2006

Design factors for the communication architecture of distributed discrete event simulation systems

Catharine McIntire Hoaglund

DESIGN FACTORS FOR THE COMMUNICATION ARCHITECTURE OF DISTRIBUTED DISCRETE EVENT SIMULATION SYSTEMS

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Catharine McIntire Hoaglund

September 2006

DESIGN FACTORS FOR THE COMMUNICATION ARCHITECTURE OF DISTRIBUTED DISCRETE EVENT SIMULATION SYSTEMS

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Catharine McIntire Hoaglund

September 2006

Approved by:

[redacted]

Ernesto Gomez, Chair, Computer Science — August 7, 2006
Date

[redacted]

Yasha Karant

[redacted]

Kerstin Voigt

ABSTRACT

This research used the Assessment of Simulation Distribution Factors (ASDF) system to compare the effect on simulation performance of three communication configurations representing common situations a simulation developer might face. In an experiment, a commercial simulation engine executed a simple discrete event simulation which was capable of responding to external events through an external interface. An independent process generated time-tagged event messages at a specified rate, and delivered them in one of three ways to the third component, the intermediary, which presented the information to the simulation as events.

The relative effects of simulation factors were compared with factors based on the communication path of external data interfaces. Performance measurements were gathered and analyzed. A factor-by-factor review was done, which concluded that patterns were visible in the data for both primary and secondary factors. The results also indicated that the secondary factors attributable to the communication mode were, in fact, important enough to overall success to be worth optimization. The relative effects of the various factors were assessed.

ACKNOWLEDGMENTS

The support of Dr. Ernesto Gomez, the Institute for Applied Supercomputing, California State University, San Bernardino, and of Metron Incorporated, Solana Beach, California, is gratefully acknowledged.

DEDICATION

To Richard Hoaglund for unwavering support

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE:

BACKGROUND

1.1 Introduction

Chapter One presents an overview of the context, motivation and significance of the work documented in this thesis. The purpose of the thesis is discussed, followed by a discussion of the context of the problem, and the significance of the thesis. Some of the specialized concepts in the problem area are introduced by way of background, and a short glossary is provided. Next, the assumptions, hypotheses, and limitations that apply to the thesis are reviewed. Finally, the organization of the remainder of the document is presented.

1.2 Purpose of the Thesis

The purpose of the thesis was to investigate the influence communication architecture decisions have on the performance of a simulation system with distributed components. In particular, the objective was to assess the relative importance of factors affecting reliability and variability of an external data interface to the performance of the simulation, as compared to factors within the simulation itself.

To investigate these questions, the Assessment of Simulation Distribution Factors (ASDF) system was created, test runs were conducted, and the results were analyzed, as discussed in the following chapters.

1.3 Context of the Problem

The type of simulation addressed in this research is Distributed Discrete Event Simulation (DDES), using active time management. A critical feature of this class of program is that it is event-driven, with time as a logical value, advancing or rolling back to earlier values under the control of the simulation engine in response to events. The work is distributed to various logical processes, which progress independently, and which may or may not be hosted on physically separate computers.

Such systems are frequently combined and extended to model larger or more complex entities, and sometimes must exchange data with systems for which time is not such a flexible entity. For example a DDES may be required to accept and react to an incoming stream of real-time data points from a hardware-in-the-loop sensor. There are a number of factors which influence the success of a simulation based on such a combination.

1.4 Significance of the Thesis

In the realm of distributed simulation, one frequent challenge is to integrate independently developed models, simulation engines, or other components into a system that captures meaningful data about a real-world system of larger scope. If, for example, it is desired to simulate a proposed new system using existing subsystems A, B, and C, and validated models exist for all three, it seems sensible to compose the new simulation to use the existing programs. But if the underlying approach of Model C differs from the other two, or if Model B is optimized for input at a different rate than A and C, the three good models may combine to make a system with terrible performance, or worse, with biases in the aggregate data.

Such a situation can occur when the overall simulation is based on an event-based "Virtual Time" simulation advancement protocol, such as Time Warp [12], and input data comes from a component tied to wall clock time. The subsystem being integrated might have hardware-in-the-loop, or might be running at a fixed multiple of wall clock time. The key aspect is that the incoming frame-based data stream must be able to set the pace for the event-oriented main framework, while still allowing

the overall simulation to process asynchronous events at varying rates of occurrence.

This work looks at the relative importance of the choice of communication mechanism to the success of such an integration effort. While much research attention has been devoted to integrating large, independently developed simulations by means of the High Level Architecture (HLA) [9] or similar industrial strength solutions, less research is available to aid the developer who needs to bring in a local data source and faces choices as to where it should execute and how it should deliver its data.

Of the myriad factors that affect the success of such an enterprise, the most important are the time management style of the event-driven framework, and the update rate of the data source. It is frequently the case that the developer cannot feasibly change these first level factors, but must work with what is available. The elements affected by communication architecture choices are primarily the reliability of delivery, the latency of the data transmission, the variability of message delivery times, and the processing overhead. These are secondary factors, but are things that are often under developer control, and are the focus of this research.

1.5 Definition of Terms

The following terms are defined as they apply to the thesis.

Distributed Simulation - A simulation in which the computational work is divided among cooperating processes, which may or may not be executing on the same host. The simulation system may include specialized components such as sensors, timing devices, or data acquisition and monitoring subsystems. In this research the simulation engine and five "logical processes" are co-located on the same computer, with a distributed external data source.

Event - An event is an abstraction used to model some change in the state of the real-world system that is being simulated, or a change in the state of the simulation. In this thesis the events of interest are of two types, those internal to the simulation, and those that are representing external data being delivered by means of a message. Events are always time-stamped. [9, 19, 25]

Event rollback - A Discrete Event Simulation (DES) processes events internal to each logical process in time stamp order. When an event with an earlier time stamp is received by means of a communication from another component, the computation must be "rolled back" to return

the state of the receiving process to what it was when it processed the last previous time stamp. [9, 19, 25]

Global Virtual Time (GVT) - The logical processes each maintain a logical clock, but the simulation engine maintains an overall time value that represents the earliest time across all logical processes. Finding out what that value is, at a given point, requires a synchronization operation that is referred to as a "GVT event". [9, 19, 25]

Real-time frame - Many real-time systems operate in a periodic fashion, with a series of repeated actions occurring within a regular time window called a frame. If an action is scheduled to occur but does not complete before the next frame time it is said to "break frame" and any associated data is discarded. [16]

Time management - DES systems employ various algorithms to compute and advance GVT. One of the most important characteristics of these algorithms is the degree to which they permit computation to proceed beyond the last GVT, representing the risk that computation will have to be rolled back. This research uses three time management protocols. Time Warp (TW) is a high risk "optimistic" algorithm, Breathing Time Buckets (BTB) is a low risk "conservative" algorithm, and Breathing Time Warp

(BTW) is a medium-risk "balanced" algorithm. For this work the implementation of these algorithms is handled by the simulation engine in response to a line in a configuration file. [9, 12, 19, 24]

1.6 Assumptions and Hypotheses

The primary assumption made regarding this research is that modeling three configurations in a single system can yield worthwhile insight, that it will reveal properties of the systems being modeled that are not simply artifacts of variations in programming skill. Considerable effort was spent to make this assumption valid, by keeping the processing as close to the same between modes as possible.

The research began with the following two hypotheses: First: The effects of the primary factors of data rate, time management style and internal simulation complexity far overwhelm effects of the secondary factors relating to communication mechanism. Second: It is nonetheless possible to identify enough variation in the results due to the secondary factors to suggest where it is worthwhile to put effort. It is expected that it will be worthwhile to lessen variability of delivery time and improve delivery reliability of communication.

1.7 Limitations

During the development of this thesis, a number of limitations were noted. A primary difficulty has been that there are too many possible factors to address even a reasonable fraction at one time. Therefore the research is limited in that it does not address network loading, scalability, hardware supported synchronization, the effect of packet size, or the effect of distributing the simulation itself over multiple machines, all of which are topics of great interest.

1.8 Organization of the Thesis

This thesis is divided into five chapters. Chapter One provides an introduction to the context of the problem, the purpose and objectives of the thesis, significance of the research, the major limitations of the work, definitions of specialized terms, and this description of the organization of the document. Chapter Two consists of a review of relevant literature. Chapter Three documents the methodology used in this thesis, including requirements and goals, elements of the problem being modeled, a discussion of the design, the objectives of the data analysis plan, and notes on the actual implementation. Chapter Four presents the results from the

experiment, and goes into more detail about the analysis performed. Chapter Five discusses the validation of the thesis, and suggests several areas for subsequent research. The Appendices follow Chapter Five, and include Design, Data Analysis, Source Code, and References.

CHAPTER TWO:

LITERATURE REVIEW

2.1 Introduction

Chapter Two consists of a discussion of the relevant literature. Specifically, it covers literature on distributed simulation, research relating to the context of the study, and references involving specific products used.

2.2 Literature on Distributed Simulation

The landscape of distributed simulation is fissured by many divisions, with the researchers in each area pursuing different goals than their neighbors on the other side of whatever chasm divides them. Thus, the people working in analytical simulation may pursue distributed solutions primarily to achieve parallel speed-up, while those constructing a virtual training simulation may require geographical dispersion. There is no better map to this terrain than Richard Fujimoto's *Parallel and Distributed Simulation Systems,* [9], though there are many other references with valuable insights. [15, 27, 32]

To locate the area of interest for this work, the relevant attributes are: the simulation of interest is a

discrete event system, using active time management, distributed by logical process rather than parallel computation, communicating between processes via messages, and capable of incorporating real-time data from a hardware interface. The last attribute is a key one, as it requires a bridge between the kind of simulations that are allowed to go as fast as they can (in terms of simulation time) and those that must recognize the wall clock's authority.

There is quite a lot of research going on in this corner of the simulation world, but not much that bridges the gap in time philosophy on a modest scale. Ever since Jefferson's seminal paper, "Virtual Time", in 1985 [12], there has been a great deal of activity devoted to improving the Time Warp family of algorithms. Some interesting examples are papers by Fujimoto and Hybinette [8], Nichol and Liu [20], and Steinman [24, 25], but none deal with integration of externally generated data streams.

Bagrodia [1], Carothers [3], Fujimoto [9], and Zeigler [32], all provide useful guidance on the construction of distributed simulation in general, and all agree that the problem of efficiently managing inter-

process communication is key to successfully sharing the work.

Much of the published research is focused on very large distributed simulations using Distributed Interactive Simulation (DIS), or the High Level Architecture for simulation (HLA). [9] These protocols are intended to support up to 100,000 simulated objects which may be geographically separated by thousands of miles. An example of the network requirements for this scale of simulation is found in RFC 2502, *Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment,* issued by the Large-Scale Multicast Applications working group (LSMA).[22] They list requirements for security, multicast support and network management for such large-scale simulations, but the network service requirements for latency and jitter are also applicable to more modest endeavors: "real-time packet delivery, with low packet loss (less than 2%), predictable latency on the order of a few hundred milliseconds, after buffering to account for jitter (variation of latency) such that less than 2% of packets fail to arrive within the specified latency, in a shared network."

A number of papers have covered other aspects of combining simulations under the HLA for very large-scale simulations, an activity that inherently must integrate systems with differing time management approaches. Ferenci, Perumalla, and Fujimoto offer useful insight on problems when federating parallel simulators [7], and Pham and Bagrodia deal explicitly with time issues [21]. Zhang and Tropper discuss reducing the cost of processing causality violations that lead to "rollback explosions" that can halt processing. [33] Steinman [26], McGraw [17], Weiland [29], and Wright [31] all report on such efforts using the same simulation engine chosen for this study, SPEEDES. Despite much material of interest, the results for all of these studies were primarily of use to a developer with a very high degree of control and a large budget. The research listed was, however, useful in formulating the hypotheses for this work and in the choice of configuration variables.

2.3 Research on the Context of the Study

There are a number of other references which proved useful in understanding specific aspects of the type of system this experiment attempted to model. Compared to the high volume of literature on the theoretical basis for

Discrete Event Simulation, there are relatively few relevant papers on hardware-in-the-loop simulation, where simulation time is paced strictly by the need to deal with real-time data. An example of such an environment is the Integrated Multi-spectral Environment described by Whitted, Meidenbauer, and Louton [30]. The system they describe is required to synchronize multiple distributed simulations that drive signal generators to radiate in different parts of the spectrum. The electromagnetic environment created has to fool the sensors on a military aircraft, and so the per-frame timing budget is measured in tens of microseconds and requires special timing equipment to synchronize all components. Getting a hard real-time simulation like this to work with an optimistic Time Warp simulation is tricky, and yet this is increasingly being attempted. Liu [16] has good background on hard real-time systems, and RFP 2502, previously discussed, takes them into account. [22]

An example of this kind of convergence is described by Wright [31], who reports on tests conducted by the Joint Advanced Distributed Simulation (JADS) Task Force to assess the feasibility of linking installations such as the one discussed by Whitted into a federated simulation using HLA. This report concluded that such linking was of

varying utility, but worth pursuing. It added that latency, out-of-order messages, and data loss were significant issues, said "Variance in latency may be more problematic for some test designs", and noted "Data loss must be addressed in test design... the lowest latency computer communication protocol consistently showed the highest data loss."

Croak offers a number of valuable thoughts about ways to evaluate this kind of complex system, though he is talking about communication "middleware", not simulation interface mechanisms.[5] His discussion of using the dimensions of variability in a problem to identify throttling effects led to the idea that looking at secondary factors might be worthwhile.

Three papers on communication mechanisms proved helpful. Booth et al. looked at the way the Breathing Time Buckets and Time Warp algorithms used memory in a shared memory system, and found for both that memory access time degraded the actual parallel speedup, an effect accentuated as the number of processors increased [2]. Chandra, Larus, and Rogers performed a careful study of message passing versus shared memory and "found no clear performance advantage for message passing." [4] Kranz et al. looked at integrating message passing and shared

memory and itemized some areas where shared memory was less efficient than message passing, but concluded that there were problems best suited by the use of one or the other. [13] All three papers were inconclusive about which mechanism was better.

2.4 Literature on Specific Products

The SPEEDES system used in this experiment is primarily documented in *SPEEDES User's Guide, The Synchronous Parallel Environment for Emulation and Discrete-Event Simulation* [19], but there are several other papers of interest as well that discuss using the product in contexts similar to this experiment. [11, 18, 25, 29, and 31] Steinman provides an overview of the use of the framework [26], and discusses the way SPEEDES implements three time management algorithms used in this work. [24]

The primary reference for PVM is Geist et al. [10], *PVM: Parallel Virtual Machine.* However the Carothers et al. paper provides an interesting example of the use of the PVM system with a Time Warp simulation. [3]

2.5 Summary

Some of the literature important to the thesis was presented in the above sections, but space does not permit listing more than a fraction of the material available on the topics. The references listed above were chosen primarily by the degree to which they had been covered in highlighter and tape flags, a rough measure of utility.

CHAPTER THREE:

METHODOLOGY

3.1 Introduction

This research used the Assessment of Simulation Distribution Factors (ASDF) system to compare the effect of three communication configurations which represent common situations a developer might face. In an ASDF experiment, a commercial simulation engine executes a simple discrete event simulation which is capable of responding to external events through an external interface. An independent process generates time-tagged event messages at a specified rate, and delivers them in one of three ways to the third component, the intermediary, which presents the information to the simulation as message events through the engine's external interface. Performance statistics are gathered and analyzed to see what factors were influential. In this research, the focus is on reliability and variability of delivery time, and on simulation overhead.

The rest of the chapter is divided into a discussion of the requirements and motivations for the design, the design of the system, an overview of the data analysis plan, and implementation notes.

3.2 Requirements

Requirements, Goals and Design Constraints

Although no formal requirement document was prepared, the ASDF engineering notebook records the following items as having an important influence on the design:

1. The system should capture some quality of interest from each of three configurations, including the ability to meet a real-time frame requirement, and being subject to different sources of variability.
2. To control extraneous variability and provide an upward growth path, the system should execute on two identical machines from the Raven cluster (described in section 3.3).
3. Since there is difficulty getting funding, the system must not rely on hardware support such as timing cards or reflective memory. Synchronization will be handled by the Network Time Protocol (NTP).
4. The simulation engine should be one actually used in the aerospace industry, preferably object-oriented.
5. The simulation component should be configurable, should have a well-documented external interface,

should have good statistical reporting, and should have repeatable, not stochastic, performance.

6. The system elements will not be optimized to get better performance, but will try to capture the default states as closely as possible.
7. For simplicity, the system will only model an external data stream coming in, and the only reaction will be annotations in log entries. From the simulation's point-of-view, the incoming messages are not essential to continue, and the simulation can complete without them.
8. Initial synchronization can be dependent on a flag event to make sure all components are ready. Set up time is not part of the individual message time cost but is part of the time cost of the run.
9. Messages will be generated at a selectable rate, set as a configuration parameter. The messages will be sent in a "fire-and-forget" mode, with the generator able to complete without response.
10. Any parameters that may ever be of interest should be put in configuration files, in command line arguments, or at the least be easy to change at a header level. The programs should be able to

operate in an ad hoc or scripted mode using the same input parameters.

11. All data, debugging information, configuration files and results should be automatically archived to a test directory, and a post-processing program should extract results of interest to a comma separated value file.

Situations Modeled by the Experiment

The communication architecture was designed to capture qualities relevant to three situations where simulations are being composed from existing elements.

The first case is where the simulation and data source are on different host systems, with inefficient communication, and are subject to both network variability and possible loss of data due to timing out at the end on a time window. This is the ASDF case "m", message-passing/FIFO.

The second case is where the simulation and data source are on either the same host systems or have a hardware-supported shared memory. Communication is efficient and there is low variability, but is subject to loss of data due to time out, and breaking frame may be more critical than in the preceding case. This is the ASDF Case "s", shared memory.

The third and last case is where the simulation and data source are on different physical host systems but are linked by software in a "virtual machine" that mitigates communication loss risk, guaranteeing delivery but not timeliness. This is ASDF Case "v", Virtual Machine.

3.3 System Design

System Elements

The experiments use a pair of computers in the Raven cluster of machines, providing identical high-performance systems. The Raven cluster of dual processor 1.4 GHz Pentium 3 systems all have 256 MB RAM, and are connected via a gigabit Ethernet LAN. Due to other research conducted on the Raven cluster, the network drivers have been tuned for latency rather than optimized for bandwidth.

Only two machines were used, and the experiment was run at a time when there were no other network users to minimize network conflict. The Raven cluster is synchronized using the Network Time Protocol (NTP) facilities of the operating system, but has no hardware synchronization support. The synchronization was not ideal, but since the runs were all performed within a single span of a few hours it may reasonably be supposed

that the degree of skew is constant over the test points. (See Appendix B, Time Skew.)

The simulation framework is The Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES Version 2.0.1). SPEEDES is a product originally created for the National Aeronautics and Space Administration (NASA), has since been frequently used in aerospace applications, and meets all the other requirements discussed in section 3.2. (It should be noted that version 2.0.1 is an older version made available free for research by Metron Inc., and any speed and performance limitations noted in this work should not be considered to apply to more current versions, which have reportedly made significant improvements.) [19, 11]

The Virtual Machine case was implemented using the widely used Parallel Virtual Machine (PVM Version 3.4) system, developed by the University of Tennessee, the Oakridge National Laboratory, and Emory University, and made available from the Massachusetts Institute of Technology. PVM provides services for networking heterogeneous computers to support parallel and concurrent computing. PVM requires a "pvmd" daemon to be running on the machine on which the program will be started, and that PVM must be installed on all machines to be used. After

that, PVM then handles the rest of the connectivity management, guaranteeing in-order delivery of messages, and supporting individual and group communication. [10]

The operating system used for the experiment was Red Hat Linux 9. The code was developed primarily in C++ with configuration management and control functions implemented as Linux shell scripts.

Software Design

The ASDF system consists of five cooperating capabilities, represented in the Unified Modeling Language (UML) diagrams in Appendix A as packages. {See section Package Diagram for ASDF.] Two packages are common to every run, Ships and Harvester, while the other packages, MP for Message Passing/FIFO, SM for Shared Memory, and VM for Virtual Machine, provide variant methods to accomplish the task of inserting external data into the Ships simulation.

During the following discussion of the software design, the reader is invited to refer to the UML class, state, activity and sequence diagrams in Appendix A. To get a picture of what processes are invoked on each host in each configuration, another set of non-UML illustrations called Concurrent Component Activity Diagrams with Artifacts may be useful, although they use

non-standard symbology to depict scripts, configuration files, data files, FIFO pipes and shared memory.

The test system runs a simulation in which four spaceships, represented by Ship objects, patrol an area of space, writing log files and communicating position and status information with each other and with a Base Control object. From time to time their patrolling regime is interrupted by messages from an external Alarm Sensor that sends messages regarding the presence of alien invaders. (See Appendix A, Virtual World of the Simulation.)

This simulation is written in C++ modules that conform to the SPEEDES Application Programming Interface and are compiled with the SPEEDES libraries. The fundamental building block is the simulation object, and the fundamental motivating mechanism is sending a message that schedules an action in virtual time on the event list for an object. Actions, including output to the screen or to a file, can be either rollback-able, or non-rollback-able.

Despite the intricacies of watching the ship objects talk to each other, our true interest lies in the relative efficiency by which the external message is generated and time-stamped (to the microsecond), and is then conveyed from the Alarm Sensor process to the External Module

Process, which puts its own timestamps on it and sends it into the simulation. (See Appendix A, Sequence Diagrams.)

Using the MP package, the Array_mgr class "send" operation is instantiated by member functions that make a Remote Procedure Call to the Listener server on the same host system as the simulation. The Listener process then writes the message to a FIFO named pipe. In parallel, the External Module Manager has been instantiated, and has in turn invoked the correct "read" member functions of the EMComm class to pick up the message from the FIFO. The message is time-stamped and forwarded into the Ships simulation through the External Connection Manager, a service of the SPEEDES Server. The External Module process alternates checking in with the SPEEDES Server to see what the virtual time is, or to deliver a message, and checking to see if a new message is waiting at the FIFO. Therefore, variation in the delivery time at the FIFO might result in two messages arriving too closely to be both handled.

The message generation process and the Ships simulation function identically in the SM shared memory mode. But since all processes are on the same host, semaphores and shared memory replace the Listener and FIFO function in the Array_mgr and EMComm instantiation of the "send" and "read". Since the Listener function provided a

small message buffering effect with the RPC function, the shared memory connection window is tighter and should exhibit less variation in message delivery, representing a hard real-time constraint.

Due to the way PVM establishes the virtual machine, the separate executables for Alarm Sensor and External Module are replaced by separate functions called by child processes of the PVM program "VM_Starter" on the appropriate machines. However, after that change, the lower member functions operate as before, with the "send" and "read" instantiated to use PVM library routines.

No matter the communication method, after the Alarm Sensor finishes the set of messages (currently set at 30 messages) it terminates, followed by the External Module and Ships. This is the cue for the Harvester program, which opens all the configuration and data files and creates a merged database file with all values written in each row, and with one row for every message attempt, whether successful or not.

3.4 Objectives for the Data Analysis Plan

Factors

This phase of the research is looking for the relative importance of communication factors, so the test design varied two factors of importance to the communication environment: the communication mode and the message generation rate. Each factor was varied through three settings. In addition it varied two factors of importance to the simulation environment: the time management strategy and the number of internal nodes managed by the engine; again, each factor was given three settings.

This combination of factors required 81 test configurations, and each was tested with 30 messages, for 2430 data points. The data analysis looked at the four single independent variables one by one, comparing the 810 records associated with each of the three settings to the 1620 associated with the other two settings. The objective was to quantify the effect of each factor by examining the range over which each output parameter varies with each state. (See Appendix B, Analysis Planning.)

The effects associated with each factor are thus summarized by a figure that represents the main effect, the magnitude of changes in the output parameter of

interest. The main effect values for the factors were then compared to establish an ordinal ranking of relative importance. Although the values are real numbers, giving an impression of high precision, the methodology and number of data points temper the statistical confidence that can be placed in the result. Therefore the assessment should be considered as primarily qualitative, rather than quantitative.

Requirements for Measures of Performance

The measures of performance gathered by the running programs were augmented by a number of derived values and descriptive statistics about the data points. The details of the input parameters, the output data values, and the computed and statistical result data are included in Appendix B, Data Dictionary. The objectives were developed during the design phase in the form of questions that framed the actions needed to turn measurements into information. The key questions that drove the choice of measures of performance follow:

1. What is the average delay for each hop?
2. What is the average variability in delay?
3. What is the average inter-message arrival time?
4. What is the average number of rollbacks/events?
5. What is the average number of GVT cycles?

6. What is the average CPU and wall clock time?

7. What is the average CPU time per node?

8. What is the average percent of messages completed?

Descriptions of Measures of Performance

Each step in the message transmission was denoted as a "hop", and the time received minus the time sent was recorded as the delay for that hop. Of the hop times gathered, only the delay between message generation and receipt by the External Module was used, as that is a characteristic of the communication architecture. The inter-message arrival time was calculated as of the point the message started the first hop, as a measure of process variability in the generating process, and also calculated at the arrival at the External Module, as a measure of communication path variability. Rollback and GVT synchronization events are supporting indicators of internal simulation work performed. The measures of time and of successful completion are the most direct measure of the acceptability of the simulation system in a real-world context.

