-level, multi-resolution modeling support			X	X		X
Specification of aggregate/global performance measures		X		X		X
Specific techniques applicable to performance analysis			X	X		X
Complexity classifications		X		X	X	X
Automated, traceable mapping function between the framework, the relevant performance measures, and the analysis objectives			X			
A component of the framework to include the integration of live C4I systems		X		X	X	X
Composable taxonomy of performance measures based on distributed application type and analysis requirement		X				

VITA

David B. Cavitt was born on January 28, 1959 at Patuxent, Maryland. He grew up as a Navy dependent, but slipped up on the opportunity to pursue a career in Naval Aviation, as his father and older brother had before him. As it turns out a better plan included taking the hand of his lovely wife, Ellen; children followed and then higher education called him away from his career as a boat builder.

Mr. Cavitt tended to his undergraduate studies in Computer Science at Old Dominion University between 1985 and 1989, at which time he was awarded his Bachelors in Science Degree. He started his career in the U.S. defense industry in 1987, working as a software engineer developing real-time command and control software. He then had a short tenure at NASA Langley Research Center in Hampton, Virginia before returning to Old Dominion University to pursue graduate studies. Simulation has been an integral part of his entire career track. The thesis research presented in this dissertation is submitted to the faculty of the Computer Science Department, Old Dominion University, in Norfolk, Virginia in fulfillment of the Mr. Cavitt's Ph.D. requirements.

Mr. Cavitt is currently a research engineer with BMH Associates, Inc., Norfolk, Virginia. He has 13 years of experience in the use and development of simulations for military and engineering applications. His research interests include modeling and simulation, performance analysis, and distributed systems. Mr. Cavitt is a member of ACM and IEEE CS.