In addition to the measures of performance, another set of questions was developed during the design phase that can only be answered through statistical analysis, while recognizing that this experiment is only a snapshot

of a detail in the full picture of operational effectiveness. These "effectiveness" questions will be discussed in the appropriate sections of Chapter 4.

3.5 Implementation and Execution

Configuration Management

Important tasks for the development effort were maintenance of configuration management, and providing a way to archive results. For these purposes a master source code and data directory was established on the raven0 directory, mounted via the Network File System (NFS) as home0, and various shell scripts were developed to keep everything in synchronization. However, none of the actual code requires that the NFS mount be present.

Actual Conduct of the Experiment

The ASDF program was installed and tested on the raven4 and raven5 computers in the Raven cluster. A script was constructed to automatically execute all 81 test cases. This script was executed March 24, 2006, at a time when there were no other users on the network. The date utility and state of the NTP daemon were queried prior to the run; the results and some inferences on time skew may be found in Appendix B, Time Skew.

3.6 Summary

This chapter has described the motivation, design, and implementation of the ASDF system. The most relevant design documents are to be found in Appendix A. The implementation of the data analysis plan is further illustrated by the materials in Appendix B, which are also extensively discussed in Chapter 4. And finally, the source code for ASDF is included in Appendix C. The most interesting C++ modules have been pulled out to the front of Appendix C.

CHAPTER FOUR:

RESULTS

4.1 Introduction

After all the test runs were performed, the data points for all message delivery attempts were gathered into a single file, which was then analyzed using the data analysis and statistical tools in Microsoft Excel. Since the Harvester data postprocessor labeled each data row with the run identifier and the test configuration settings, all data is fully traceable to the numerous archived raw data and log files, making it possible to use Linux utilities to verify that the data extracted is complete and accurate, and to associate other items of interest, such as the number of GVT events per run.

After a review of the characteristics of the overall data set, a factor-by-factor review was done to see whether any questions could be answered concerning their effect on simulation reliability and performance. This chapter will discuss these topics in turn, followed by a conclusions subsection.

4.2 Characteristics of the Sample

The measurements of interest, as described in section 3.4, were the time taken for the message to arrive at the external module process (delay1); the inter-message arrival time at the point of the initial "send" by the Alarm Sensor and at the point the message was initially received by the simulation Base Controller object (iatAS, iatBC); the numbers of simulation events, rollbacks, and their ratio (e, r, r2e); the CPU and wall clock time used by the simulation (cpu, wall); and the CPU time adjusted for the numbers of simulation nodes (c2node). Basic statistics were computed over the data set as a whole, and also for extracts of each third of the data points associated with a given communication mode, as presented in the Appendix B subsection, Characteristics of the Data - Descriptive Statistics.

The shape of the distributions was examined by creating histograms for delay1, iatAS, iatBC, r2e, cpu, wall, and c2node, and these are presented in Appendix B, Characteristics of the Data - Histograms. For each measurement there are four graphs, one representing the distribution over the full set of 2430 data points in the upper left, with the other three reflecting the

distribution associated with the extract by communication mode: m, s, or v.

The distribution of delay1 has the basic shape of the exponential distribution, reflecting the fact that the median is 0.96 seconds, but a few laggard cases took as long as 330.15 seconds. This is the only result that is a clear fit for a particular distribution. The other measurements reflect the interaction of multiple trends, resulting in mixed distributions. This makes many statistical tests that assume normal distributions inappropriate.

Looking at histograms of the other output parameters in the sets of four, one sees that dividing the data points into three disjoint sets by communication mode does only a little to simplify the distributions, an indication that there is a more complex interaction than just the one factor of communication mode. The exception to this is the Rollback/Event histogram, which reveals that something, at least, was occurring in the shared memory case that was not in the other two cases, though it is not clear what that might be. (The shared memory case will be further discussed in a later section.)

It is worth noting that some types of result data are more appropriately considered on a "by run" level rather

than by message. Examples of this are GVT events per run, message percent by run, rollbacks/events, and jitter. The histograms for GVT and MsgOk are based on by-run summary data. They also are not normally distributed.

4.3 Communication Reliability Results

The first major set of questions to be addressed asks whether any setting(and particularly any one of the three modes) had a distinct effect on the reliability of message delivery, and if so, what the effect was on the simulation effectiveness. Loss of data is a significant problem in some applications, though others can tolerate intermittent dropouts.

To analyze this question, the effect of each of the three states of each factor in turn was isolated in a series of pair-wise comparisons, using the pivot table query function of Excel. For example, to see the effect of the numbers of simulation nodes (nnodes) on the number of messages successfully received (recv), the 810 cases where nnodes = 1 can be compared to the 1620 cases in the complement set where nnodes = 2 or nnodes = 5. The effect of each setting is the amount by which the isolated value varies from the mean, so all three "effect" values sum to zero. The magnitude of the greatest departure from the

mean can be used to compare the relative effect of each factor, as seen in Table 1, an extract from the complete table in Appendix B, The Main Effects of Each Factor on Messages Received.

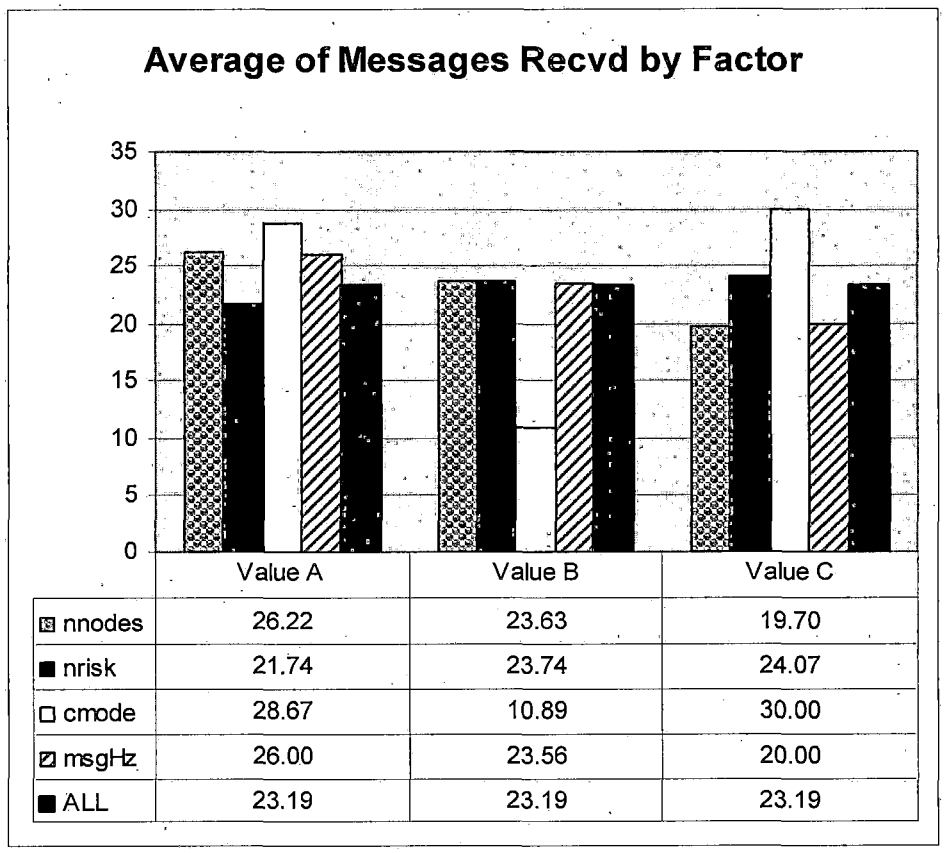
Table 1. Example: Effect of Nodes on Messages Received

CASE 1-6	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nnode	**1**	**2**	**5**	**1 & 2**	**1 & 5**	**2 & 5**
nrisk	(All)	(All)	(All)			
cmode	(All)	(All)	(All)			
mash	(All)	(All)	(All)			
Sum of rcvd	708	638	532	673	620	585
Average of rcvd	26.222	23.630	19.704	24.925	22.963	21.667

	EFFECT A 1 NODE	EFFECT B 2 NODE	EFFECT C 5 NODE
Sum of rcvd	123	18	-141
Average of rcvd	4.556	0.667	-5.222

These calculations are summarized in Figure 1 and Figure 2. Since there are three states to each factor they have been generically labeled with states A, B, and C. By referring to the Analysis Plan or the full version of Table 1 in Appendix B the reader may readily determine that the dramatic difference in the center bar of the set reflects that the communication mode setting (third bar) had an extremely poor rate of message completion in state

B, shared memory, and in state C, virtual machine mode, had a perfect score of 30 complete per run. The number of simulation nodes and message rate had a few differences, but the degree of simulation risk is right at the average (the rightmost bar in each state.)

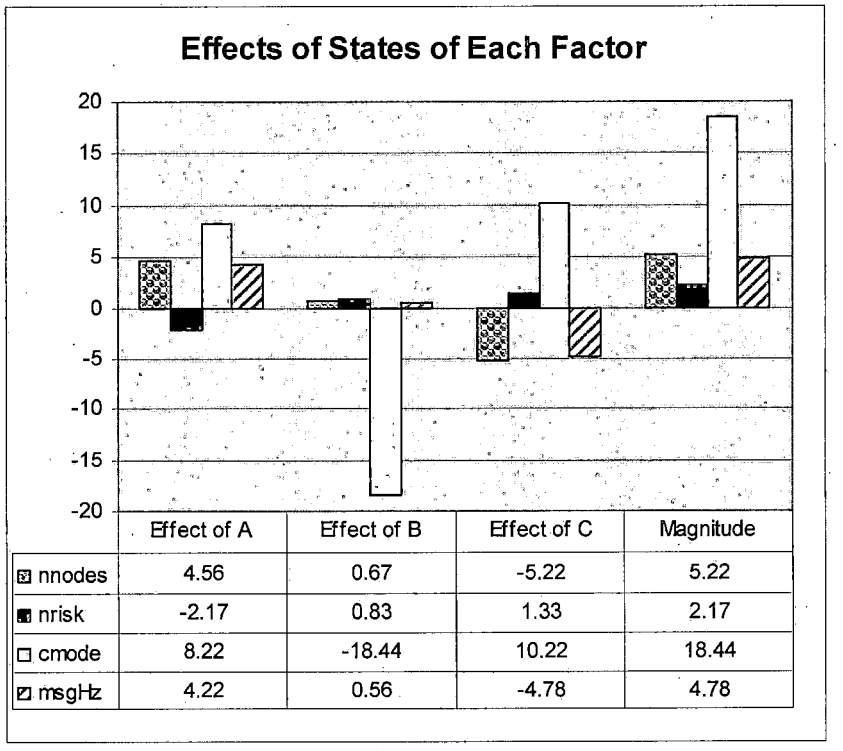


	Value A	Value B	Value C
nnodes	26.22	23.63	19.70
nrisk	21.74	23.74	24.07
cmode	28.67	10.89	30.00
msgHz	26.00	23.56	20.00
ALL	23.19	23.19	23.19

Figure 1. Average of Messages Received by Factor

The effects described in the preceding paragraph become very visible in Figure 2, which compares the relative magnitude of the deviations caused by each

factor. Here it is clear that communication mode has by far the greatest effect on reliability.



	Effect of A	Effect of B	Effect of C	Magnitude
nnodes	4.56	0.67	-5.22	5.22
nrisk	-2.17	0.83	1.33	2.17
cmode	8.22	-18.44	10.22	18.44
msgHz	4.22	0.56	-4.78	4.78

Figure 2. Effect of States of Each Factor on Messages Received

This is rather a surprising finding. The difference in the way the shared memory interface is accessed by the external module compared to the access of data on the FIFO seems to make it more sensitive to delays on the simulation side, making it easier to break frame. This was a somewhat expected result, but the magnitude of the

difference is much greater than expected. There is very little overall correlation (-0.046) between the complexity of the simulation internal processing (as measured by the number of GVT events) and the message success rate, so delays in simulation interface may not play as big a role as suspected. It appears that the use of more non-blocking calls increases the temporal sensitivity and causes the External Module to miss the window, and incidentally, makes this a better model for a hard real-time interface. It is also possible there are other inefficiencies inherent to shared memory at work, similar to those described by Kranz et al. [13]

Another question that can be addressed is how likely it is that the difference associated with each factor setting is significantly different from the average of all data points. For that comparison the average for all settings was compared to the average for the setting in question, and the results generally show better than 95% confidence that the differences are significant, with the exceptions of nnode = 2, 61% confidence, nrisk = M, 71%, nrisk = H, 92%, and msghz = 2, 53%. This confirms that the time management style is essentially unimportant to dropout rates. This assessment is found in Appendix B, Worksheet Used to Find Confidence Interval, and Confidence

Intervals for Messages Received. The pair-wise comparisons were also examined by this technique but did not yield different insights, and have not been included here.

4.4 Simulation Performance Results

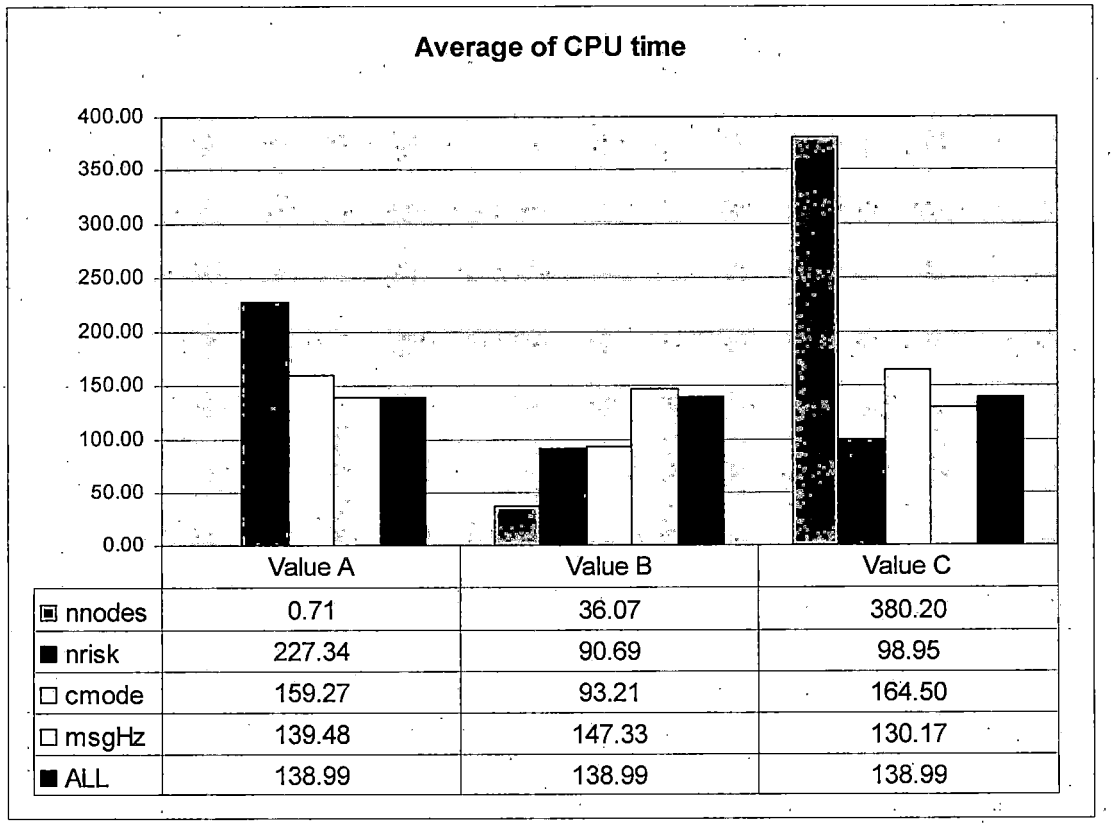
The design phase of the experiment generated three questions to be answered in this area:

1. What is the relationship between external communication method and simulation performance?
2. Is the effect of the external communication method greater than the effect of time management risk?
3. Is the effect of the external communication method greater than the effect of the number of simulation nodes? (i.e. greater than the effect of the degree of internal simulation complexity?)

In this section the discussion and the figures will focus on the total CPU time reported by the simulation engine as a measure of work performed by the simulation. Measurements were also taken which recorded the wall clock time taken, the CPU time per simulation node, the number of rollback actions over the number of events, and the number of GVT events, all of which are also measuring some aspect of work. All of these other measures are included in the results in Appendix B, but they are quite

consistent with the CPU time data, and are not separately shown in the text. (For example, the correlation between CPU and Wall Clock time is better than 99.6%.)

All of the time measures refer to the work performed by the simulation only and do not include any work involved with the external communication mechanism or message generation program.



	Value A	Value B	Value C
nnodes	0.71	36.07	380.20
nrisk	227.34	90.69	98.95
cmode	159.27	93.21	164.50
msgHz	139.48	147.33	130.17
ALL	138.99	138.99	138.99

Figure 3. Average of CPU Time by Factor

Figure 3 illustrates the simulation CPU time under the various experimental conditions. The four factors in each of three states (A, B, and C) are shown, with the rightmost bar in each state representing the average for the entire data set, included for comparisons. For "nnodes", the three states A, B, and C represent 1, 2 and 5 nodes repectively. For "nrisk" the values were Low, Medium and High, for "cmode" they are "m", "s", and "v", and for "msgHz" they are 1 Hz, 2 Hz, and 10Hz.

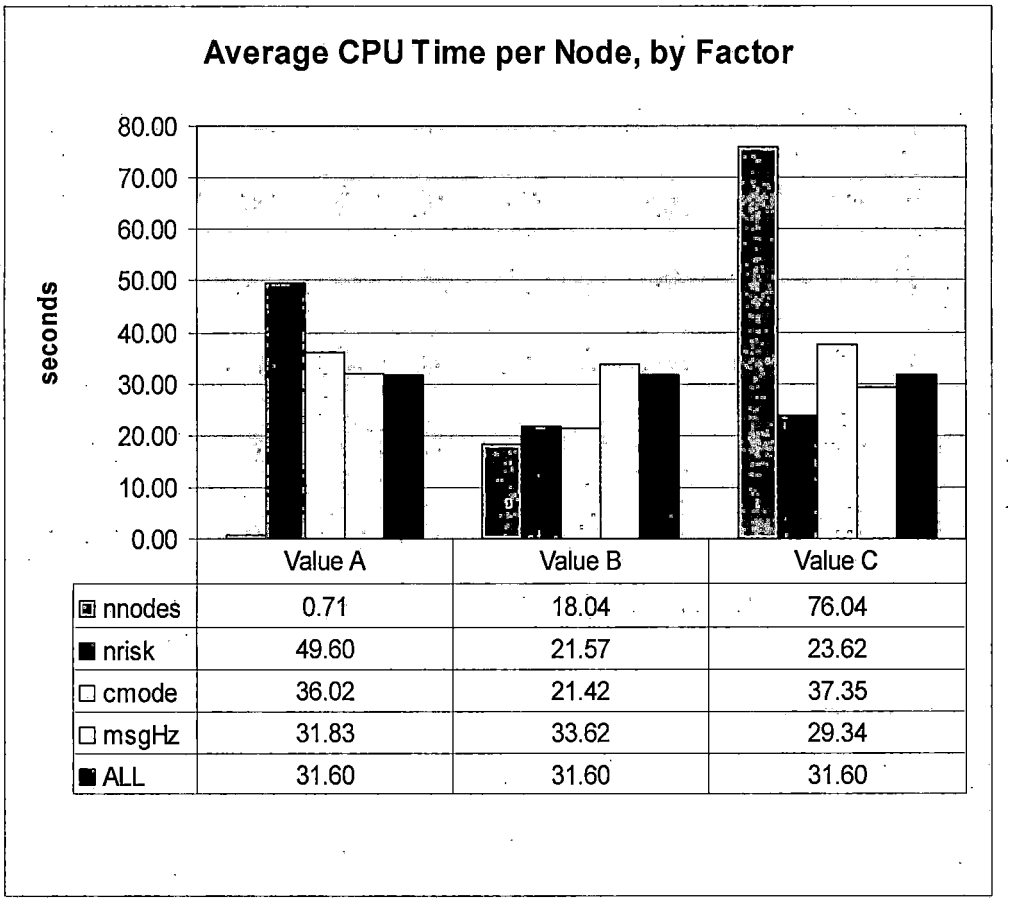
Comparing the time for each factor and for the total set of data points, it is clear that the number of simulation nodes (leftmost bar, labeled "nnodes") makes a drastic difference. When the configuration file indicates all logical processes are on a single node on a single host, the SPEEDES engine uses efficient shared memory internal communication instead of its normal socket based messaging. [18, 19] Thus, the disparity between the work done in the single node and two node state is not surprising, but the difference between two and five nodes is also striking. The stress case, with five logical processes competing for resources on a single dual processor machine, proved to be about the most complex this un-optimized simulation could handle. Preparatory

experiments demonstrated that a scenario with six or more nodes frequently failed to complete the simulation.

The second bar, labeled "nrisk", reveals that the "low" time management setting involved a noticeable increase in simulation work over the medium and high risk protocols. This will be further addressed later.

The third bar, "cmode", involving the communication mode, does show a difference in the work performed by the simulation, with the shared memory mode requiring less work. This effect makes shared memory appear more efficient, but it is probably due to the fact that shared memory data transfers fail more often, sending fewer messages in to the simulation, resulting in less work.

The last two bars reflect the message generation rate ("msghz"), and overall average ("ALL"). The incoming message rate does not appear to have much relationship with CPU time for the simulator, contrary to hypothesis. The data summary in Appendix B, The Main Effects of Each Factor on CPU Time, shows that the mean and the variance for the 1 Hz and 2 Hz rate are extremely close to the overall mean and variance. Figure 4 shows the same thing.



	Value A	Value B	Value C
nnodes	0.71	18.04	76.04
nrisk	49.60	21.57	23.62
cmode	36.02	21.42	37.35
msgHz	31.83	33.62	29.34
ALL	31.60	31.60	31.60

Figure 4. Average CPU Time per Node, by Factor

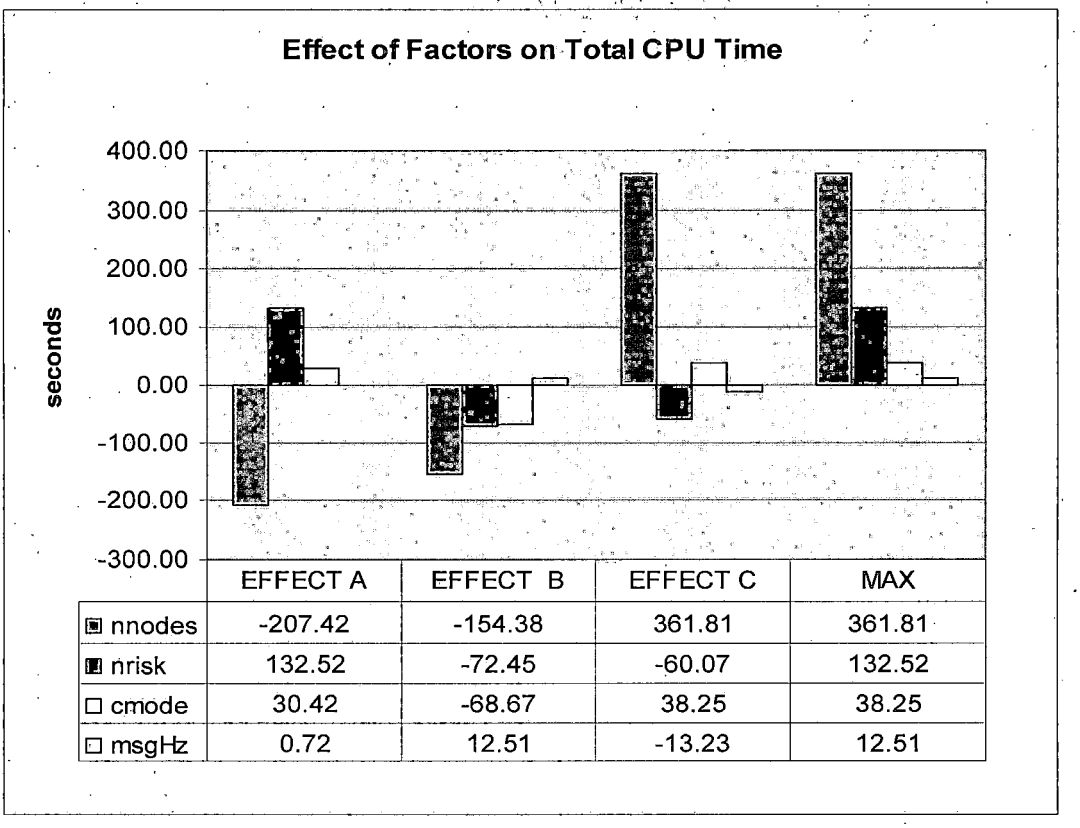
The highest message rate used, 10 MHz, was difficult for the lower powered laptop on which much of the development work was performed, but it seems probable that this "high" rate was insufficient to distinguish rate effects on Raven-class machines.

There is an interesting point of comparison between Figure 3, which shows the total CPU time, and Figure 4, showing the CPU adjusted by node. Although dividing by the

number of internal nodes acts to smooth out the differences between the other factors somewhat, it only highlights the disproportionate role played by simulation-internal communication. This can also be seen in Figure 5, which shows the effect each setting of each factor has on the total CPU time when compared to the other two settings.

In Figure 5 the first three columns show what effect each setting of each factor produced in terms of moving the value away from the mean, with the magnitude of largest displacement in the fourth column, labeled MAX. The ranking in MAX is a way to visualize the relative performance impact of the factors, and provides quick visual answers to the three questions that started this chapter. Number of nodes has the greatest impact by far, time management risk is a distant second, but communication mode is more influential than message rate.

In Figure 5 it is very clear that the data rate of message generation has virtually no effect on the work done by the simulation, indicating that the other effects are not due simply to inadequate resources to complete within the real-time frame.



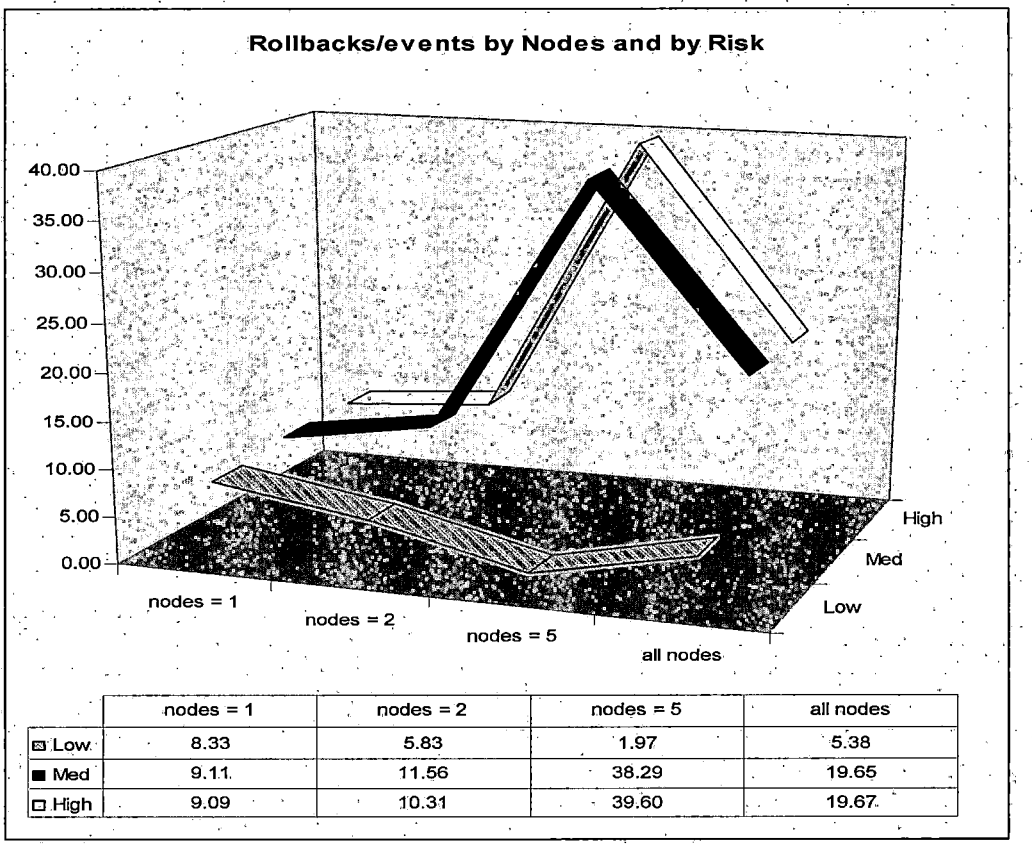
	EFFECT A	EFFECT B	EFFECT C	MAX
nnodes	-207.42	-154.38	361.81	361.81
nrisk	132.52	-72.45	-60.07	132.52
cmode	30.42	-68.67	38.25	38.25
msgHz	0.72	12.51	-13.23	12.51

Figure 5. Effect of Factors on Total CPU Time

Time Management Effect on Rollback

Examining the rollback/event measurement provided a surprise that merits further investigation. Based on the literature [19, 24], the Breathing Time Buckets (BTB) algorithm was expected to be substantially lower in risk of rollbacks than Time Warp (TW), though at the cost of increased costly GVT events. The medium risk BTW protocol actually alternates between the other two algorithms and was expected to be right in the middle in rollback ratio.

Instead, the ratio is close for all three cases, and virtually identical in the case of the medium risk BTW and high risk TW case. (See Figure 6).



	nodes = 1	nodes = 2	nodes = 5	all nodes
Low	8.33	5.83	1.97	5.38
Med	9.11	11.56	38.29	19.65
High	9.09	10.31	39.60	19.67

Figure 6. Rollbacks per events by Nodes, Risk

An obvious possibility is that something about the way this particular scenario plays out fails to trigger the change to the state where SPEEDES is using the BTB algorithm. Although the middle value is not as expected,

we do see the effect of the high and low setting, so the magnitude difference figure is still trustworthy.

Variability Effects

The last question from the original measures of effectiveness set has a slightly different motivation from the others. In papers and in conversations with simulation developers [22, 28, 30, 31], the collective wisdom is that variability in information delivery is a great problem, often mentioned. But it is primarily regarded as a symptom associated with latency problems, and appears to seldom be studied on its own. In this work the relevant question is: "Does variability of delivery time have an identifiable influence on overall simulation performance?"

The approach taken was to look at variation in terms of the jitter in three measurements: the inter-message arrival time at the point Alarm Sensor is sending the message out (iatAS) the inter-message arrival time at the point the message is received by the simulation (iatBC), and the measurement of the latency of the path from generator to External Module (delay1). For the purposes of this work jitter is calculated by finding the minimum, maximum and mean over a range, and using the maximum displacement from the mean. Thus it is only meaningful to

speak of jitter with respect to a particular range of data points.

The iatAS jitter (which really should probably have been called inter-message departure time in this case) represents the variation in the process by which a wake-up signal is generated and then handled by the timer manager and a message then is assembled for transmit. If perfect, the difference would be equal to 1 second divided by the message Hertz rate. The jitter in the iatBC measurement compounds the original variation with the variability in the communication path, and it is the measure that logically should predict simulation performance effects. The jitter in delay1 captures just the variations due to the communication path.

The range of measured values used was at the test level, taking the mean, minimum and maximum over the 30 data points associated with one configuration of input parameters.

The output parameters for CPU time, CPU per node, and the ratio of rollbacks to events were also grouped by test and averaged. The number of GVT events per test was already in the form of the 81 row table, so it was included in the correlation. Table 2 presents the results of the correlation matrix.

Table 2. Correlations to Jitter

	JitterAS	*JitterBC*	*JitterDel*	*GVTs*	*Avg cpu*	*Avg r2e*
JitterAS	1					
JitterBC	0.61521	1				
JitterDel	-0.00183	0.66232	1			
GVTs	0.50618	0.86360	0.61313	1		
Avg cpu	0.50697	0.88229	0.71036	0.95334	1	
Avg r2e	0.09605	-0.00539	0.21837	0.03938	0.22872	1

The results in Table 2 indicate that while jitter may play a role in simulation performance, it is by no means as dominant as suggested by conversations with practitioners. The jitter at the point the new information is available to the simulation, represented by iatBC, does have a moderately strong correlation with the number of GVT events and with simulation CPU events, but not, apparently, through the mechanism one would suppose, by creating more rollback events.

However, with just a little more work a different picture appears. Since the grouped data in the "by run" forms not only preserved the test number but also sorted the rows on it, it was easy to partition the data for mode-specific correlations. Table 3 presents the same data, but now correlated within the three communication modes. For communication mode "m" this breakout now shows a strong correlation between jitter in all three

measurement spots with the work the simulation must do. On the other hand, the correlations with mode "s" are weak and inconsistent. Interestingly, the correlation to jitter at the source is weak for mode "v", but strong for the jitter at the point of arrival at the simulation, after the message has been through the buffering action of the PVM transport mechanism.

Table 3. Correlations to Jitter by Communication Mode

cmode=m

	JitterAS	*JitterBC*	*JitterDel*	*GVTs*	*Avg cpu*	*Avg r2e*
JitterAS	1					
JitterBC	0.878642	1				
JitterDel	0.930708	0.97527	1			
GVTs	0.869028	0.985187	0.963051	1		
Avg cpu	0.867078	0.954837	0.945796	0.981507	1	
Avg r2e	0.205377	0.023961	0.089511	0.075055	0.251752	1

cmode= s

	JitterAS	*JitterBC*	*JitterDel*	*GVTs*	*Avg cpu*	*Avg r2e*
JitterAS	1					
JitterBC	0.763556	1				
JitterDel	0.449328	0.90743	1			
GVTs	0.411182	0.67582	0.701773	1		
Avg cpu	0.441639	0.684893	0.706944	0.988263	1	
Avg r2e	-0.10942	-0.29062	-0.30332	-0.35764	-0.31683	1

cmode= v

	JitterAS	*JitterBC*	*JitterDel*	*GVTs*	*Avg cpu*	*Avg r2e*
JitterAS	1					
JitterBC	0.014549	1				
JitterDel	0.178765	0.943906	1			
GVTs	0.129787	0.947599	0.935416	1		
Avg cpu	0.205473	0.945601	0.995029	0.94367	1	
Avg r2e	0.601229	-0.01006	0.261065	0.052692	0.278334	1

The effect of jitter clearly is dependent on the communication method involved. Since the correlations in the message passing/FIFO and virtual machine cases are quite a bit higher, it appears probable that the high number of dropped messages in the shared memory mode is disrupting the normal effect of jitter. It is tempting to speculate that the simulation has more time to complete work between messages when there are far fewer messages, but it is premature to conclude that is the actual reason.

This table also confirmss that jitter in an intermediate message transportation transfer point ("JitterAS") or in the message latency "JitterDel"), matters much less than jitter at the point where data becomes available to the simulation ("JitterBC").

4.5 Summary

To summarize the findings, it is important to return to the two hypotheses with which we began. The first hypothesis was that the effects of the primary factors of data rate, time management style and internal simulation complexity would far overwhelm the effects of the secondary factors relating to communication mechanism.

The findings on this point are mixed, depending on what measure of performance is being considered. The

communication mode turned out to be overwhelmingly important to the message completion rate, though without a truly satisfactory explanation as to why the shared memory mode was so unsuccessful. It can fairly safely be concluded, however, that if one has a simulation with a hard-real-time requirement, this is not a good way to build it.

However, the primacy of the number of nodes as a predictor of simulation work, overhead, and wall clock time was unchallenged, and the time management style did in fact prove to be quite important. Against the hypothesis, the message generation rate turned out to be unimportant, possibly indicating that the rates were poorly chosen and the higher rate represented too little of a challenge to the power of the Raven cluster.

The second hypothesis is more clearly confirmed. It did in fact prove possible to identify effects from the interaction of communication mode with other factors to suggest some of the trade-offs. The buffering effect of the PVM environment could be seen to reduce both jitter and message loss despite not having much effect on simulation overhead time, and might be a good choice where dropping messages is not acceptable. But, just in terms of the simulation performance, the inefficient chain of RPC

and FIFO worked about as well, so time reworking such a chain is probably less useful than trying to improve the use of the time management algorithm unless dropouts are an issue. It really does not seem to be worth trying to tune the message generation rate, although there are logical limits to that assertion.

The methodology of this experiment suggests that these findings should be regarded as qualitative rather than quantitative. The following chapter has some suggestions for increasing the confidence of these results.

CHAPTER FIVE:

VALIDATION

5.1 Introduction

This chapter discusses validation of the results obtained in this experiment, and covers possible avenues for future work. A summary of the work completes the chapter.

5.2 Validation

Law and Kelton [15] define validation as follows: "*Validation* is the *process* of determining whether a simulation model (as opposed to the computer program) is an accurate representation of the system, *for the particular objectives of the study.*" This presents a difficulty in a comparative study such as this where there is no real-world analog about which we can make statements that independent observers can rate for accuracy. But there is hope in the last clause, "... *for the particular objectives of the study.*" If simulation developers find ASDF to provide useful indicators, it succeeds in the objectives of the work, although actual validity will be a fuzzy thing indeed. One experiment is not enough to give any developer confidence in the results, so the validation

process must be to replicate the experiment in such a way that confidence increases each time.

One of the design decisions which limits the generality of this work was to perform all test runs at one time, on an unloaded network, with systems that were not performing other user tasks. Although this reduced the variation between test runs, it also removed a great deal of realism from the modeled environment. Now that the ASDF system has been created and the current results have been captured, this data set may be used as a baseline. A number of test replications under less tightly controlled circumstances will create a more comprehensive and more widely applicable statistical picture. This will help validate that the results are generally applicable.

As Carothers et al. [3] have noted, distributed discrete event simulations that execute on workstations in an environment where the network and computers are shared with other applications are "subject to uncontrollable external loads" and this can "often result in load imbalances and dynamic fluctuations in delivered resources which can be a major source of performance degradation." His solution is to augment the information available via the simulation itself with a "Graphic Visualization" tool that runs on top of PVM. This program captures and

displays information about the network status, memory usage, and the percent of time each process is spending on doing computational work versus executing message send and receive actions. Adding a visualization utility such as this one to ASDF would not remove fluctuations, but could provide explanations to improve understanding of these effects. Explaining these influences would add to the validity of the ASDF results.

5.3 Future Work

There are many other areas into which this research could productively be extended. With no changes to the run-time code, the number of factors to be varied could be extended to look at a wider range of message rates and variations of the simulation's virtual pacing rate, and the interactions between the factors could be examined in more detail while holding the number of simulation nodes constant and small. Very minimal changes would be needed to support investigations of the effects of different message lengths with network latency tuning. The effects of scaling up from one external data source to many would be interesting to investigate.

In the field, a frequent part of the solution to integrating disparate data sources with distributed

simulations is to provide hardware support for clock synchronization and memory transfers. It would be extremely interesting to compare the shared memory mode used in this work with a true reflective memory system. [13, 14, 29, 30] Noticeable improvements might be obtained merely by adding timing cards with interrupt capability that reference an external time source.

5.4 Summary

Validation of this work will be an incremental process, as there is not a single real-world system being modeled. Further development should expand the understanding of the factors by varying the test conditions.

APPENDIX A

DESIGN DOCUMENTS FOR THE ASDF PROGRAM

SUBSECTIONS OF APPENDIX A: DESIGN DOCUMENT

A1. Configurations

A2. Virtual World of the Simulation

A3. Package Diagram for ASDF

A4. UML Diagram Notes

A5. Class Diagrams

A6. Concurrent Component Activity Diagrams with Artifacts

A7. Sequence Diagrams

A1. Configurations

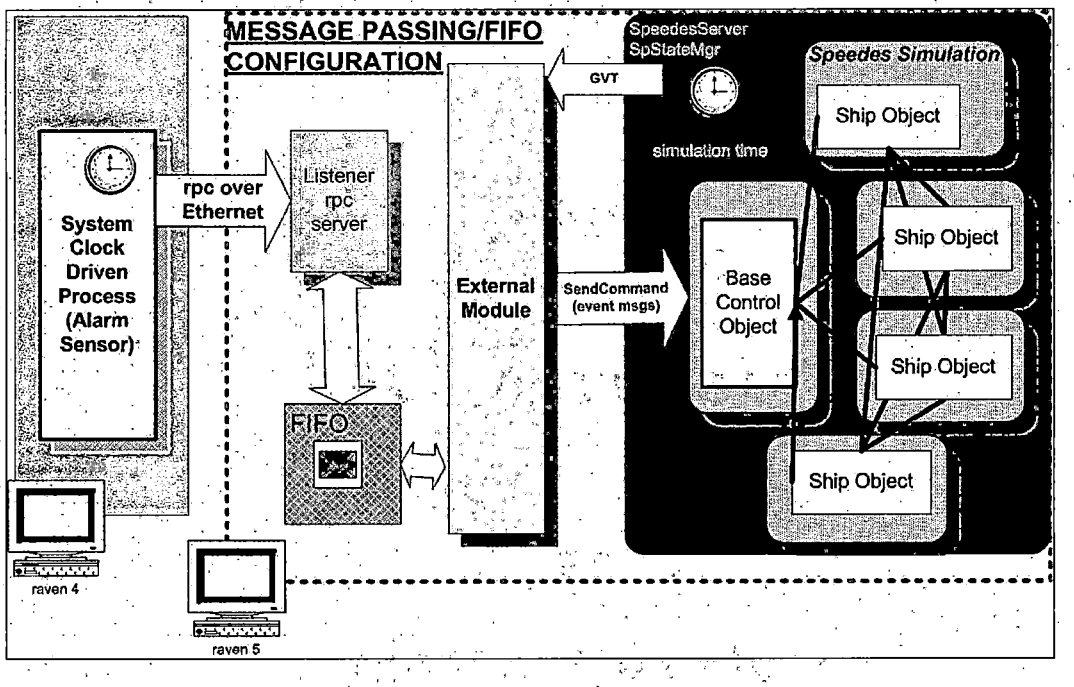


Figure A 1 Message Passing/FIFO Mode Configuration

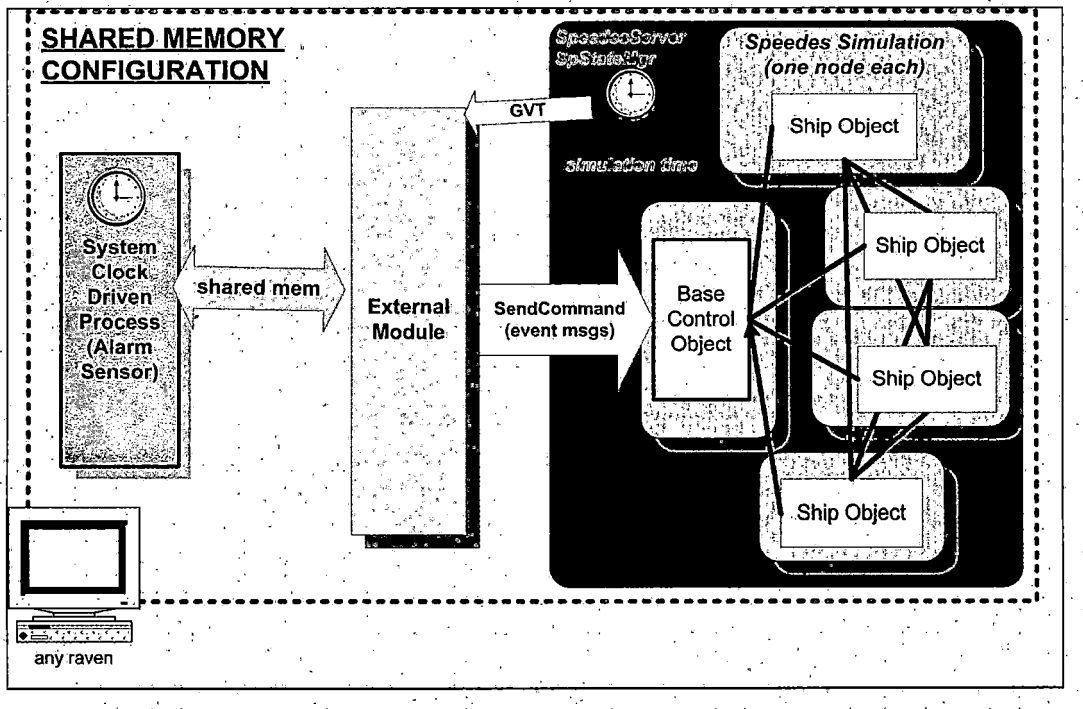


Figure A 2 Shared Memory Mode Configuration

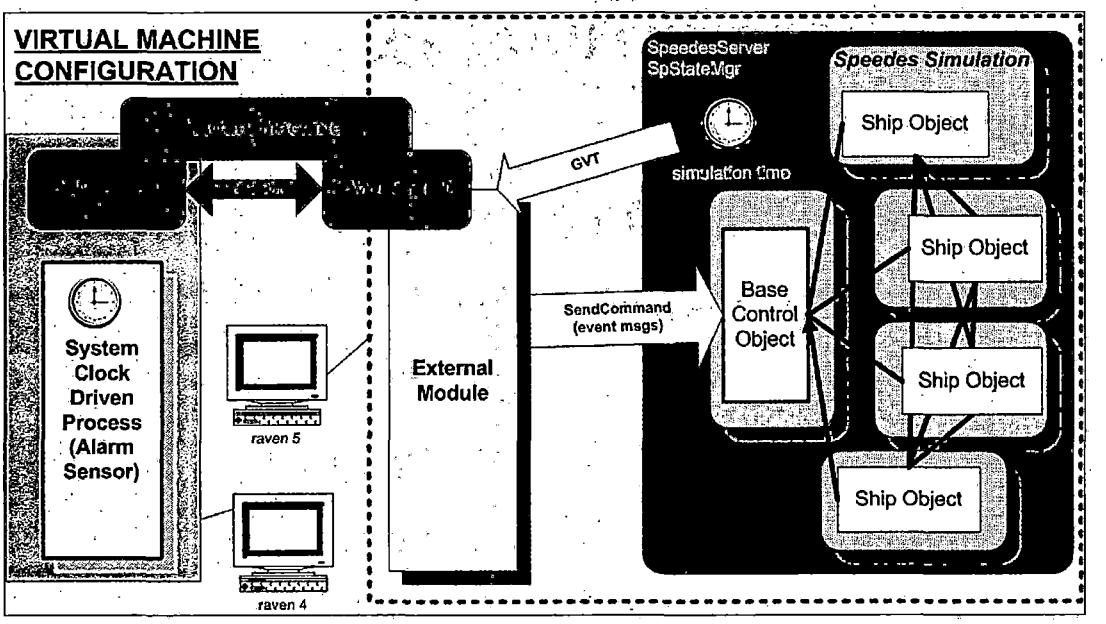


Figure A 3 Virtual Machine Mode Configuration

In all three configurations SpeedesServer, the five Ship simulation objects and External Module are on host raven5, and communicate via SpeedesServer socket-based communication routines. In the Message Passing/FIFO and Virtual Machine configuration the Alarm Sensor process is on host raven4, but in the Shared Memory configuration it too is on raven5.

A2. Virtual World of the Simulation

The following diagram depicts the environment being simulated. The work done by the four Ship objects and Base Control is subject to rollback; not that by Alarm Sensor.

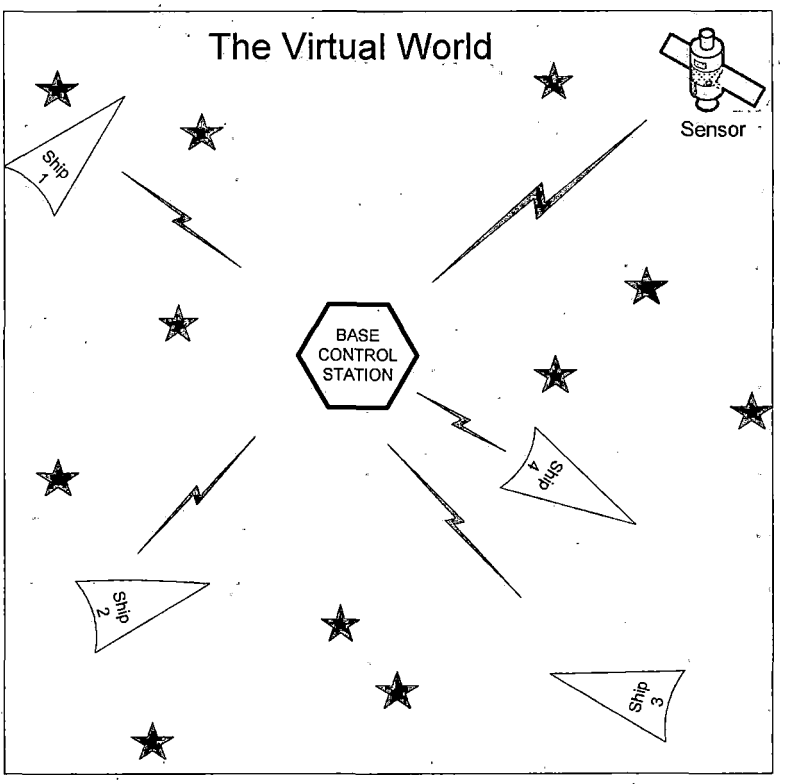


Figure A 4 Virtual World

Base Control exchanges regular status messages with four patrol ships, keeps a log, and retransmits Alien Detection messages from the border sensor. The successful delivery of these messages is the primary measure of success. The Ships patrol the region independently with varying speeds and trajectories. They exchange position messages with each other, check in with Base Control, and write log files of events. The Alarm Sensor sends periodic one-way broadcasts of Alien Detection messages, which can cause rollbacks in other objects.

A3. Package Diagram for ASDF

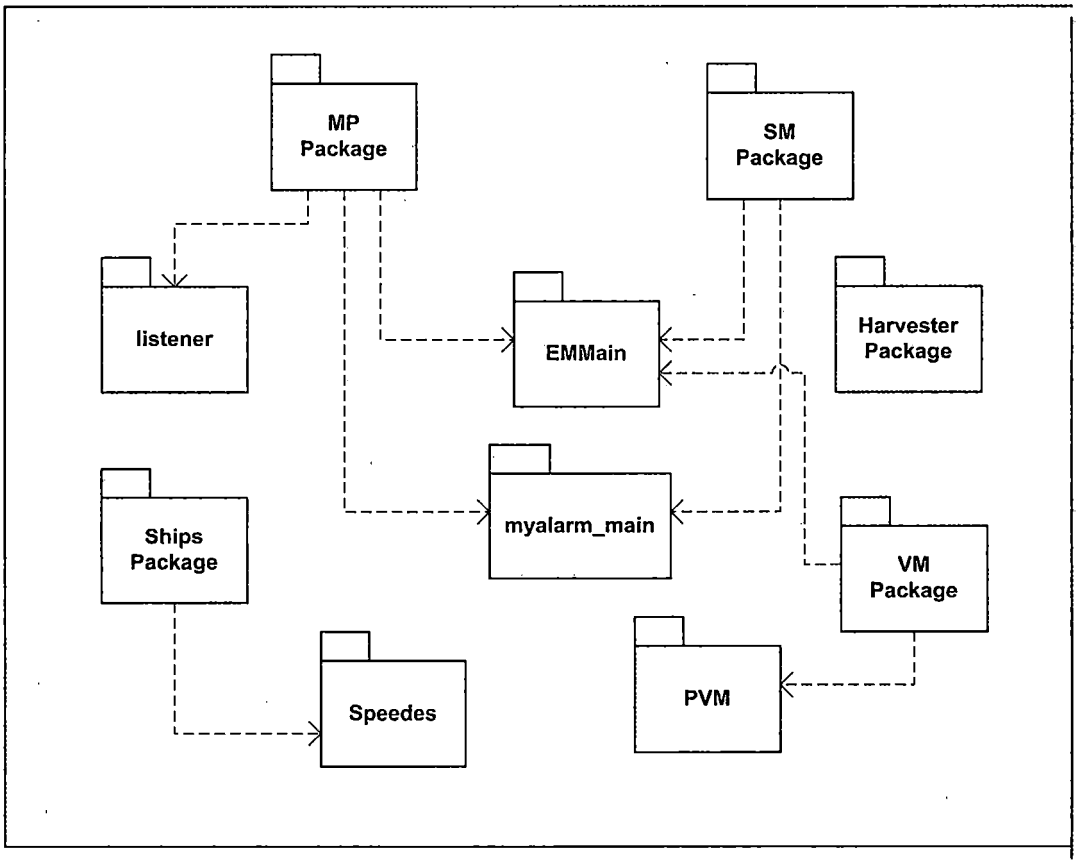


Figure A 5 Package Diagram

The MP and SM Packages implement the Message Passing/FIFO and Shared Memory communication modes and depend on the same versions of the EMMain and myalarm_main packages, but MP also depends on the RPC server, Listener. The VM package, implementing the Virtual Machine mode, includes a customized version of the myalarm_main class

Array_mgr, but depends on the same version of the ExternalModuleMgr class in the EMMain package. The SPEEDES and PVM systems are shown as packages representing Application Programming Interfaces (APIs) for the commercial routines. The Ships package contains the mysim package and various configuration files. The Harvester package stands alone.

A4. UML Diagram Notes

The following set of diagrams is based on the UML standard from the time the design work began in 2001. However, since the purpose was to support the design work as a key component of the ASDF Engineering Notebook, rather than for communication with customers or other developers, several diagrams have been adapted to suit the visualization needs of this work. The ideas embodied in Bruce Powel Douglass's *Real-time UML* [6] were particularly useful, although not many of his specialized diagrams were used in the end.

A5. Class Diagrams

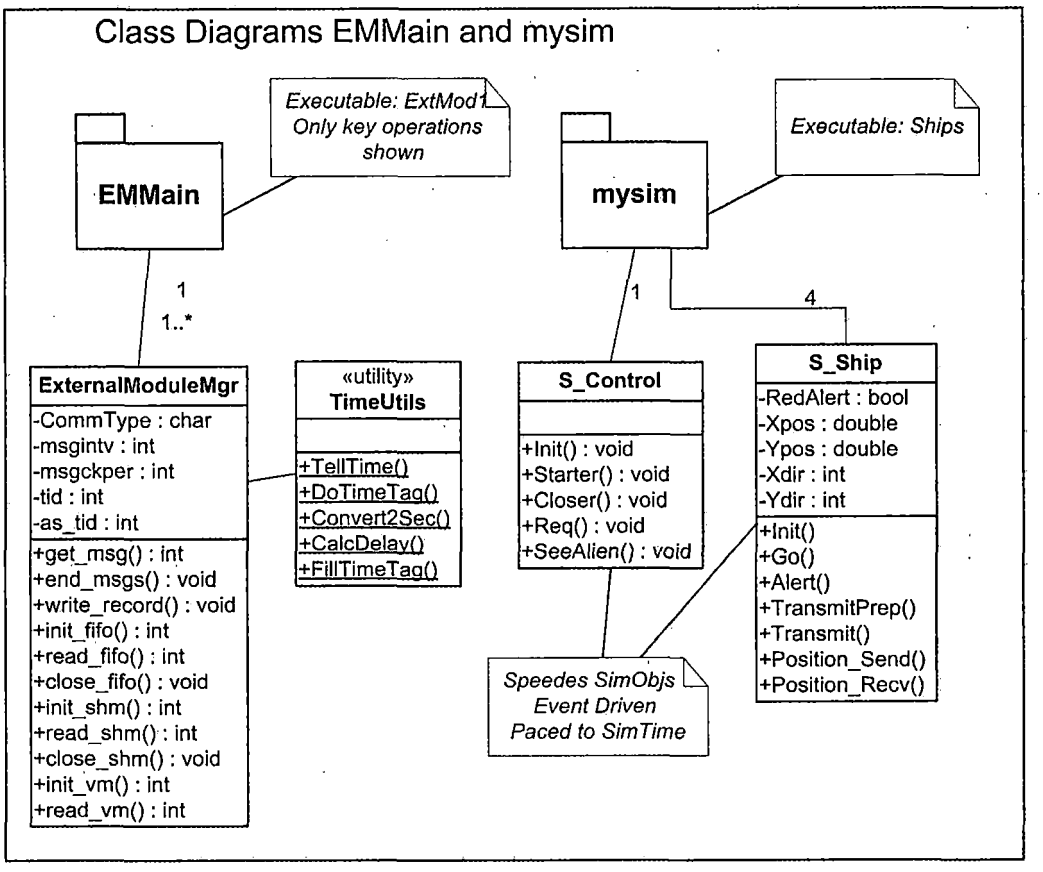


Figure A 6 Class Diagrams EMMain and mysim

The EMMain package is responsible for the job of being the intermediary between the SPEEDES external interface provided by SpeedesServer and the appropriate communication mode.

The S_Control and S_Ship classes are instantiated under the SPEEDES framework.

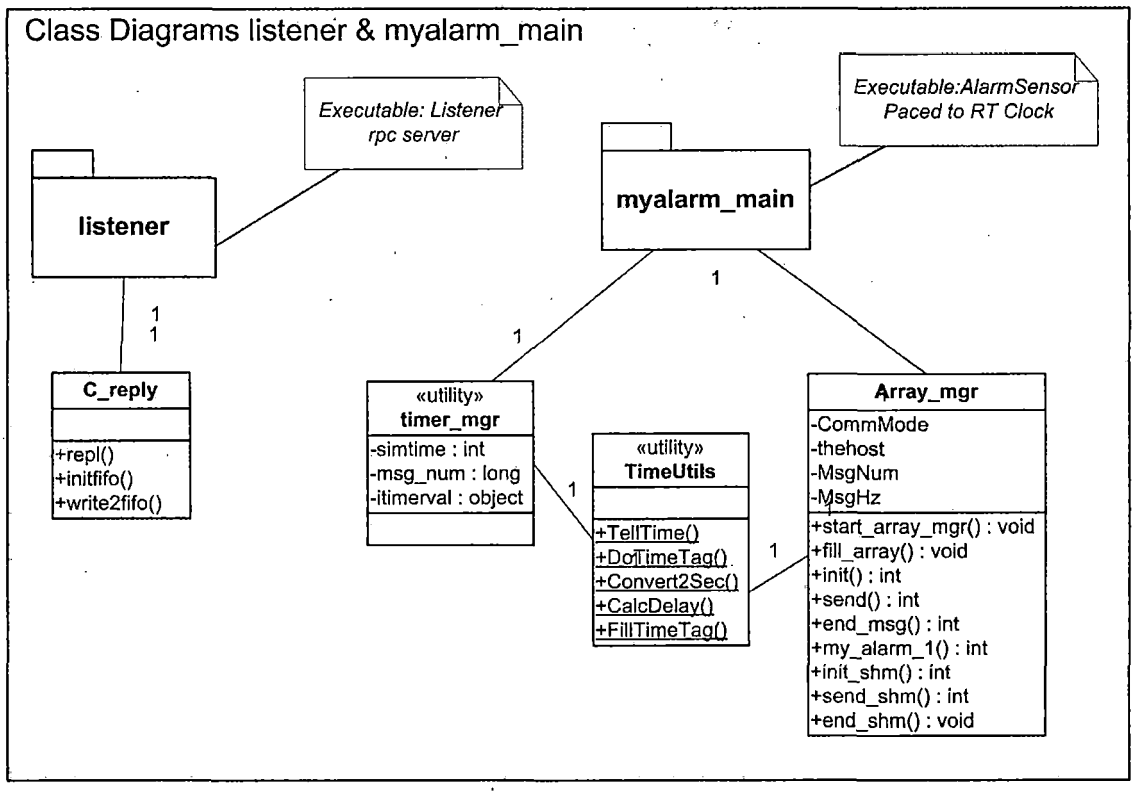


Figure A 7 Class Diagrams for listener and myalarm_sim

The listener package provides an RPC server that writes to a named pipe.

The myalarm_main package contains the timer function, timer_mgr, which uses signals to wake periodically. It instantiates the Array_mgr object to handle the packaging and sending of the message array.

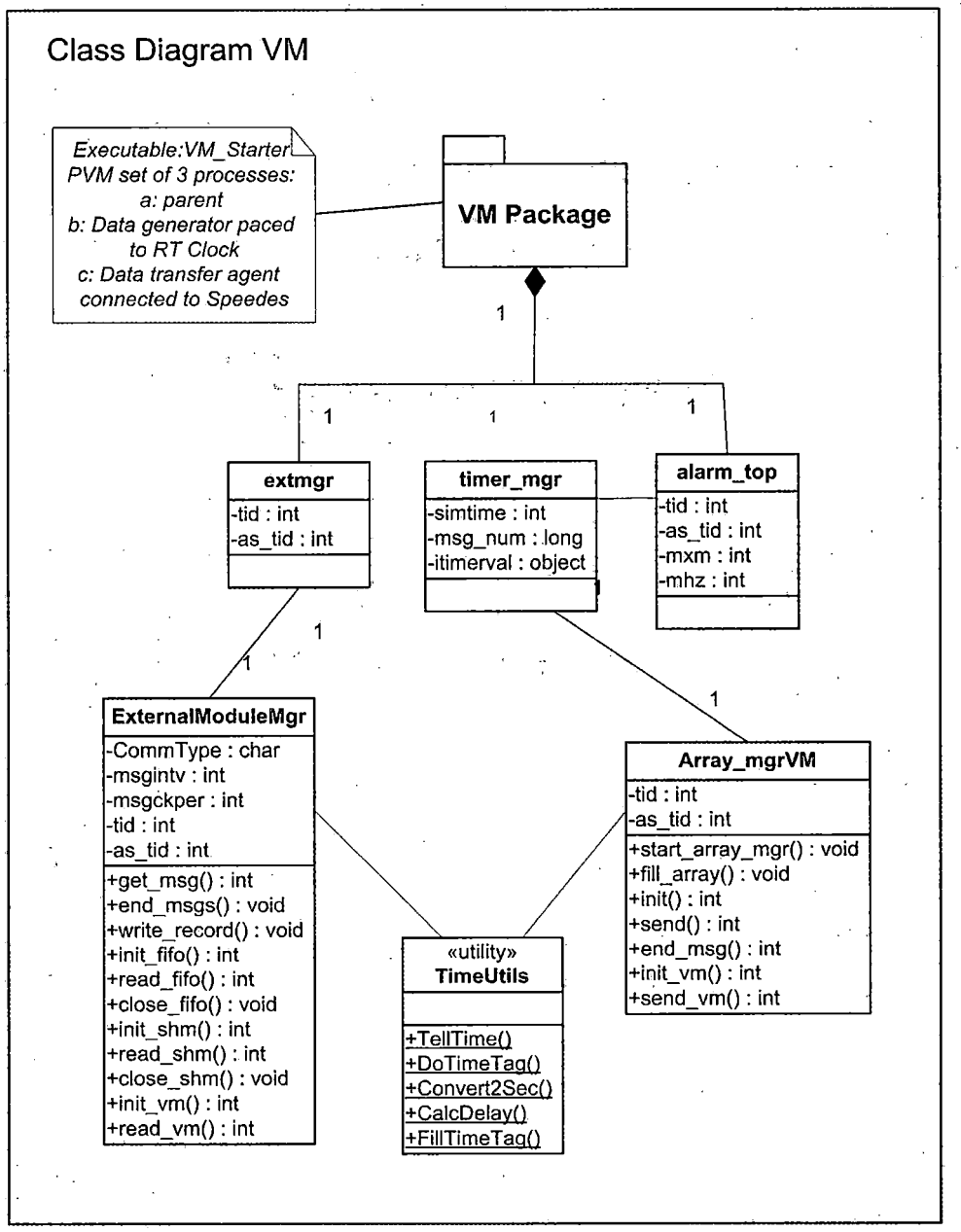


Figure A 8 Class Diagram VM

The VM package combines the responsibility of both the External Manager and Alarm Sensor Capabilities. The main program is written to use PVM services. It runs as a

PVM parent, and then invokes two PVM daemons, one on each host, to start two copies of itself. Depending on the task identification, one child process instantiates the ExternalModuleMgr, and the other, ArrayMgr_VM.

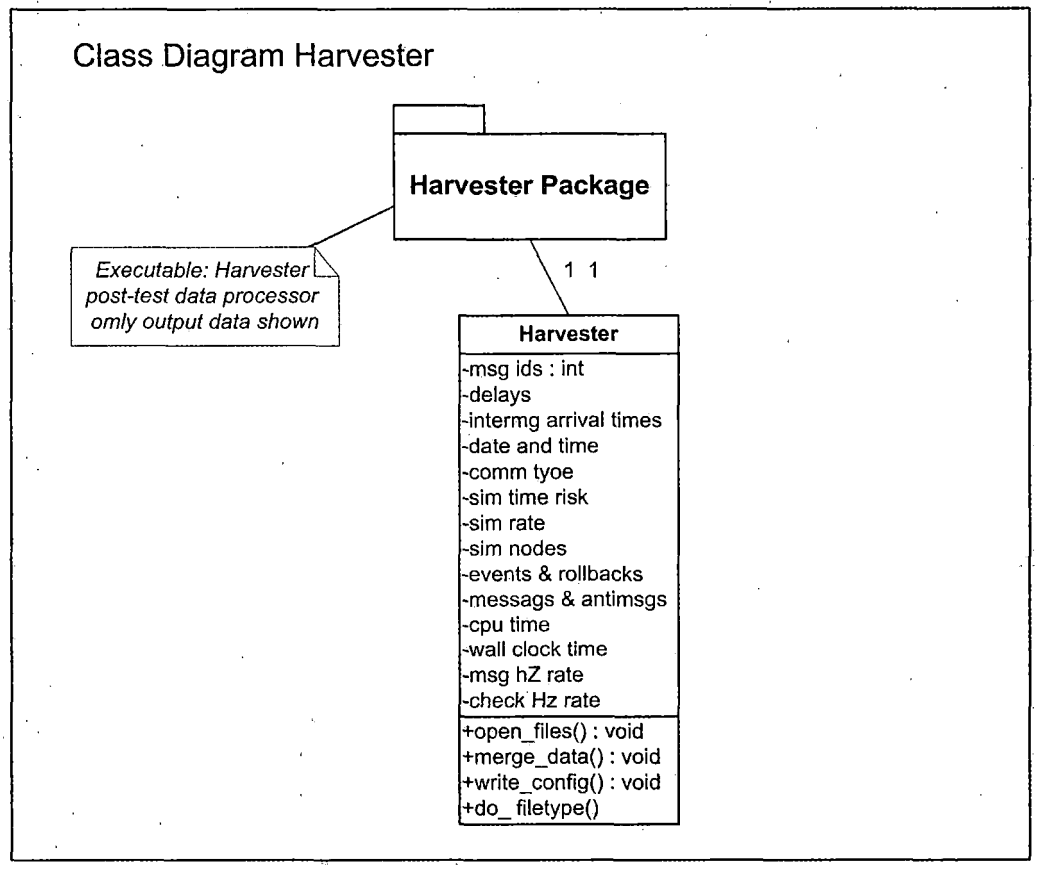


Figure A 9 Class Diagram Harvester

The Harvester package consists of a single class which does post-processing, merging the data gathered from a number of files into two comma separated value files.

A6. Concurrent Component Activity Diagrams with Artifacts

The following three diagrams diverge from the UML standard to capture the sequence of start up and shut down process control associated with each component on each specific host. They also record the associated scripts, configuration files, data files and other artifacts. These diagrams borrow the component and node symbols from component diagrams, and the use of arrows to capture threads of control and transition actions from activity diagrams.

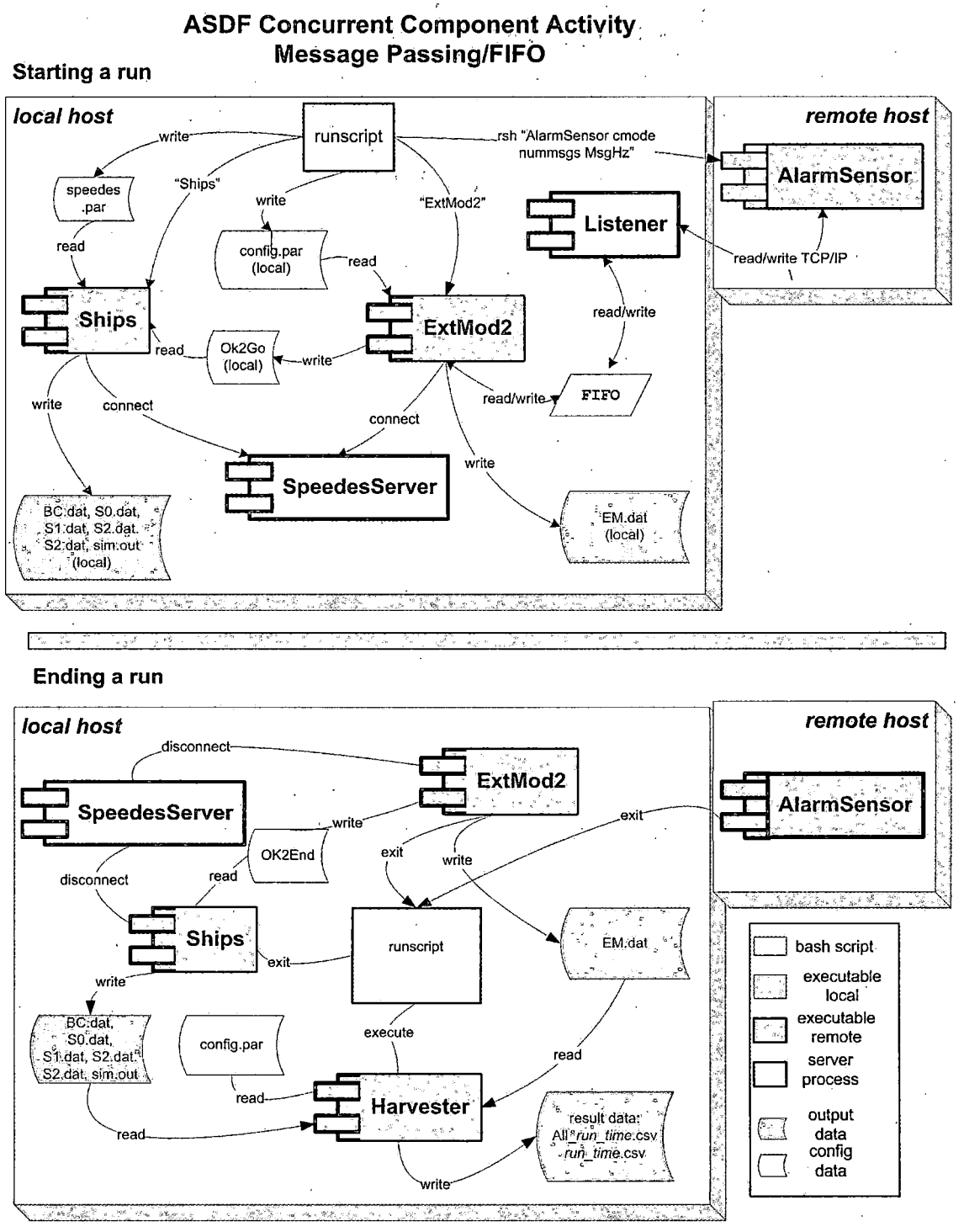


Figure A 10 Concurrent Component Activity Mode M

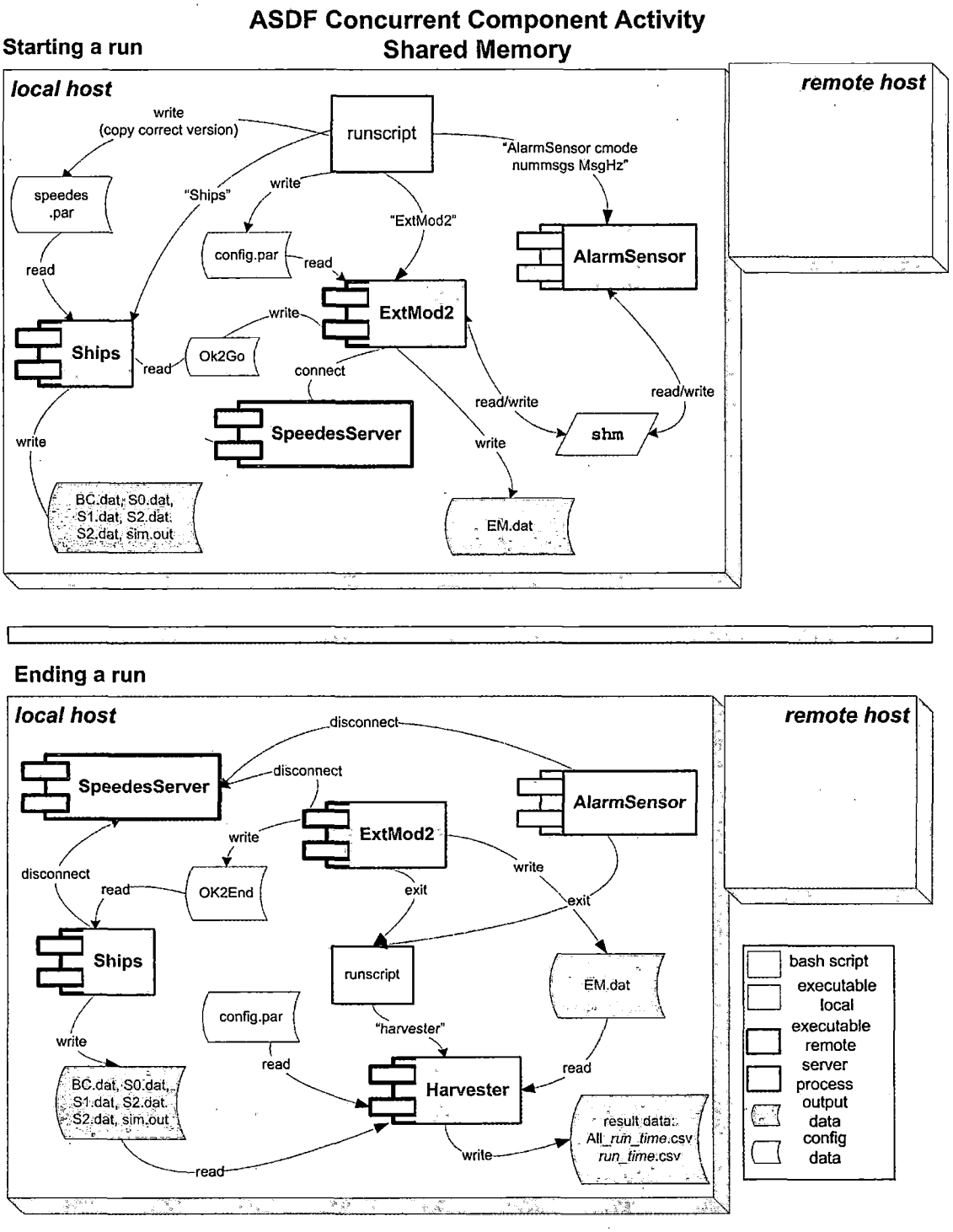


Figure A 11 Concurrent Component Activity Mode S

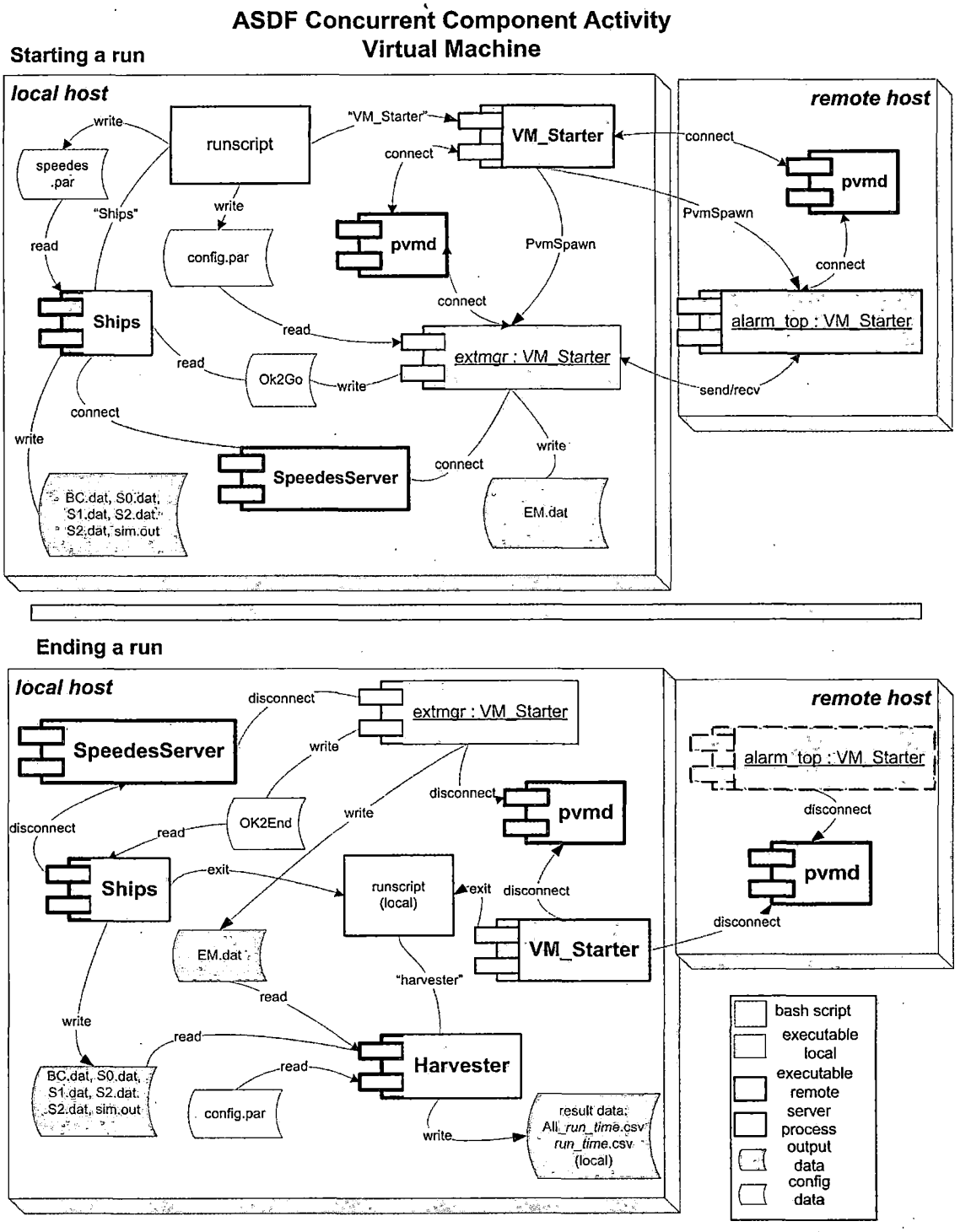


Figure A 12 Concurrent Component Activity Mode V

A7. Sequence Diagrams

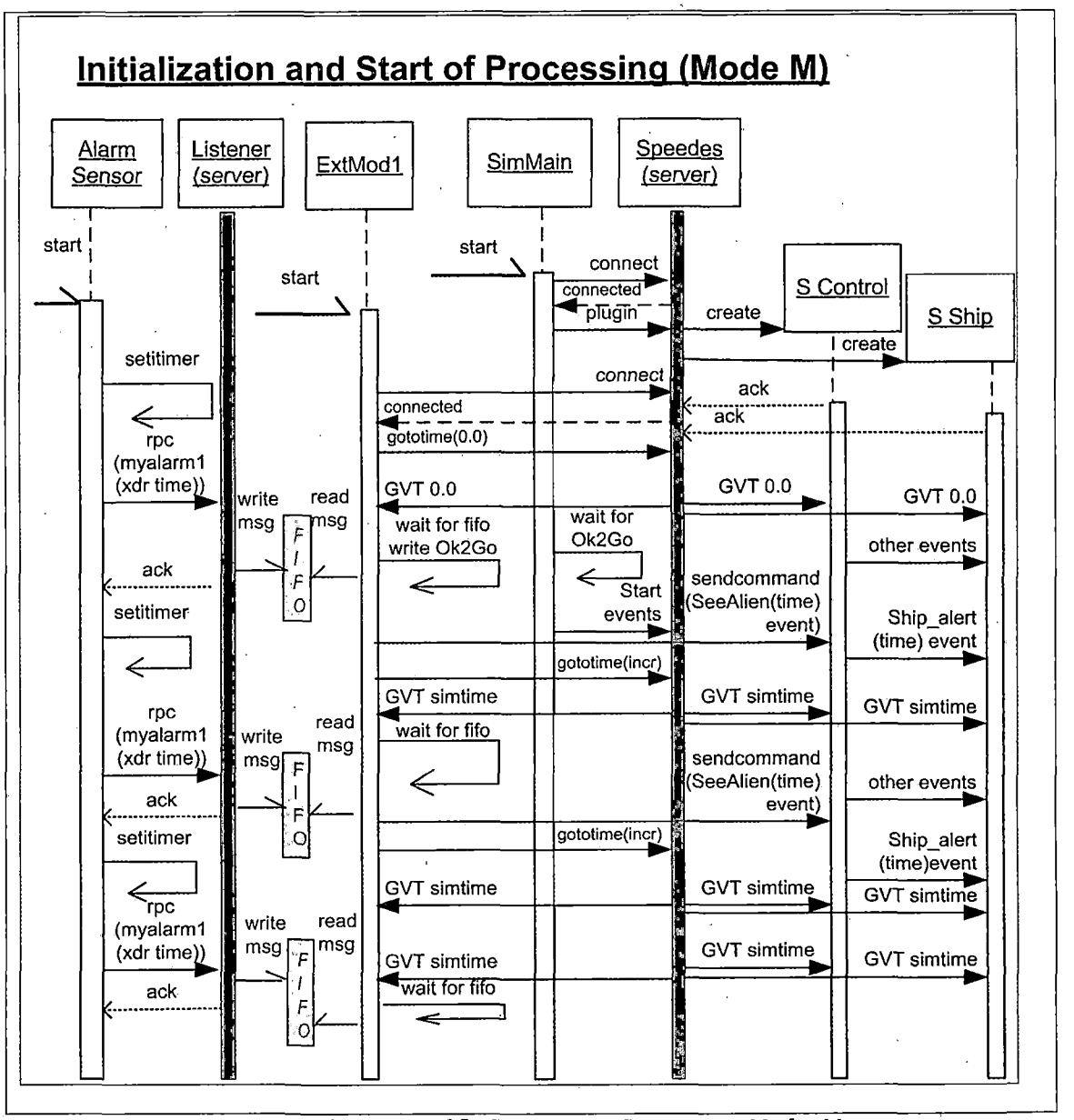


Figure A 13 Start-up Sequence Mode M

SpeedesServer and Listener must already be running before ExtMod and AlarmSensor are started by the script.

After connecting via SpeedesServer, ExtMod waits for SpeedesServer to declare everyone is at Time 0, then blocks till there is message data in the predefined FIFO, or till timeout is reached. After getting the first message it writes the Ok2Go file as a flag to the simulation. When there is data, ExtMod repackages it and sends it into the simulation, then blocks till simulation time (Global Virtual Time) meets or exceeds its next increment. The alternating block continues till SpeedesServer ends or the maximum message number is reached.

The bottom read of the FIFO shown in the diagram illustrates that a component of the latency is the time after the message is written when the ExtMod is waiting on SPEEDES and has not gotten back to look for new messages.

SimMain starts the Speedes framework and connects the Control and Ship SimObjects. They have other regularly scheduled events and processing triggered by simulation time updates in Control that are not detailed here. These may cause GVT updates of no interest to the ExtMod. The simulation does not start event processing till it sees the Ok2Go file.

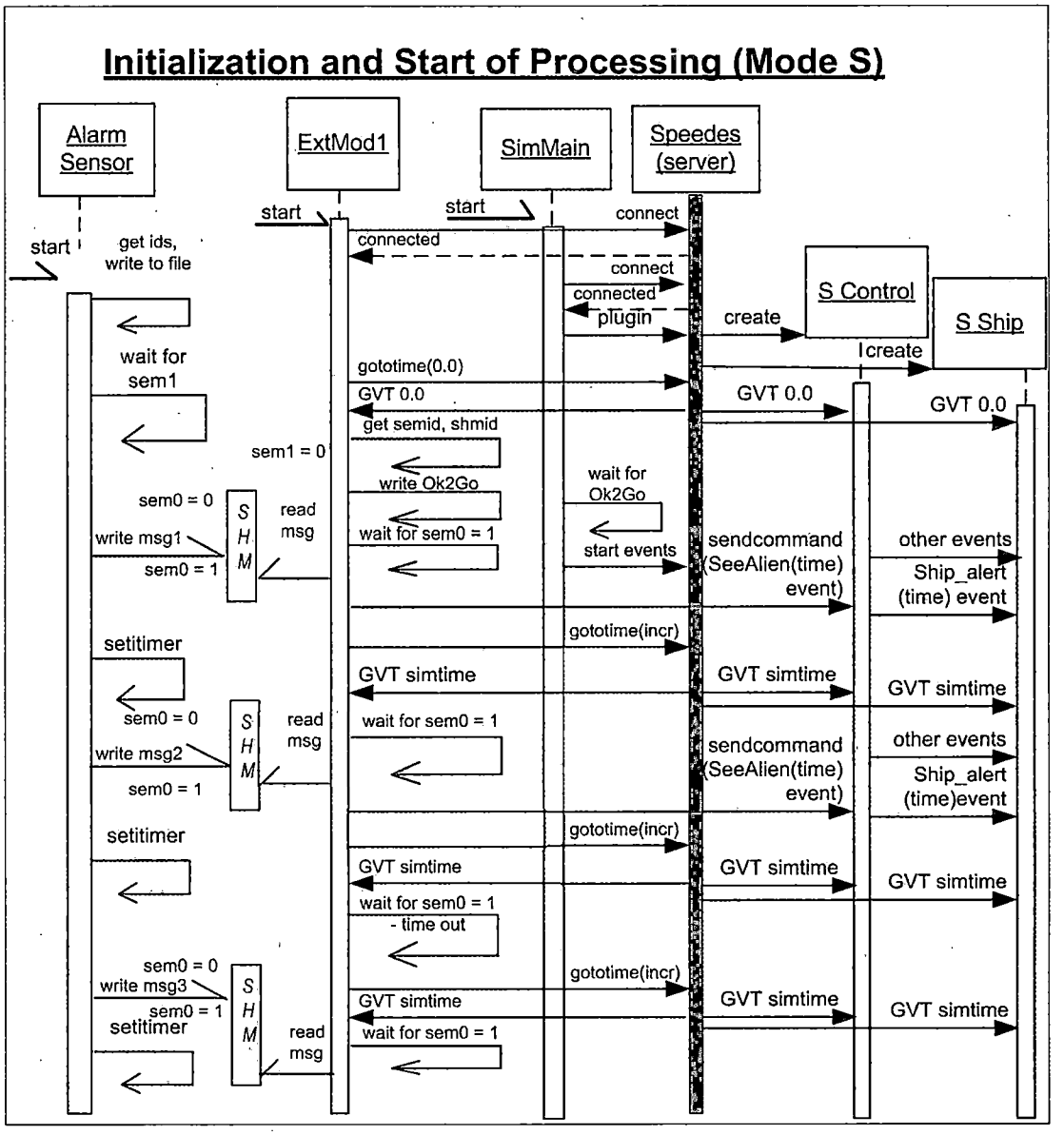


Figure A 14 Sequence Diagram Mode S

The SPEEDES side of the initialization is the same for shared memory as it is for the message-passing/FIFO case. However, there is no Listener server as the Alarm

Sensor process is on the same host. Access to the shared memory area is controlled by semaphores.

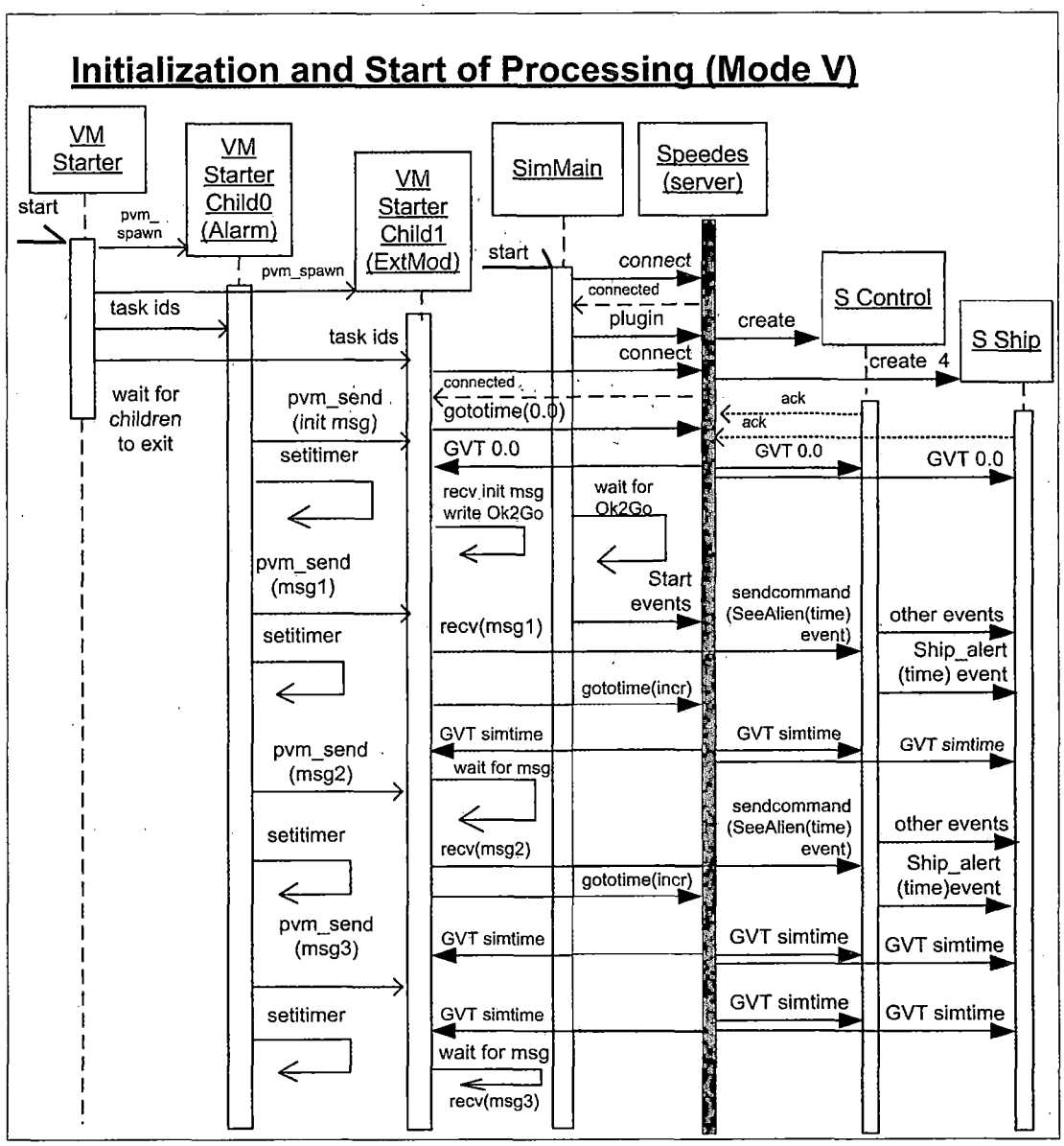


Figure A 15 Sequence Diagram Mode V

In the virtual machine configuration the parent process, VM_Starter, spawns two child processes, which

execute the same executable on two different machines. They are given task identification (id) information by the parent, which then just waits until both children have completed. Based on the task id, one child then takes on the role of the Alarm Sensor executable in the prior cases, and the other performs the ExtMod function. Communication is by explicit "PVM Send"/ "PVM recv" actions. The SPEEDES side is unchanged.

APPENDIX B

DATA ANALYSIS

SUBSECTIONS OF APPENDIX B: DATA ANALYSIS

B1. Time Skew

Table B 1 Output of one run of checktime script

Time at raven4

Fri Mar 24 12:12:34 PST 2006

Time at raven5

Fri Mar 24 12:12:34 PST 2006

Table B 2 Output of "ntpq -p

NTP JITTER MEANS RAVEN4 AND RAVEN5 COULD BE 0.173 SEC APART

ntpq -p at raven

remote	refid	st	t	when	poll	reach	delay	offset	jitter
*bigben.ucsd.edu	.GPS.	1	u	831	1024	133	18.129	-0.469	2.625
=+dewey.lib.ci.ph	clock.via.net	2	u	962	1024	377	43.407	9.908	0.43
ntp1.tummy.com									
nist1.symmetric		2	u	22	1024	377	44.602	-0.289	1.227

ntpq -p at raven4

remote	refid	st	t	when	poll	reach	delay	offset	jitter
*raven	bigben.ucsd.edu	2	u	21	128	377	0.149	-0.023	0.008

ntpq -p at raven5

remote	refid	st	t	when	poll	reach	delay	offset	jitter
*raven	bigben.ucsd.edu	2	u	934	1024	377	0.145	-2.265	0.165

Time skew accounts for an unknown fraction of the time measured in output variable delay1 (time message received by External Module Manager less time AlarmSensor reported for the send). The lower bound on the amount of skew is the minimum delay1 (-0.002612) and the maximum is the sum of the NTP jitter, 0.173.

B2. Analysis Planning

Table B 3 Combinations of Factors

Conditions in command line order, with states:	Number of runs		Analysis Case number	complement
comm mode (cmode)	3			
m = message passing/fifo			13	14 & 15 [#16]
s = shared memory			14	12 & 15 [#17]
v = virtual machine			15	12 & 13 [#18]
number of speedes nodes (nnodes)	3			
1 = simple internal sim comm			1	2 & 3 [#4]
2 = minimum socket sim comm			2	1 & 3 [#5]
5 = heavy socket sim comm			3	1 & 2 [#6]
time management risk (nrisk)	3			
L = Breathing Time Buckets			7	8 & 9 [#10]
M = Breathing Time Warp			8	7 & 9 [#11]
H = Time Warp			9	1 & 8 [#12]
message Hertz rate (msgHz)	3			
1 Hz			19	20 & 21 [#22]
2 Hz			20	19 & 21 [#23]
10 Hz			21	19 & 20 [#24]
Number of test cases in set	81			
runs	81			
data points/run	* 30			
rows	2430			

B3. Data Dictionary

Table B 4 Extract From ASDF Data Dictionary

(Columns not shown: input/output, null value, count. Bold = used)

column	variable name	description	measured/ derived	type	discrete category	remarks
input						
A	testnum	run id + msg form a unique id		string		
B	**cmode**	communication mode		char	m, s, v	Comm Factor each state 1/3 of total
C	**nnodes**	number of speedes nodes		int	1, 2, 5	Sim Factor each state 1/3 of total
D	**nrisk**	time management risk code		char	L, M, H	Sim Factor each state 1/3 of total
E	**MsgHz**	rate msgs generated in Hz		int	1, 2, 10	Comm Factor each state 1/3 of total
F	physconf	machines used		int	54, 50	
output						
G	msg	message number	measured	int	30: 1..30	used to count records
H	**delay1**	time taken for delivery of message to EM	measured	double	n/a	If blank, msg was dropped. Can be < 0 due to clock skew
I	**iatAS**	AS time since last msg, always blank for 1st message in run	measured	double	n/a	nominal: % missing is 3.33%
J	**iatBC**	BC time since last msg recv at sim, always blank for 1st msg in run	measured	double	n/a	nominal: % missing is 3.33%
K	e	events	measured	int	positive	same for all msgs in test run
L	r	rollbacks	measured	int	non-neg	same for all msgs in run
M	**cpu**	number of cpu seconds used	measured	double	n/a	same for all msgs in test run
N	**wall**	number of wall clock seconds	measured	double	n/a	same for all msgs in test run

extract! column	variable name	description	measured/ derived	type	discrete category	remarks
derived:						
O	**r2e**	ratio rollbacks/ events. Measure of work performed	derived proportion	double	non-neg	same for all msgs in test run
P	**c2node**	cpu time / nodes	derived proportion	double	non-neg	same for all msgs in test run
Q	**msgok**	True if message delivery was complete (from IF formula)	derived	boolean	T, F	
derived by run						
Dropout !A	rcvd	proportion of messages completed in a run	derived proportion	float	non-neg	msgok grouped & averaged by test run
by run!O	**jitterAS**	max variation in time source msg sent. Measure of data process variability	derived	float	non-neg	same for all msgs in test run
by run!R	**jitterBS**	max variation in time msg got to sim. Measure of comm path variability	derived	float	non-neg	same for all msgs in test run
by run!T	**jitterDel**	max variation in time msg got to sim. Measure of comm path variability	derived	float	non-neg	same for all msgs in test run
by run!S	**GVTs**	num of GVT events/run, a measure of sim internal comm complexity	measured	int	non-neg	same for all msgs in test run

B4. Characteristics of the Data

Table B 5 Descriptive Statistics

DESCRIPTIVE STATISTICS ALL COMM MODES

name	delay1	iatAS	iatBC	e	r
min	-0.00261	0.09076	0.000106	588	748
max	330.146	50.0221	56.8939	850	48987
median	0.964487	1.00002	1.02976	844	7378
average	19.47354	1.696328	2.979132	785.3086	12110.77
stddev	49.78539	3.415066	5.519041	96.99197	13036.43
count	1878	1797	1797	2430	2430
jitter	310.6725	48.32577	53.91477	197.3086	36876.23

name	cpu	wall	r2e	c2node
min	0.18	7.23517	1.250836	0.18
max	732.66	419.724	57.63176	146.532
median	35.84	35.5553	8.865089	17.465
average	138.9947	92.59758	14.89955	31.59585
stddev	210.2059	111.5513	15.44051	40.53945
count	2430	2430	2430	2430
jitter	593.6653	327.1264	42.73222	114.9361

DESCRIPTIVE STATISTICS MODE M ONLY

name	delay1	iatAS	iatBC	e	r
min	0.003278	0.099996	0.00011	719	1473
max	51.3981	50.0221	56.8939	850	45328
median	0.516395	1.02003	1.0202	845	7378
average	2.548115	3.014328	3.071571	834.2963	15082.81
stddev	5.051672	4.763277	5.6636	31.8442	14940.46
count	774	747	747	810	810
jitter	48.84999	47.00777	53.82233	115.2963	30245.19

name	cpu	wall	r2e	c2node
min	0.41	20.9268	2.048679	0.41
max	732.66	417.511	53.38987	146.532
median	37.23	36.7157	8.740521	18.615
average	159.2744	106.0487	18.0617	36.02007
stddev	221.3284	116.985	17.97673	42.29767
count	810	810	810	810
jitter	573.3856	311.4623	35.32817	110.5119

DESCRIPTIVE STATISTICS MODE S ONLY

name	delay1	iatAS	iatBC	e	r
min	0.002903	0.099994	0.000147	588	748
max	34.6087	23.7937	43.6249	846	16704
median	0.082415	1.00004	1.0301	616	6401
average	0.621838	1.419901	1.843184	675	6332.519
stddev	2.847331	2.403643	4.34448	94.21565	4577.441
count	294	267	267	810	810
jitter	33.98686	22.3738	41.78172	171	10371.48

name	cpu	wall	r2e	c2node
min	0.18	7.23517	1.250836	0.18
max	593.55	310.732	26.30551	118.71
median	26.77	32.3875	8.291451	13.16
average	93.21222	61.1137	9.048376	21.42159
stddev	175.612	86.5962	6.497783	34.31112
count	810	810	810	810
jitter	500.3378	249.6183	17.25714	97.28841

DESCRIPTIVE STATISTICS MODE V ONLY

name	delay1	iatAS	iatBC	e	r
min	-0.00261	0.09076	0.000106	844	1904
max	330.146	1.00017	56.6177	850	48987
median	7.99678	0.500011	1.02991	846	7861
average	42.48919	0.533186	3.278298	846.6296	14916.96
stddev	69.2081	0.368417	5.691926	2.058744	14694.82
count	810	783	783	810	810
jitter	287.6568	0.466984	53.3394	3.37037	34070.04

name	cpu	wall	r2e	c2node
min	0.6	19.991	2.253254	0.6
max	720.22	419.724	57.63176	144.044
median	37.21	36.7129	9.291962	18.605
average	164.4974	110.6304	17.58856	37.34589
stddev	222.8933	121.1538	17.28882	42.56422
count	810	810	810	810
jitter	555.7226	309.0936	40.0432	106.6981

B5. Descriptive Statistics for Data at Run-level

Table B 6 Global Virtual Time Descriptive Statistics

Sum of GVTs	
cmode	Total
m	412
s	291
v	412
Grand Total	1115

GVT Basic statistics	
max	56
min	2
range	54
bin est	1.8
bin size	2

Table B 7 Summary of Messages Received Grouped by Run

min	1
max	30
median	30
average	23.185185
stddev	10.919147
count	81

B6. Characteristics of the Data - Histograms

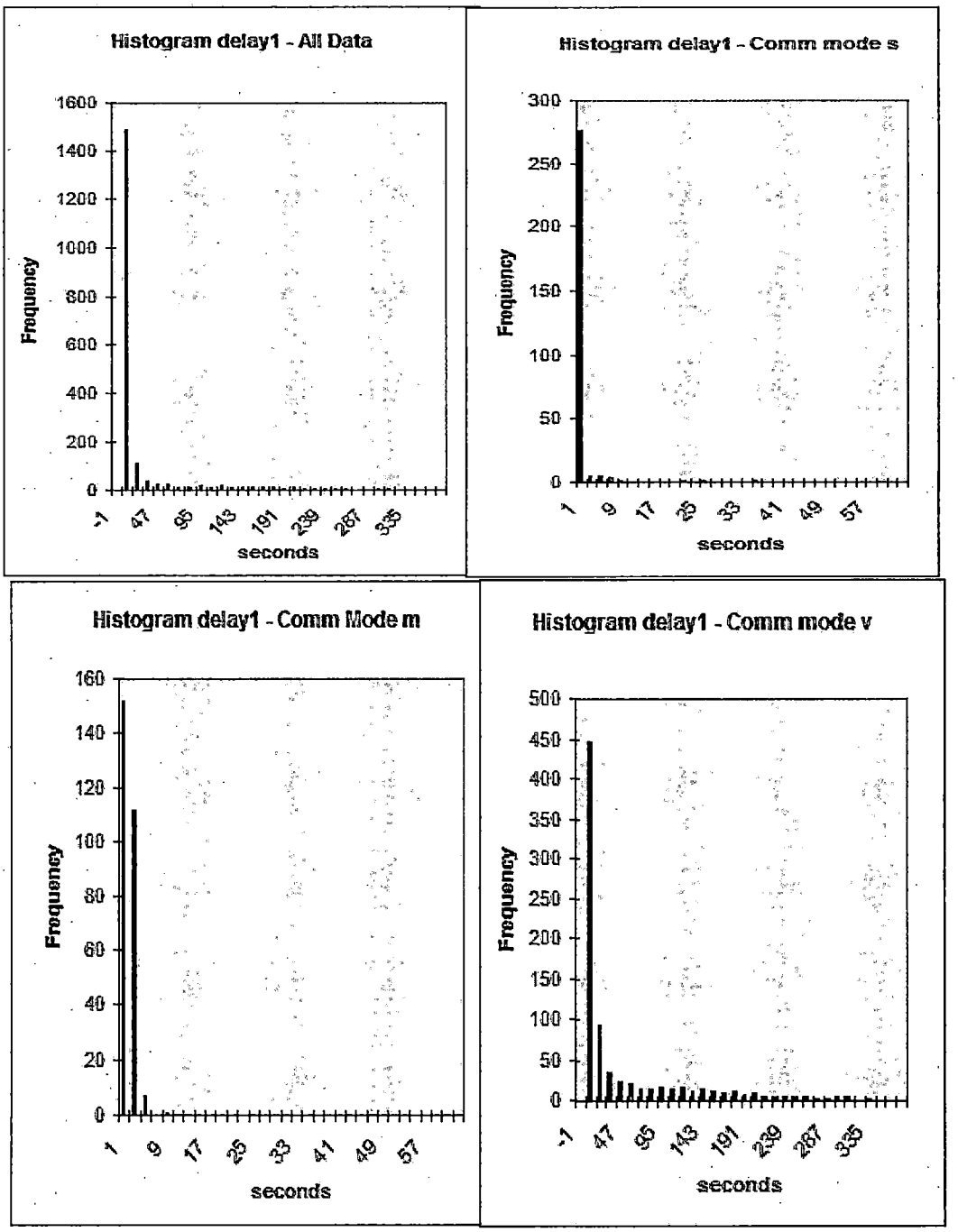


Figure B 1 Distributions for delay1

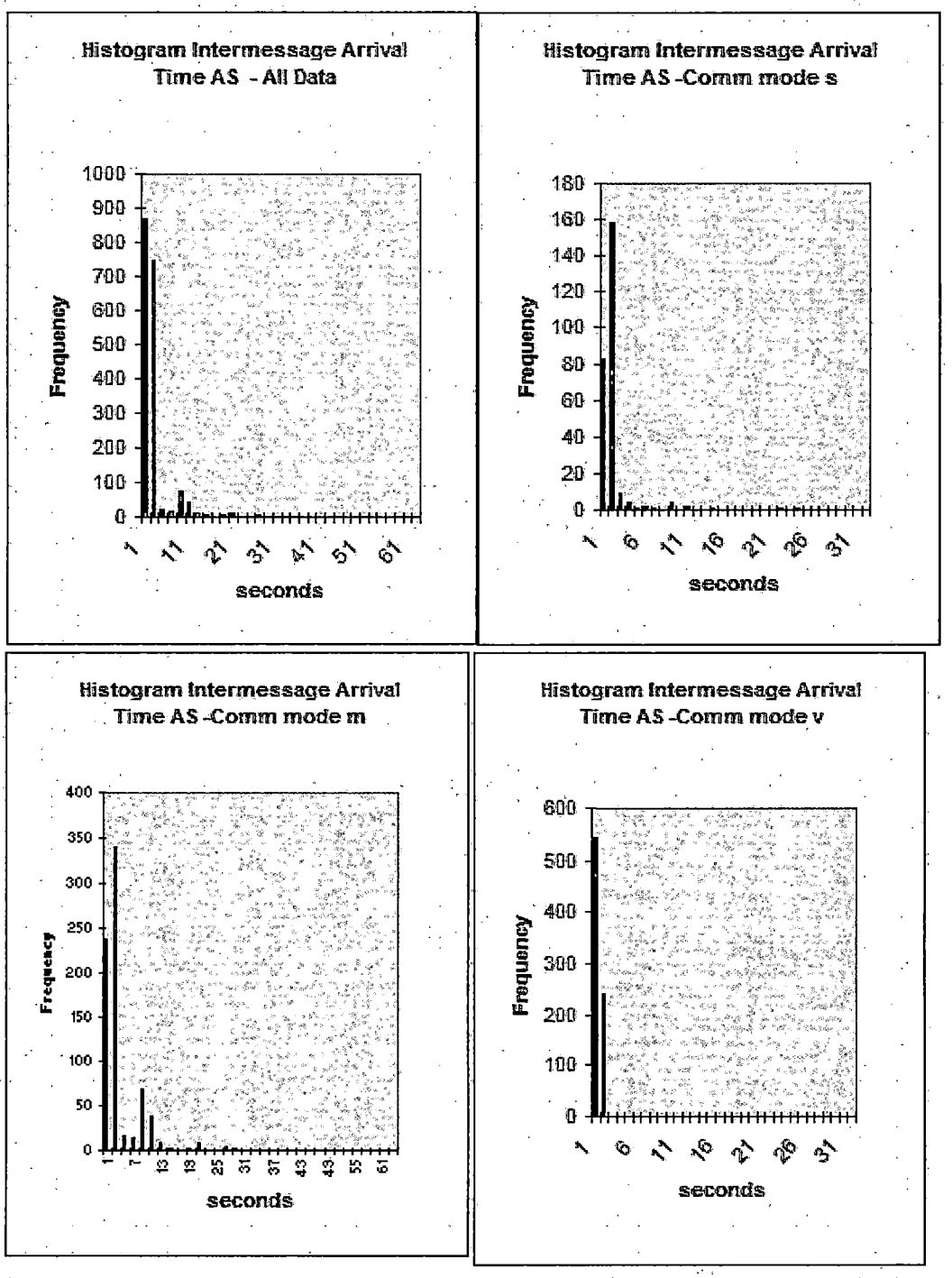


Figure B 2 Distributions for iatAS

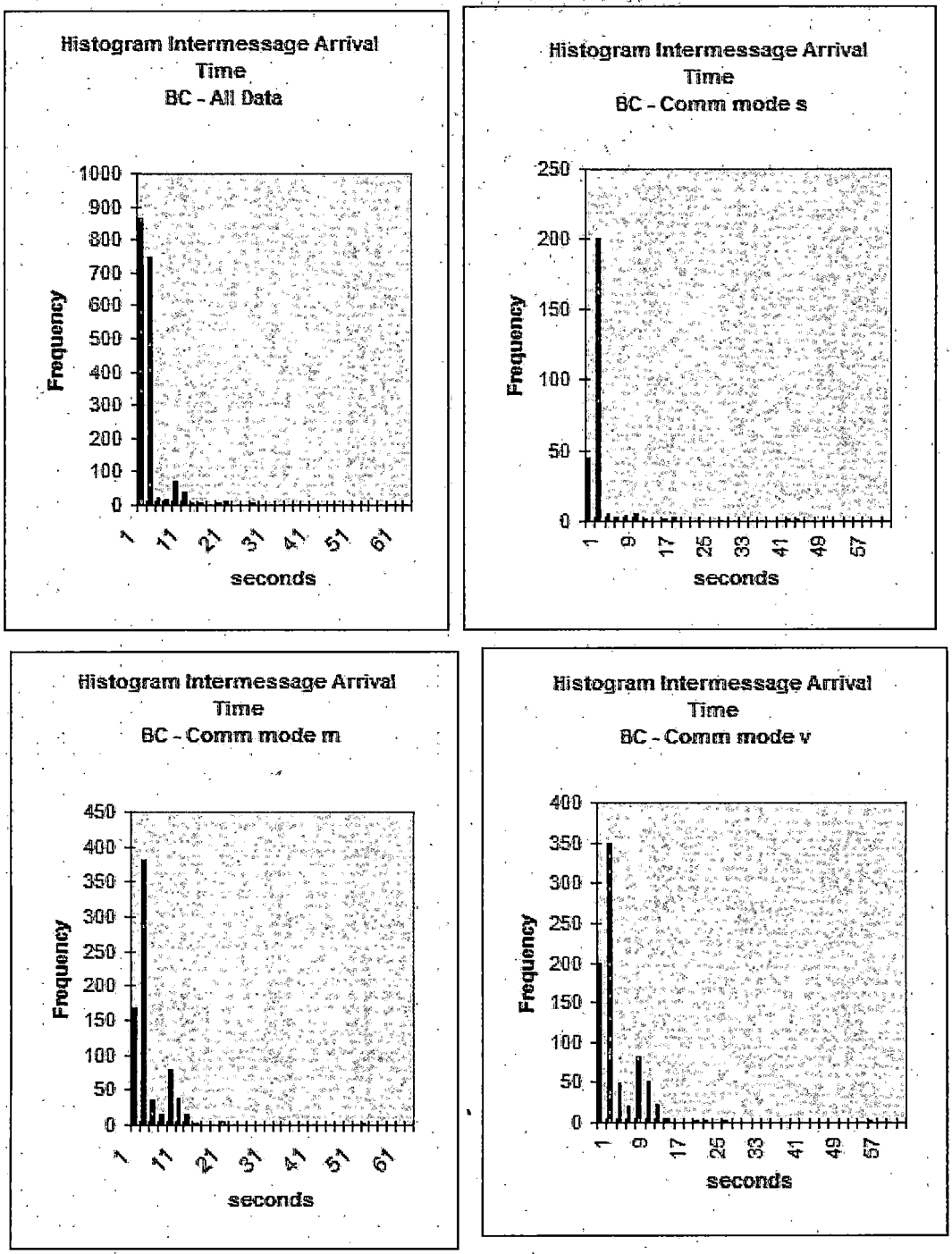


Figure B 3 Distributions for iatBC

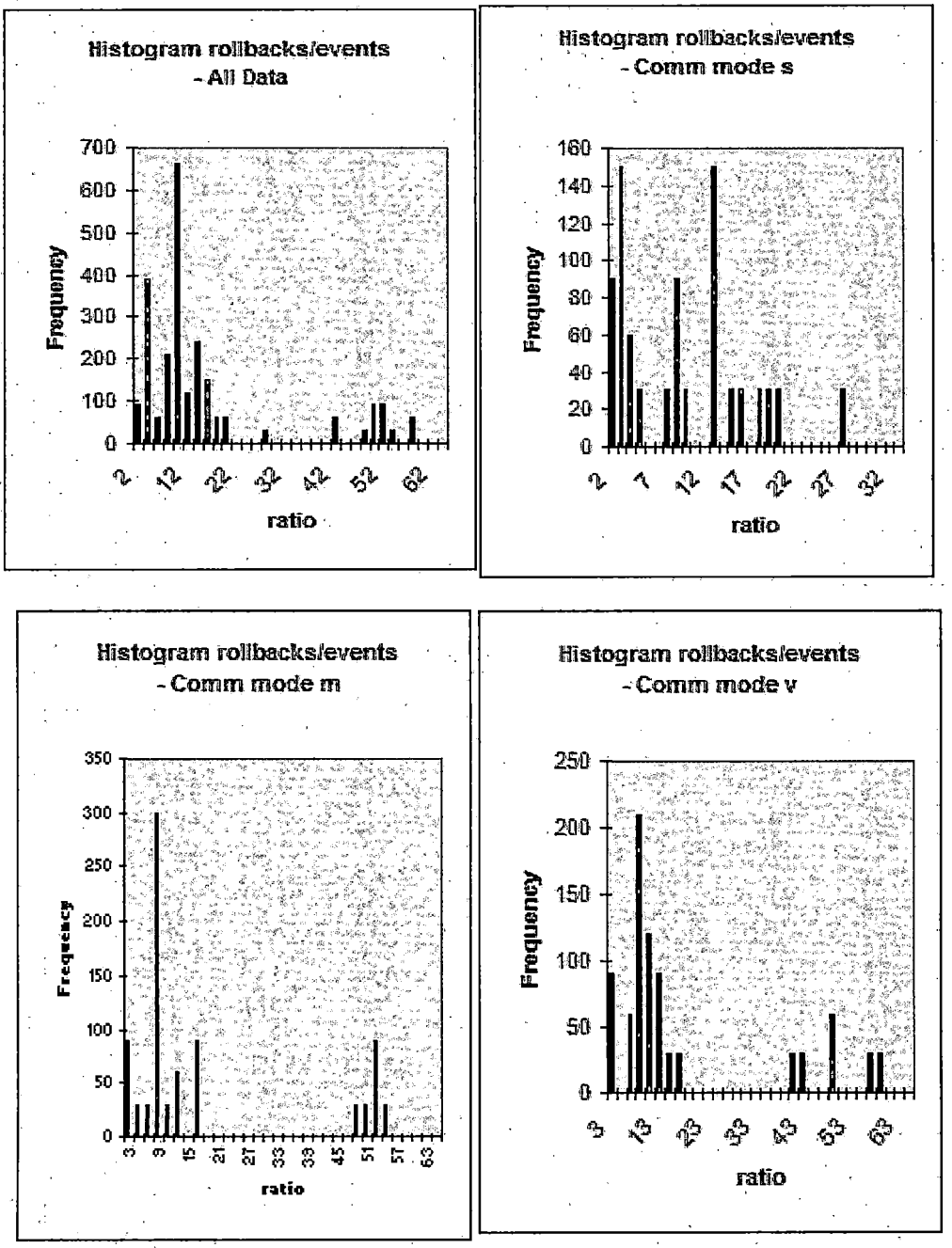
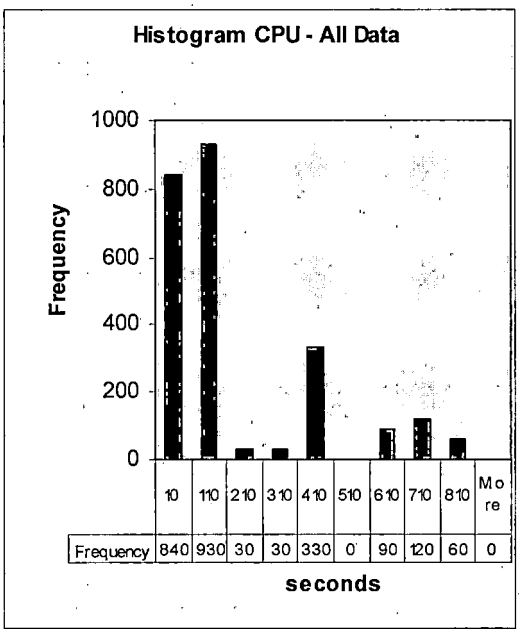
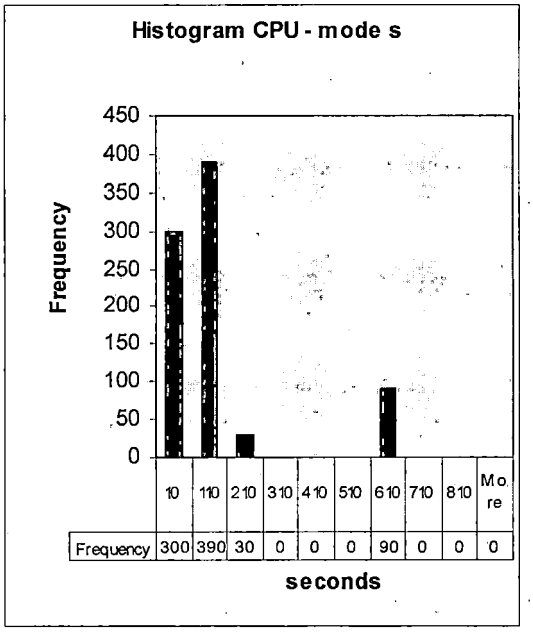
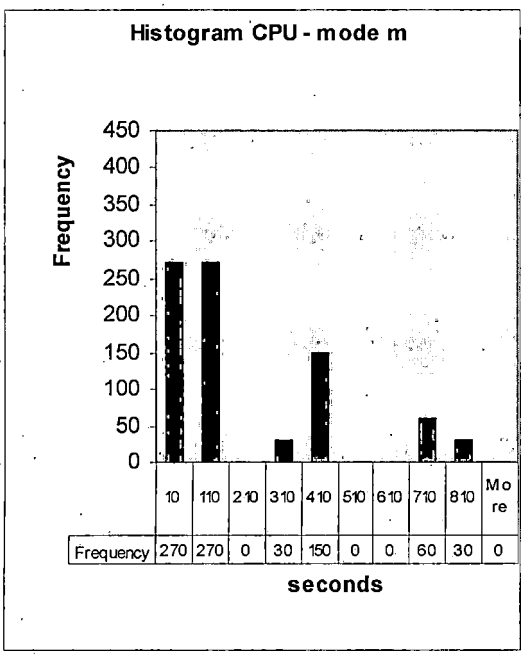


Figure B 4 Distributions for r2e







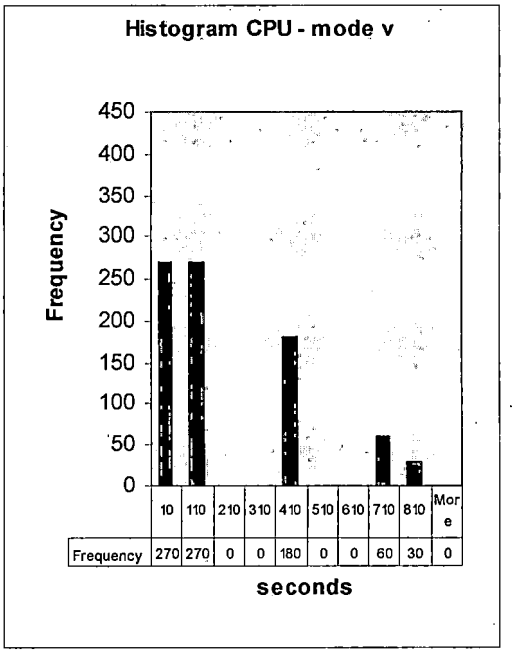


Figure B 5 Distributions for cpu

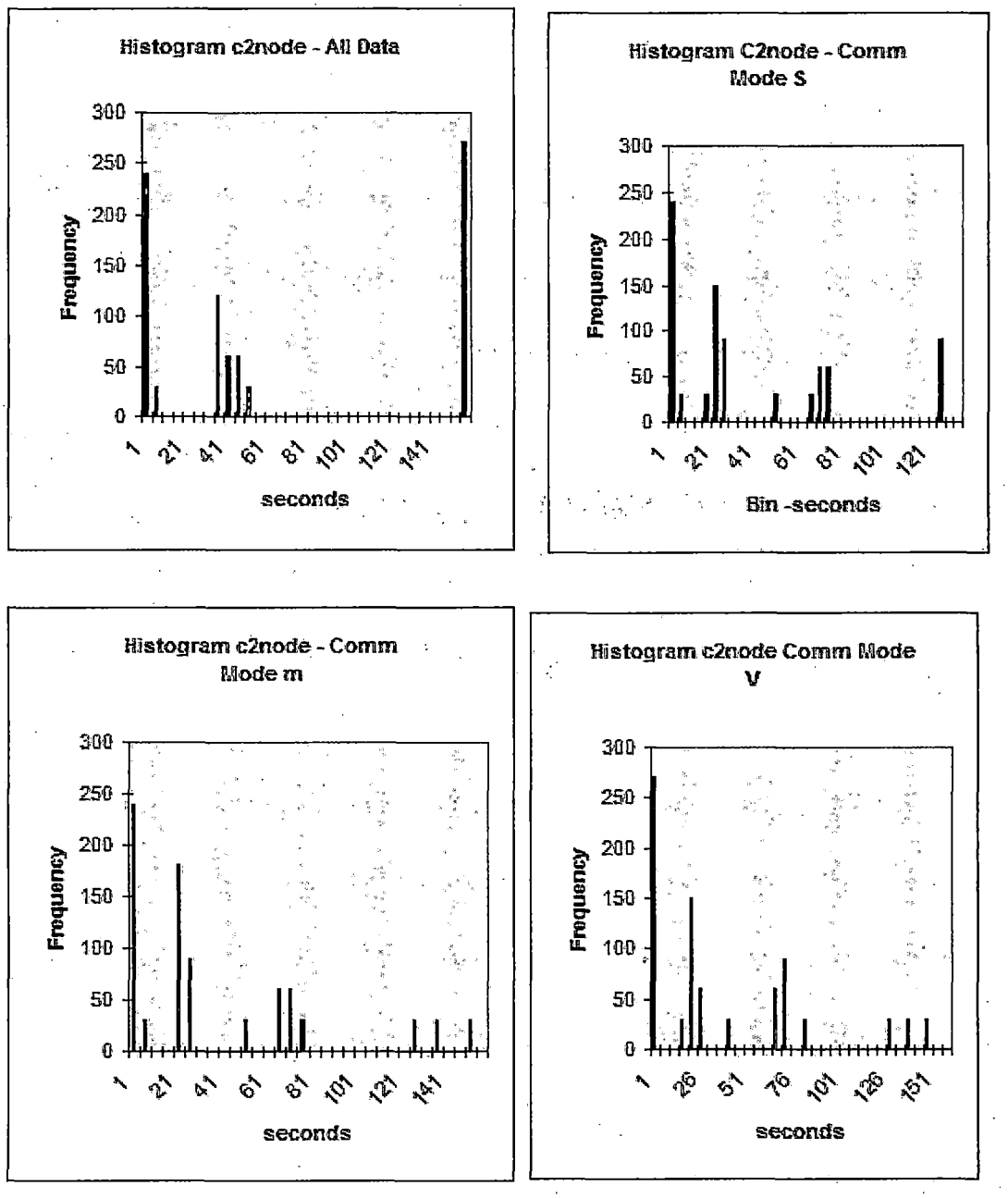


Figure B 6 Distributions for c2node

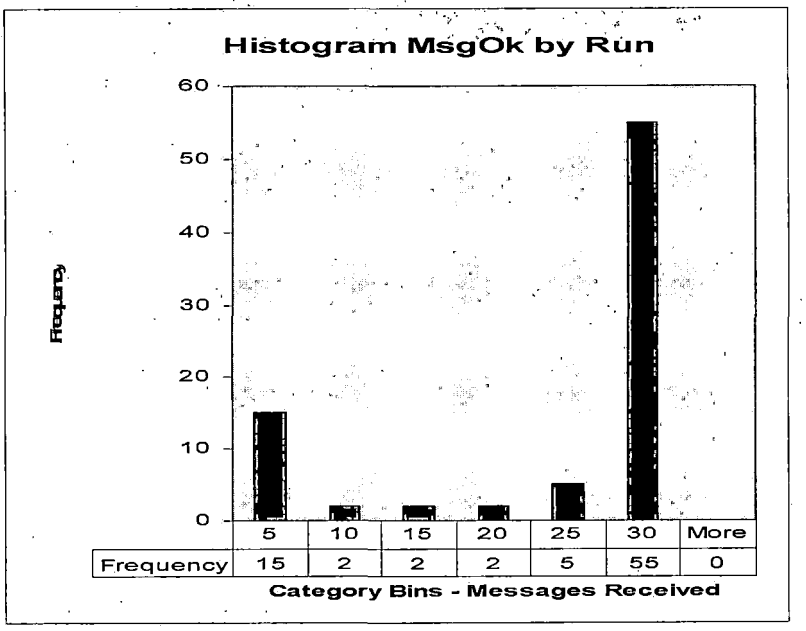


Figure B 7 Distribution for MsgOk

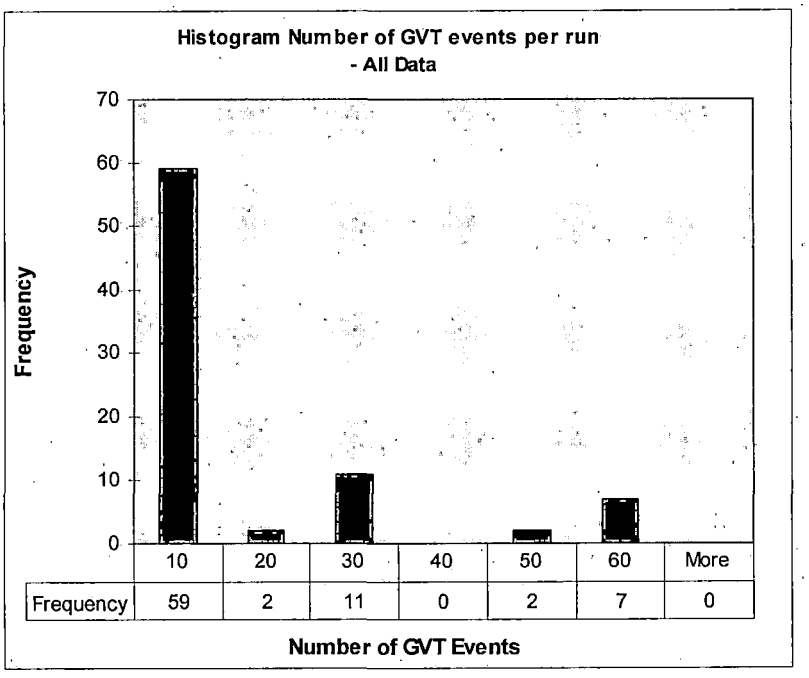


Figure B 8 Distribution for GVT Events

B7. The Main Effect of Each Factor on Message Received

Table B 8 Main effect calculations

CASE 0

nnode	(All)
nrisk	(All)
cmode	(All)
msgHz	(All)

Data	Total
Sum of rcvd	1878
Average of rcvd	23.19
StdDev of rcvd	10.92
Var of rcvd	119.23

CASE 1-6	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nnode	**1**	**2**	**5**	**1 & 2**	**1 & 5**	**2 & 5**
nrisk	(All)	(All)	(All)			
cmode	(All)	(All)	(All)			
msgHz	(All)	(All)	(All)			
Sum of rcvd	708	638	532	673	620	585
Average of rcvd	26.22	23.63	19.70	24.93	22.96	21.67
StdDev of rcvd	8.40	10.84	12.51			
Var of rcvd	70.56	117.47	156.45			

	EFFECT A **1 NODE**	EFFECT B **2 NODE**	EFFECT C **5 NODE**
Sum of rcvd	123	18	-141
Average of rcvd	4.56	0.67	-5.22

CASE 7-12	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nnode	(All)	(All)	(All)			
nrisk	**L**	**M**	**H**	**L & M**	**L & H**	**M & H**
cmode	(All)	(All)	(All)			
msgHz	(All)	(All)	(All)			
Sum of rcvd	587	641	650	614	618.5	645.5
Average of rcvd	21.74	23.74	24.07	22.74	22.91	23.91
StdDev of rcvd	12.01	10.60	10.35			
Var of rcvd	144.20	112.28	107.07			

	EFFECT A **L**	EFFECT B **M**	EFFECT C **H**
Sum of rcvd	-58.5	22.5	36
Average of rcvd	-2.17	0.83	1.33

CASE 13 - 18	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nnode	(All)	(All)	(All)			
nrisk	(All)	(All)	(All)			
cmode	**m**	**s**	**v**	**m & s**	**m & v**	**s & v**
msgHz	(All)	(All)	(All)			

Sum of rcvd	774	294	810	534	792	552
Average of rcvd	28.67	10.89	30.00	19.78	29.33	20.44
StdDev of rcvd	3.65	10.82	0.00			
Var of rcvd	13.31	117.10	0.00			

	EFFECT A **m**	EFFECT B **s**	EFFECT C **v**
Sum of rcvd	222	-498	276
Average of rcvd	8.22	-18.44	10.22

CASE 19 - 24	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nnode	(All)	(All)	(All)			
nrisk	(All)	(All)	(All)			
cmode	(All)	(All)	(All)			
msgHz	**1**	**2**	**10**	**1 & 2**	**1 & 10**	**2 & 10**

Sum of rcvd	702	636	540	669	621	588
Average of rcvd	26.00	23.56	20.00	24.78	23.00	21.78
StdDev of rcvd	9.11	10.14	12.73			
Var of rcvd	83.00	102.79	162.15			

	EFFECT A **1**	EFFECT B **2**	EFFECT C **10**
Sum of rcvd	114	15	-129
Average of rcvd	4.22	0.56	-4.78

B8. Worksheet Used to Find Confidence Values

(Does not rely on data being normally distributed)

COMPARING 2 PROPORTIONS

Group 1		Group 2
[redacted]	# of Occurrences	[redacted]
	Sample Size	
77.28%	Estimated Percent	66.67%
	Observed Absolute Difference	10.62%
	Standardized Difference (Z)	6.014056816
Hypothesis Test (Equality of Proportions)		
	P-value	1.80937E-09
Conclude the proportions are NOT equal		T
Confidence Interval (Difference of 2 Proportions)		
[redacted]	Confidence Level	95.0%
	Lower Bound	0.069684081
	Upper Bound	0.142661598

Ref #1: Worksheet from "EXCELing in Basic Statistics" short course, Data Vision Statistical Consulting and Training, Jan 24-25, 2005, Edwards AFB, CA

Ref #2: Montgomery D.C. and Runger G.C. Hubele N.F. (2004). *Engineering Statistics*, pg 230-235, John Wiley and Sons, New York, NY.

formula for Z is C10/(SQRT((A4+C4)/(A5+C5)*(1-(A4+C4)/(A5+C5))*(1/A5+1/C5)))

formula for P-value is 2*(1-NORMSDIST(C11))

formula for Lower Bound Is C10-NORMSINV(1-(1-A16)/2)*(SQRT(A8*(1-A8)/A5+C8*(1-C8)/C5))
formula for Upper Bound is C10+NORMSINV(1-(1-A16)/2)*(SQRT(A8*(1-A8)/A5+C8*(1-C8)/C5))

Figure B 9 Worksheet Used to Find Confidence Values

B9. Confidence Intervals for Message Received

Table B 9 Confidence Intervals Message Received

	GR1	GR 2				
nnode	(All)	**1**	**2**	**5**	(All)	(All)	(All)
nrisk	(All)	(All)	(All)	(All)	**L**	**M**	**H**
cmode	(All)	(All)	(All)	(All)	(All)	(All)	(All)
msgHz	(All)	(All)	(All)	(All)	(All)	(All)	(All)

Sum of rcvd	Total	Total	Total	Total	Total	Total	Total
Total	1878	708	638	532	587	641	650
sample size	2430	810	810	810	810	810	810
est percent	77.28%	87.41%	78.77%	65.68%	72.47%	79.14%	80.25%
obs Abs diff		10.12%	1.48%	11.60%	4.81%	1.85%	2.96%
Z		6.2165	0.8766	6.5526	2.7819	1.0973	1.7637
P-Value		0.0000	0.3807	0.0000	0.0054	0.2725	0.0778
Differ signif		T	T	T	T	T	T
at 95% conf		95.00%	[redacted]	95.00%	95.00%	[redacted]	[redacted]
Lower bound		7.30%	0.05%	7.94%	1.32%	0.09%	0.10%
Upper bound		12.95%	2.92%	15.27%	8.31%	3.61%	5.83%

	GR1	GR 2					
nnode	(All)	(All)	(All)	(All)	(All)	(All)	(All)
nrisk	(All)	(All)	(All)	(All)	(All)	(All)	(All)
cmode	(All)	**m**	**s**	**v**	(All)	(All)	(All)
msgHz	(All)	(All)	(All)	(All)	**1**	**2**	**10**

Sum of rcvd	Total	Total	Total	Total	Total	Total	Total
Total	1878	774	294	810	702	636	540
sample size	2430	810	810	810	810	810	810
est percent	77.28%	95.56%	36.30%	100.00%	86.67%	78.52%	66.67%
obs Abs diff		18.27%	40.99%	22.72%	9.38%	1.23%	10.62%
Z		11.6848	21.4910	14.8925	5.7420	0.7298	6.0141
P-Value		0.0000	0.0000	0.0000	0.0000	0.4655	0.0000
Differ signif		T	T	T	T	T	T
at 95% conf		95.00%	95.00%	95.00%	95.00%	[redacted]	95.00%
Lower bound		16.08%	37.28%	21.05%	6.51%	0.02%	6.97%
Upper bound		20.46%	44.69%	24.38%	12.26%	2.44%	14.27%

Comparison of each single factor proportion (as Group 2) to overall sample proportion (always Group 1)

B10. Correlation of Time Values

The correlation between all three output parameters reflecting facets of simulation time is very high, so CPU time (cpu) is used as a proxy for the other two.

Table B 10 Correlations Between Time Values

	cpu	*wall*	*c2node*
cpu	1		
wall	0.996523	1	
c2node	0.992106	0.988721	1

B11. The Main Effect of Each Factor on CPU Time

Table B 11 Main effect calculation on cpu

CASE 1-6	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nrisk	(All)	(All)	(All)			
nnodes	**1**	**2**	**5**	**1 & 2**	**1 & 5**	**2 & 5**
msgHz	(All)	(All)	(All)			
cmode	(All)	(All)	(All)			
Sum of cpu	576.30	29218.20	307962.60	14897.25	154269.45	168590.40
Sum of wall	19502.45	29431.00	176078.68	24466.72	97790.57	102754.84
Sum of c2node	576.30	14609.10	61592.52	7592.70	31084.41	38100.81
Average of cpu	0.71	36.07	380.20	18.39	190.46	208.14
Average of wall	24.08	36.33	217.38	30.21	120.73	126.86
Average of c2node	0.71	18.04	76.04	9.37	38.38	47.04
StdDev of cpu	0.25	12.13	210.99			
StdDev of wall	7.65	11.25	117.12			
StdDev of c2node	0.25	6.07	42.20			
Var of cpu	0.06	147.17	44517.24			
Var of wall	58.51	126.57	13716.28			
Var of c2node	0.06	36.79	1780.69			

	EFFECT A **1 NODE**	EFFECT B **2 NODE**	EFFECT C **5 NODE**
Sum of cpu	-168014.10	-125051.25	293065.35
Sum of wall	-83252.39	-68359.57	151611.96
Sum of c2node	-37524.51	-16475.31	53999.82
Average of cpu	-207.42	-154.38	361.81
Average of wall	-102.78	-84.39	187.18
Average of c2node	-46.33	-20.34	66.67

CASE 7-12	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nrisk	**L**	**M**	**H**	**L & M**	**L & H**	**M & H**
nnodes	(All)	(All)	(All)			
msgHz	(All)	(All)	(All)			
cmode	(All)	(All)	(All)			
Sum of cpu	184147.50	73460.40	80149.20	128803.95	132148.35	76804.80
Sum of wall	113077.94	53544.03	58390.16	83310.98	85734.05	55967.09
Sum of c2node	40174.29	17469.48	19134.15	28821.89	29654.22	18301.82
Average of cpu	227.34	90.69	98.95	159.02	163.15	94.82
Average of wall	139.60	66.10	72.09	102.85	105.84	69.10

Average of c2node	49.60	21.57	23.62	35.58	36.61	22.59
StdDev of cpu	295.68	125.12	133.54			
StdDev of wall	156.44	66.79	71.35			
StdDev of c2node	56.78	23.89	25.51			
Var of cpu	87424.83	15654.01	17833.35			
Var of wall	24473.68	4460.85	5091.11			
Var of c2node	3224.24	570.73	650.61			

	EFFECT A	EFFECT B	EFFECT C
	L	**M**	**H**
Sum of cpu	107342.70	-58687.95	-48654.75
Sum of wall	57110.84	-32190.03	-24920.82
Sum of c2node	21872.48	-12184.74	-9687.74
Average of cpu	132.52	-72.45	-60.07
Average of wall	70.51	-39.74	-30.77
Average of c2node	27.00	-15.04	-11.96

CASE 13 -18	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nrisk	(All)	(All)	(All)			
nnodes	(All)	(All)	(All)			
msgHz	(All)	(All)	(All)			
cmode	**m**	**s**	**v**	**m & s**	**m & v**	**s & v**
Sum of cpu	129012.30	75501.90	133242.90	102257.10	131127.60	104372.40
Sum of wall	85899.43	49502.09	89610.60	67700.76	87755.02	69556.35
Sum of c2node	29176.26	17351.49	30250.17	23263.87	29713.21	23800.83
Average of cpu	159.27	93.21	164.50	126.24	161.89	128.85
Average of wall	106.05	61.11	110.63	83.58	108.34	85.87
Average of c2node	36.02	21.42	37.35	28.72	36.68	29.38
StdDev of cpu	221.33	175.61	222.89			
StdDev of wall	116.99	86.60	121.15			
StdDev of c2node	42.30	34.31	42.56			
Var of cpu	48986.25	30839.58	49681.41			
Var of wall	13685.50	7498.90	14678.23			
Var of c2node	1789.09	1177.25	1811.71			

CASE 13 -18 continued	EFFECT A	EFFECT B	EFFECT C
	m	**s**	**v**
Sum of cpu	24639.90	-55625.70	30985.80
Sum of wall	16343.08	-38252.92	21909.84
Sum of c2node	5375.43	-12361.73	6986.30
Average of cpu	30.42	-68.67	38.25
Average of wall	20.18	-47.23	27.05
Average of c2node	6.64	-15.26	8.63

CASE 19 - 24	Value A	Value B	Value C	A+B/2	A+C/2	B+C/2
nrisk	(All)	(All)	(All)			

nnodes	(All)	(All)	(All)			
msgHz	**1**	**2**	**10**	**1 & 2**	**1 & 10**	**2 & 10**
cmode	(All)	(All)	(All)			
Sum of cpu	112976.10	119341.20	105439.80	116158.65	109207.95	112390.50
Sum of wall	77991.46	79035.29	67985.37	78513.38	72988.42	73510.33
Sum of c2node	25778.76	27234.21	23764.95	26506.48	24771.85	25499.58
Average of cpu	139.48	147.33	130.17	143.41	134.82	138.75
Average of wall	96.29	97.57	83.93	96.93	90.11	90.75
Average of c2node	31.83	33.62	29.34	32.72	30.58	31.48
StdDev of cpu	203.02	222.04	204.95			
StdDev of wall	105.28	119.40	109.12			
StdDev of c2node	38.86	42.85	39.73			
Var of cpu	41215.82	49300.31	42004.90			
Var of wall	11084.09	14257.31	11906.84			
Var of c2node	1510.46	1836.43	1578.26			

	EFFECT A **1Hz**	EFFECT B **2 Hz**	EFFECT C **10 Hz**
Sum of cpu	585.60	10133.25	-10718.85
Sum of wall	4481.13	6046.88	-10528.01
Sum of c2node	279.18	2462.35	-2741.53
Average of cpu	0.72	12.51	-13.23
Average of wall	5.53	7.47	-13.00
Average of c2node	0.34	3.04	-3.38

APPENDIX C

SOURCE CODE

SUBSECTIONS OF APPENDIX C: SOURCE CODE

The major program units are included in the first section of the appendix for ease of reference. The remainder of the set of programs and artifacts (the rest of the code, scripts, utilities, configuration files, etc.) that comprise the ASDF system is included in the second section, organized by directory.

SOURCE CODE

C1. Listing of Most Important Source Code

Code used to create AlarmSensor executable

Code used to create ExtMod1 executable

Code used to create Harvester executable

Code used to create Ships executable

Code used to create VM_Starter executable

C2. Supporting Source Code Files

Code used to create AlarmSensor executable

```
============================================================
 MAJOR FILES: /home/choaglun/asdf/alarmsensor
============================================================
In This section
-rw-rw-r--    1 choaglun choaglun     7910 Mar  6 10:46 myalarm_main.cpp
-rw-rw-r--    1 choaglun choaglun     2138 Feb 25 05:29 myalarm_mgr.h
-rw-rw-r--    1 choaglun choaglun    13464 Mar  6 11:01 myalarm_mgr.cpp
-rw-rw-r--    1 choaglun choaglun    10584 Feb 25 17:30 listener.cpp
------------------------------------------------------------------------------


/*****************************************************************/
// Program Name: myalarm_main.cpp             for Sim Ver 12
// Executable: AlarmSensor
// Command: AlarmSensor recipient_hostname comm_mode max_msgs
//                      arraysze message_hZ
// Author: Cathy Hoaglund
//                                      Date Last Modified: 24 Feb 06
/*****************************************************************/
/* Description:
    The myalarm process uses signals and setitimer to wake itself up
    periodically and send out a message with a timetag and status
    variables.

    It reads command-line values to determine the communications
    mode, number of messages, data rate, and size of the message
    array to be sent.

   Classes & functions in myalarm application:
       Signals are handled by timer_mgr, which uses the setitimer
       interval timer to create a cycle with microsecond resolution.

       It uses Array_mgr to fill an array with a counter and time-
       tagged sim values, and then transmit the message.
*/
/*****************************************************************/

#include <ctype.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream.h>
#include <unistd.h>
#include <iostream>
#include <signal.h>
#include <sys/time.h>

#include "../utils/TimeUtil.h"
#include "myalarm.h"
#include "myalarm_mgr.h"

// outfile for debugging only
#define ASOUTFILE  "/home/choaglun/asdf/AS.out"

// --- Global variables - command line
char *host;                 // remote host
char CommType;              // comm mode (m, s)
int MaxMsgs;                // number of messages expected
int MsgHz;                  // rate of message generation

int Interval;               // in microseconds
int prior = -1;             // last message acknowledged
int done = 0;               // flag, 1 when no more msgs
int ArrSize = 4;             // number of array vals to send{4..500)

Array_mgr am;               // instantiate the Array Manager class
```

```
void timer_mgr(int sigval);

//=========================================================================
// main handles the command line argument (if any) and registers
// timer_mgr as the signal manager for SIGALARM. It then raises SIGALRM
// and then waits till timer_mgr says we are done.
//=========================================================================

int main (int argc, char *argv[])
{
   //open output file
   ofstream outfile;
   outfile.open(ASOUTFILE, ios::out);
   if(! outfile){
      perror("alarm main outfile open error");
      return(-1);
   }

   if (argc < 4) {
      cerr  << "error. Usage: "
            << "AlarmSensor hostname CommType MaxMsg MsgHz" << endl;
      outfile << "Usage error - argc = " << argc << endl;
      return (-1);
   }
   else {
      host     = argv[1];
      CommType = * argv[2];
      MaxMsgs  = atoi( argv[3]);
      MsgHz    = atoi(argv[4]);
   }

   if (MsgHz == 0) {
      cerr << "MsgHz error! " << endl;
      outfile << "MsgHz error! " << endl;
      return (-1);
   }
   Interval = 1000000/MsgHz;

   outfile << "Alarm_main configuration parameters: \n"
     << " host =" << host << " CommType = "  << CommType << " MaxMsgs = "
     << MaxMsgs << ", MsgHz = "<< MsgHz << ", Interval = "<<Interval << endl;
   //cout << "Alarm_main configuration parameters: \n"
   //    << " host = " << host << "CommType = " << CommType << " MaxMsgs = "
   //<< MaxMsgs << ", MsgHz = "<< MsgHz << ", Interval = "<<Interval << endl;

    // Record starting time
    long hh, mm, ss, uu;
    char timetag[16] = "00:00:00.000000";
    DoTimetag(&hh, &mm, &ss, &uu, timetag);
    outfile << timetag << " Alarm_main Starting initialization" << endl;
    cout    << timetag << " Alarm_main Starting initialization" << endl;

    outfile.close();

    //Array_mgrVM am() is a class to handle array detail & communication
    // Instantiated globally to be available to signal handler

    am.start_Array_mgr(host, CommType, ArrSize, Interval);

    signal(SIGALRM,timer_mgr);
    raise(SIGALRM);

    //Wait till signal processing is over
    while (!done);

    exit (0);
}

//=========================================================================
```

```
// timer_mgr: Handles the repetive alarm function by using the
//   ITIMER_REAL to raise SIGALRM for every Interval number of usecs.
//   It uses the Array_mgr class to generate data and take
//   care of transmitting it to the External Module process.
//=====================================================================

void timer_mgr(int sigval)
{
  static int simtime =0;
  static long int msg_num =0;
  FILE *fp;
  long h, m,s, u;
  char timetag[16]="00:00:00.000";

  struct itimerval t; //declare & initialize time structure
  t.it_interval.tv_usec = Interval;
  t.it_interval.tv_sec = 0;
  t.it_value.tv_usec = Interval;
  t.it_value.tv_sec = 0;

  int maxtime = MaxMsgs * Interval;
  int errstat = 0;

   //open output file
   ofstream outfile;
   outfile.open(ASOUTFILE, ios::app);
   if(! outfile){
      perror("timer_mgr outfile open error");
      return;
   }


  // *********Do init to allow Sim to get going, init comm *********
  if (simtime == 0) {
    errstat = am.init();

    if (errstat != 0) {
       done = 1;
       return;
    }

    // We have init success, start message processing
    errstat = am.send(++msg_num);

    DoTimetag(&h, &m, &s, &u, timetag);

    if (errstat != 0) {
       outfile << timetag<< "timer_mgr: Lost msg "<< msg_num<< endl;
       cout    << timetag<< "timer_mgr: Lost msg "<< msg_num<< endl;
    }


    // Start timer
    simtime = simtime + Interval;
    outfile << timetag << " timer_mgr, after msg " << msg_num
      << "simtime = " << simtime<<" Interval = "<< Interval << endl;

    static sigset_t sigmask;
    sigemptyset(&sigmask);
    if (sigaddset(&sigmask,SIGALRM)==-1 ||
        sigprocmask(SIG_SETMASK, &sigmask, 0)==-1)
      perror ("set signal mask");

    signal(SIGALRM, timer_mgr);
    setitimer(ITIMER_REAL, &t, 0);
  }


  //********* Normal interval processing till maxtime *************
  else if (simtime < maxtime) {
    errstat = am.send(++msg_num);
```

```
    DoTimetag(&h, &m, &s, &u, timetag);

    if (errstat != 0) {
       outfile << timetag<< "timer_mgr lost msg "<< msg_num <<endl;
       cout    << timetag<< "timer_mgr lost msg "<< msg_num <<endl;
    }

    simtime = simtime + Interval;

    static sigset_t sigmask;
    sigemptyset(&sigmask);
    if (sigaddset(&sigmask,SIGALRM)==-1 ||
         sigprocmask(SIG_SETMASK, &sigmask, 0)==-1)
      perror ("set signal mask");

    signal(SIGALRM, timer_mgr);
    setitimer(ITIMER_REAL, &t, 0);
  }

  //********* Simtime >= maxtime *************
  else {

     DoTimetag(&h, &m, &s, &u, timetag);
     outfile << timetag << " timer_mgr simtime > maxtime" <<endl;
     //cout    << timetag << " timer_mgr simtime > maxtime" <<endl;

     am.endmsg();      //release any resources

     DoTimetag(&h, &m, &s, &u, timetag);
     outfile << timetag << " ** END MESSAGE GENERATION **" <<endl;
     cout    << timetag << " ** END MESSAGE GENERATION **" <<endl;

    t.it_interval.tv_usec = 0;
    t.it_value.tv_usec = 0;
    setitimer(ITIMER_REAL, &t, 0);
    done = 1;
  }

  outfile.close();

  return;
}
```

```
// ********************************************************************
// Name: myalarm_mgr.h                                  for Sim version 12
//
// Author: Cathy Hoaglund                    Last modified: 24 Feb 06
//
// ********************************************************************

#ifndef MYALARMMGR_H
#define MYALARMMGR_H

#include <fstream.h>

//=====================================================================
// Array_mgr
//=====================================================================
class Array_mgr
{
   public:
                Array_mgr();
                ~Array_mgr();
        void    start_Array_mgr(char *inHost, char inComm,
                                int inArr_size,int inMsgIntv);
                               //set data members per config

        void    fill_array();          //timetag and fill
```

```
      int       init();                  //choose correct init
      int       send(long msg_num);      //choose correct send
      int       endmsg();                //choose correct end
      int       myalarm_1(long msg_num); //does rpc init & send
      int       init_shm();
      int       send_shm(long msg_num);
      void      end_shm();

   private:
      ofstream  outfile;          //diagnostics & status
      int       interval;         //microsecs between messages
      char      ttag[16];         //timetag
      long       h, m, s, u;      //hrs, mins, secs, microsecs.


    //sim-specific
      char      CommType;         //m = Message Passing via rpc & fifo
                                  //s = Single system Shared Memory

      long       arr[500];
      int       arr_size;        //how many elements to use [4..500]
                                  //determines packet size in mode m

      long      msg;              // arr[0], msg_num
      long      secs;             // arr[1], message time pt 1
      long      usecs;            // arr[2], message time pt 2
      long      alien;            // arr[3], alien detection T/F

    //rpc variables
      char      *thehost;         //remote destination
      CLIENT    *clnt;
      values    *result_1;
      values     myalarm_repl_1_arg;

    // shm variables
      int      semid;             //semaphore id
      int      shmid;             //shared memory id
      char*    shmaddr;           //shared memory address
      long     *buffer;           //shared memory value buffer
};

#endif
```

```
/*****************************************************************/
// Program Name: myalarm_mgr.cpp            for Sim Ver 12
// Executable: AlarmClock
// Class: Array_Mgr
// Author: Cathy Hoaglund
//        partly based on a program by Jeff Zamora & Cathy Hoaglund
//
//                                    Date Last Modified: 24 Feb 06
/*****************************************************************/
/* Description:
    The alarm process myalarm_main uses the Array_Mgr class to fill
    and transmit a message array. The communication method used is
    chosen based on the config.par file:
      m = it uses an rpc client method,myalarm_1, plus a FIFO
      s = it uses semaphores & shared memory,
      v = it uses distributed shared memory, [implemented in a
           different executable - see virtual_mach dir

    This file is one of a set of units that implement a RPC-based
    client-server application for the message passing mode.

    The related RPC files are:
    myalarm.x - original input to rpcgen
    (note: rpcgen generated files must be changed from .c to .cpp)
```

```
    myalarm_mgr.cpp - client code with programmer defined
        functionality. This has the driver for the client side.
        Template from rpcgen modified for C++
    myalarm_clnt.c - the rpcgen-generated file with client
        support code and structure to convert array to and from
        XDR format and handle communication
    listener.cpp - server code with programmer defined
        functionality to accept an array of long integers and
        change the first to errstat as an ACK.
        Template from rpcgen modified for C++
    myalarm_svc.c - the rpcgen-generated file with the driver for
        the server side. Includes support code and structures to
        convert array to and from XDR format, and to handle the
        communication functions, creating and registering the
        RPC socket.
    myalarm_xdr.c - contains 1 function that implements the RPC
        filter that actually converts the array to XDR


   Classes & functions in AlarmSensor application:
       Signals are handled by timer_mgr, which uses the setitimer
       interval timer to create a cycle with microsecond resolution.

       Class Array_mgr member functions:
         fill_array() fills an array with counter,time, and data values
        init() calls init_rpc(), init_shm(),
        send() calls send_rpc(), send_shm(),
        endmsg() calls end_shm(),
*/

/*****************************************************************/

#include <ctype.h>
#include <fcntl.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#include <sys/time.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#include "myalarm.h"
#include "myalarm_mgr.h"
#include "../utils/TimeUtil.h"
#include "../utils/SemUtil.h"

#define ASOUTFILE  "/home/choaglun/asdf/AS.out"

//=====================================================================
// Constructor & Destructor
//=====================================================================

Array_mgr::Array_mgr()
{
   h = m = s = u = 0;
   for (int i = 0; i < 500; i++)  arr[i] = 0;
   arr_size = 0;
   msg = secs = usecs = alien = 0;
   semid = shmid = 0;
}

Array_mgr::~Array_mgr()
{
   outfile.close();
}
```

```
//=====================================================================
// Array_mgr:: start_Array_mgr
//     initializes data members associated with config files,
//     opens diagnostic log file
//=====================================================================

void Array_mgr::start_Array_mgr(char *inHost, char inComm, int inArr_size,
                        int inMsgIntv)
{
   thehost = inHost;
   CommType = inComm;
   arr_size = inArr_size;
   interval = inMsgIntv;

   outfile.open(ASOUTFILE, ios::app);
   if(! outfile) {
      perror("AM outfile open error");
   }

   outfile << TmTag(ttag) << " Start Array_mgr " << endl;
   outfile.close();
}
//=====================================================================
// Array_mgr::fill_array:
//     fills an array with id, time, and a state values, then
//     pads balance
//=====================================================================

void Array_mgr::fill_array()
{
   alien = msg % 2;
   TellTime(&h, &m, &s, &u);
   RawTime(&secs, &usecs);

   outfile.open(ASOUTFILE, ios::app);
   if(! outfile) {
      perror("AM fill_array outfile open error");
   }

   arr[0] = msg;
   arr[1] = secs;
   arr[2] = usecs;
   arr[3] = alien;
   if (msg == 0) {
      arr[3] = interval;  //needed to pass interval to Listener
   }
   for (int i = 4; i < 500; i++) arr[i] = i;
   outfile.close();

}

//=====================================================================
// Array_mgr::init
//  Chooses correct init actions based on Comm Type
//  returns 0 on success, -1 on error
//=====================================================================
int Array_mgr::init()
{
   int errstat;

   if(CommType == 'm') {
      errstat = myalarm_1(0);
   }
   else if (CommType == 's') {
      errstat = init_shm();
   }
   else {
      cerr << "AM init: unexpected type" << endl;
      errstat = -2;
   }
   return(errstat);
```

```
}


//=============================================================
// Array_mgr::send
//  Chooses correct send actions based on Comm Type
//  returns 0 on success, -1 on error
//=============================================================
int Array_mgr::send(long msg_num)
{
   int errstat;

   if(CommType == 'm') {
      errstat = myalarm_1(msg_num);
   }
   else if (CommType == 's') {
      errstat = send_shm(msg_num);
   }
   else {
      cerr << "AM send:unexpected type" << endl;
      errstat = -2;
   }
   return(errstat);
}


//=============================================================
// Array_mgr::endmsg
//  Chooses correct send actions based on Comm Type
//  returns 0 on success, -1 on error
//=============================================================
int Array_mgr::endmsg()
{
   int errstat = 0;

   if(CommType == 'm') {
      //do nothing
   }
   else if (CommType == 's') {
      if(shmid >= 0) {
         end_shm();
      }
   }
   else {
      cerr << "AM endmsg:unexpected type" << endl;
      errstat = -2;
   }
   return(errstat);
}


//=============================================================
// Array_mgr::myalarm_1:
//     RPC mode - After data member array arr is filled , copies
//     to XDR parameters values_len and values_val. Creates client
//     handle and uses it to call the remote procedure. On reply
//     or timeout then releases client resources.
//     Returns 0 = success, -1 = error
//=============================================================

int Array_mgr::myalarm_1(long int msg_num)
{
    bool     sent = FALSE;

    // Fill the array
    msg = msg_num;
    fill_array();

    // Copy to XDR transport variables
    myalarm_repl_1_arg.values_len = arr_size;
    myalarm_repl_1_arg.values_val = arr;
```

```
    // Create client handle
    clnt = clnt_create (thehost, MYALARM, MYALARM_VER, "tcp");
    if (clnt == NULL) {
        clnt_pcreateerror (thehost);
        exit (1);
    }

    // Get result back from remote procedure as ACK
    result_1 = myalarm_repl_1(&myalarm_repl_1_arg, clnt);
    if (result_1 == (values *) NULL) {
        clnt_perror (clnt, "call failed ");
        clnt_destroy (clnt);
        return(-1);
    }
    else {
        sent = TRUE;
    }

    // Release resources
    clnt_destroy (clnt);
    return(0);
}


//=====================================================================
// Array_mgr::init_shm
//    Create semaphore, allocate shared memory, write ids to file,
//    then wait for ExtMod to indicate readiness via semaphore
//  returns 0 on success, -1 on error
//=====================================================================

int Array_mgr::init_shm()
{

   //outfile << TmTag(ttag) <<" *** AM init_shm *** "  << endl;

   // Get a semaphore set with unique id, set resource count to 1
   semid = start_sem(2);

   //outfile <<"  Initial state sem0: " <<test_sem(semid, 0)
   //     << " sem1: " << test_sem(semid, 1) << endl;

   // Allocate a shared memory segment
   shmid = shmget(IPC_PRIVATE, 4000, IPC_CREAT | SHM_R | SHM_W);
   if (shmid == -1){
      perror("AM init_shm: failure allocating shm ");
      return(-1);
   }

   // write semid, shmid to file for ExtMod to read
   ofstream  semfile;
   semfile.open("semid_file");
   if (! semfile)  {
      perror("Array_mgr cannot open semid_file");
      return(-1);
   }
   semfile << semid << endl << shmid << endl;
   semfile.close();
   outfile << TmTag(ttag)<< " semid_file written" << endl;

   //Attach shm
   shmaddr = (char*)shmat(shmid, 0, 0);

   // Query shm
   struct shmid_ds sbuf;
   if (0 == (shmctl(shmid, IPC_STAT, &sbuf))) {
     outfile << " shared memory id:" << shmid << endl;
     outfile << " shared memory size:" << sbuf.shm_segsz << endl;
     outfile << " shared mem num curr attach:" << sbuf.shm_nattch <<endl;
```

```
    }
    else {
       perror("shmctl err");
       return(-1);
    }
    buffer = &arr[0];

    //Wait for External Module to finish initialization
    int n = test_sem(semid,1);
    //cout  << "Waiting on Sem1 = " << n;
    while( n > 0) {
       n = test_sem(semid,1);
       sleep(1);
    }
    outfile << TmTag(ttag) << " AM init complete" << endl << endl;
    cout    <<       ttag << " AM init complete" << endl << endl;

    return(0);
}


//=====================================================================
// Array_mgr::send_shm()
//   Write message to shared memory
//=====================================================================

int Array_mgr::send_shm(long msg_num)
{

    //outfile << TmTag(ttag) << " *** AM send_shm: msg " << msg_num << endl;
    //cout << TmTag(ttag) << " *** AM send_shm: msg " << msg_num << endl;

    msg = msg_num;
    fill_array();

    int errstat = request_sem(semid, 0);

    //cout << TmTag(ttag) << " after request_sem, errstat=" << errstat << endl;

    if(errstat == -1){
       perror ("request_sem error");
       return(-1);
    }

    memcpy(shmaddr, buffer, 4000);


    errstat = release_sem(semid, 0);
    if(errstat == -1){
       perror("release_sem error");
       return(-1);
    }

    return(0);
}

//=====================================================================
// Array_mgr::end_shm
//
//=====================================================================

void Array_mgr::end_shm()
{
    struct shmid_ds sbuf;   //shm information buffer -see shm.h
    struct tm *ptr;         //pointer to time info - see time.h

    // query shm
    if (0 == (shmctl(shmid, IPC_STAT, &sbuf))) {
      //outfile << "\tshared memory size:" << sbuf.shm_segsz << endl;
    }
    else perror("shmctl query");
```

```
   while(sbuf.shm_nattch > 1) {
      if (0 != (shmctl(shmid, IPC_STAT, &sbuf))){
         perror("error shmctl query in while");
        break;
      }
      sleep(1);
    };

   outfile << TmTag(ttag) << " AM end_shm: detaching shared memory ";
   shmdt (shmaddr);

   // release shm
   if (0 == (shmctl(shmid, IPC_RMID, 0))) {
      outfile << TmTag(ttag) << " AM end_shm: deleted shm segment\n" ;
   }
   else perror("shmctl - IPC_RMID");

   shmid = -1;

   // Delete semaphore
   outfile << TmTag(ttag)<<" AM end_shm: deleting semaphore" <<semid << endl;
   int errstat = delete_sem(semid);
   if (errstat != 0) {
      outfile << "delete_sem errstat = " << errstat << endl;
   }
   else {
      semid = -1;
   }
   outfile << TmTag(ttag) << " AM end_shm: complete ";

   return;

}
```

```
/*****************************************************************/
// Program Name: listener.cpp                    for Sim Ver 12
// Executable Name: Listener
//
// Author: Cathy Hoaglund
//         partly based on program by Jeff Zamora & Cathy Hoaglund
//                                    Date Last Modified:2/16/05
/*****************************************************************/
/* Description:
    This file is one of a set of units that implement a RPC-based
    client-server application that periodically sends an id, sim time
    and a counter value from the client to the server. The server
    writes the message to a named pipe,then acknowledges the message
    by changing the first value and sending the array back.

    (see myalarm_mgr.cpp for full list of files)

    This file has:
    listener.cpp - server code with programmer defined
        functionality to accept an array of integers and deal
        with them.  A template from rpcgen modified for C++

    Class included:
       C_reply - with the member function repl() which changes the
       id value in the first position to zero, and write2fifo()
       to send values to a named pipe.

*/
/*****************************************************************/

#include <errno.h>
#include <fcntl.h>
#include <fstream.h>
```

```
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <unistd.h>

#include <sys/time.h>
#include <sys/types.h>

#include "myalarm.h"

#include "../utils/TimeUtil.h"


//=============================================================
// C_reply
//=============================================================

class C_reply
{
   public:
                  C_reply();
                  ~C_reply();
        void       repl(long int arr[], int errstat);
        int        initfifo(long int arr[], int arr_size, int *loopmax);
        int        write2fifo(long int arr[], int arr_size, int loopmax);
        int        numloop;

   private:
        FILE       *fp;
        ofstream   outfile;      //diagnostics & status
        char       timetag[16];
        long       h, m, s, u;    // time values
        char       ttag[16];      //timetag

};

//=============================================================
// Constructor & Destructor
//=============================================================

C_reply::C_reply()
{
   outfile.open("Lr.out",ios::app);
   if(!outfile) {
      perror("Listener: cannot open Lr.out");
   }

   outfile << TmTag(ttag) << " Listener started" << endl;
}

C_reply::~C_reply()
{
   outfile.close();
}

//=============================================================
// C_reply::repl
//    takes the input array and changes element 0 as an ack
//=============================================================
void C_reply::repl(long int arr[],int errstat)
{
    cout << arr[0] << ", "<< arr[1] << ", "<< arr[2] << ", "
              << arr[3] << endl;
    outfile << arr[0] << ", "<< arr[1] << ", "<< arr[2] << ", "
              << arr[3] << endl;
    arr[0] = errstat;
}


//=============================================================
// C_reply::initfifo
```

```
//
// This member function runs when msg number is 0
//
// There are 2 FIFOs (named pipes) that must exist in filesystem:
//   fifo1 is written by ExtMod, holds EMflag array, is non-blocking
//   fifo2 is written by Listener, holds message, is blocking
// Listener opens and closes FIFOs but the EMM end is open for
//  the entire run. This function performs initial handshake.
// Sequence:
//    1. Blocking write on fifo2
//    2. Non-blocking read on fifo1 to confirm EM done
//
// The number of 1 ms loops to wait is determined by the Hz rate. The
//   update interval in microseconds is sent with init message 0.
//  Returns 0 on success, -1 on error
//=================================================================


int C_reply::initfifo(long int arr[], int arr_size, int *loopmax)
{
   int fd1, fd2;                    // file descriptors for FIFOs
   int n;                           // number of bytes read
   int i;
   unsigned ACflag[2] = {1,0};       // AlarmClock ready flags

   outfile << TmTag(ttag) << " Listener initfifo" << endl;

   // Calculate how long in millisec to wait for each message
   *loopmax = (arr[3]/ 1000) - 10;
   //outfile <<"loopmax is " << *loopmax << "(" << arr[7]/1000
   //        <<" usec each)" <<endl;


   // Step 1. Blocking write on fifo2
   if ((fd2 = open("fifo2",O_WRONLY|O_APPEND, 0))< 0)  {
      perror("Listener:ERROR:initfifo open failed on fifo2");
      return (-1);
   }
   outfile << TmTag(ttag) << "Lr initfifo 1.fifo2 open. Writing msg: ";

   for (i = 0; i < arr_size; i++) {
      if ((n = write(fd2, (void*)&arr[i], sizeof arr[i])) < 0 ) {
         perror("Lr:initfifo error writing msg to fifo2");
         outfile << "Error - write to fifo2 returned "<< n << endl;
         close(fd2);
         return(-1);
      }

      outfile << arr[i] << " ";
   }
   outfile << endl;

   // Step 2. Non-blocking read on fifo1 to confirm EM done
   if((fd1 = open("fifo1", O_RDONLY|O_NONBLOCK, 0)) < 0) {
       perror("Lr:initfifo: Open failed on fifo1");
       return(-1);
   }
   outfile << TmTag(ttag) << "Lr initfifo 2. fifo1 open. \n";

   long EMflag[5] = {0,0,0,0,0};

   numloop = *loopmax;
   while(numloop > 0) {
      outfile << "\t" << numloop;

      n = read(fd1, &EMflag, sizeof EMflag);
      if (n < 0){
         perror("Lr initfifo: read failure fifo1");
         if (errno == EAGAIN) {
            usleep(1000);
            numloop--;
```

```
            }
            else {
               outfile << "ERROR Lr initfifo: exiting\n";
               close(fd1); close(fd2);
               return(-1);
            }
         }
         else if (n == 0) {
               usleep(1000);
               numloop--;
         }
         else {
            //cout << endl << TmTag(ttag)
            //   << " Lr initfifo: Got OK, EM done with init." << endl;
            outfile << endl << TmTag(ttag)
              << " Lr initfifo: Got OK, EM done with init." << endl;
            outfile << "  EMflag: ";
            for (int j =0;  j < 5; j++) {
              outfile << EMflag[j] << " ";
            }
            outfile << "\n----------------------------------------\n";
            break;
         }
      }

      close(fd1);
      close(fd2);

      return(0);
   }

   //=================================================================
   // C_reply::write2fifo
   //    takes the input array and writes it to fifo2, a named pipe
   // Reading fifo1 prevents one-sided blocking on other end
   // Sequence:
   // 1. Wait for fifo1, read EMM ready
   // 2. Write message to fifo2
   //
   //  Returns 0 on success, -1 on error
   //=================================================================

   int C_reply::write2fifo(long int arr[], int arr_size, int loopmax)
   {
      int fd1, fd2;
      int n;              // number of bytes read
      int i;


      // 1..Wait for fifo1, read EMM ready. Block till fifo1 opened
      fd1 = open("fifo1", O_RDONLY, 0);
      if (fd1 < 0) {
         perror("Lr write2fifo: Step 1 - open failed on fifo1");
         return (-1);
      }
      //outfile << TmTag(ttag) << "Lr write2fifo 1. fifo open. waiting...\n";

      // Loop till fifo written to on other end or timeout

      long lastmsg = -1;
      long EMflag[5] = {0,0,0,0,0};

      numloop = loopmax;  // set  timeout
      while (numloop > 0) {
        outfile << "\t" << numloop;

         n = read(fd1, &EMflag, sizeof EMflag);
         if (n < 0){
            perror("Lr write2fifo: read failure fifo1");
            if (errno == EAGAIN) {
               usleep(1000);
```

```
            numloop--;
         }
         else  {
            outfile << "ERROR Lr write2fifo: exiting\n";
            close(fd1);
            return(-1);
         }
      }
      else if (EMflag[1] == 0) {
         usleep(1000);
         numloop--;
      }
      else {
         //cout << endl << TmTag(ttag)
         //   << " Lr write2fifo: Step 1 - read EM flags." << endl;
         outfile << endl << TmTag(ttag)
            << " Lr write2fifo: Step 1 - read EM flags." << endl;
         outfile << "  EMflag: ";
         for (int j =0;  j < 5; j++) {
            outfile << EMflag[j] << " ";
         }
         outfile << "\n-----------------------------------------\n";

         numloop = 0;
      }
   }

   if (EMflag[1] == 0){
      //cout << TmTag(ttag)
      //        << " Lr write2fifo: Never connected with EM via fifo1 " << endl;
      outfile << TmTag(ttag)
              << " Lr write2fifo: Never connected with EM via fifo1 " << endl;
      close (fd1);
      return (-1);
   }

   // 2. Write message to named pipe - block till read

   if ((fd2 = open("fifo2",O_WRONLY|O_APPEND, 0))< 0)  {
      perror("Lr write2fifo: Open failed on fifo3");
      close (fd1);
      return (-1);
   }
   outfile << TmTag(ttag)
           << " Lr write2fifo: 2. Writing msg to fifo2 " << endl;

   for (i = 0; i < arr_size; i++) {
      if ((n = write(fd2, (void*)&arr[i], sizeof arr[i])) < 0 ) {
         perror("Lr write2fifo: Step 2 - error writing msg to fifo2");
         close(fd1); close(fd2);
         return(-1);
      }
      outfile << arr[i] << " ";
   }
   outfile << "\n=====================================\n";

   close(fd1);
   close(fd2);

   return(0);
}


//==================================================================
// myalarm_repl_1_svc:
//    (created by rpcgen) Copies transmitted XDR values array,
//    and length. Instantiates C_reply class, then invokes the
//    C_reply member function repl() and copies result back to
//    the XDR transport variables
//
//    Extended for use with Sim to write the time and state values
```

```
//    to a named pipe
//
//=====================================================================

values * myalarm_repl_1_svc(values *argp, struct svc_req *rqstp)
{
    static values  result;

    static int     loopmax;

   // Get incoming values from parameter list
   long int *arr = argp->values_val;
   int arr_size = argp->values_len;

   // Instantiate a class to handle reply & write to fifo
   C_reply cr;

   cout << endl << "Listener: got message " << arr[0] << endl;
   int errstat = 0;
   if (arr[0] == 0) {
      errstat = cr.initfifo(arr, arr_size, &loopmax);
   }
   else {
      errstat = cr.write2fifo(arr, arr_size, loopmax);
   }
 //  if (errstat != 0) cout << "errstat: " << errstat << endl;
   cr.repl(arr, errstat);

   // Return ack to client
   result.values_val = arr;
   result.values_len = arr_size;

   return &result;
}
```

Code used to create ExtMod1 executable

```
============================================================
    MAJOR FILES:   /home/choaglun/asdf/external
============================================================
-rw-r--r--    1 choaglun choaglun     8200 Mar  4 16:51 EMMain.C
-rw-rw-r--    1 choaglun choaglun     2699 Feb 12 14:45 ExternalModuleMgr.H
-rw-r--r--    1 choaglun choaglun     5395 Mar  4 17:01 ExternalModuleMgr.C
-rw-rw-r--    1 choaglun choaglun    15537 Mar  4 17:28 EMComm.C

------------------------------------------------------------------------------

// ********************************************************************
// Name: EMMain.C                                  for Sim version 12
//    External Module to send data into SPEEDES simulation
// Author: Cathy Hoaglund                Last modified: 25 Feb 06
//
// Executable used in message-passing or shared memory comm mode
// EMMain reads data from either reflective memory or a FIFO named
// pipe filled by the AlarmSensor process. It then packages up the data
// into a string to get passed in when send_command schedules an event.
//
// The main process manages the connection with the SPEEDES Host Router
// and SpeedesServer. It leaves the incoming messages to emm,an
// instance of class ExternalModuleMgr, which also writes data to a
// file with times.
// ********************************************************************

#include <fstream>

#include "SpIostream.H"
#include "SpStateMgr.H"
#include "SpDataParser.H"
#include <SpStateMgrEvent.H>
#include "SpFreeObjProxy.H"

#include "MyReflect.H"
#include "../utils/TimeUtil.h"
#include "ExternalModuleMgr.H"

//minimal constructor to connect to GVT
SpFreeObjProxy::SpFreeObjProxy(int n) {set_ntypes(n);}

#define WAIT4MSG 1.0    // Time to wait for message each cycle
#define TIMELAG  10.0   // How far behind the Sim in virtual seconds
                        // we can lag. Effectively, virtual frame length

#define EMMOUTFILE "/home/choaglun/asdf/emm.out"
#define EMMDATFILE "/home/choaglun/asdf/data/EM.dat"
#define CONFIGFILE "/home/choaglun/asdf/config/config.par"
#define GOFILE     "/home/choaglun/asdf/OK2Go.txt"
#define ENDFILE    "/home/choaglun/asdf/OK2End.txt"


//------------------------------------------------------------
// main
//------------------------------------------------------------

int main(int argc, char** argv) {
// Command line arguments currently ignored

   ofstream outfile;
   outfile.open(EMMOUTFILE);
   if(!outfile){
      perror("EMMain outfile creation error");
   }
   else {
      outfile.close();
   }
```

```
outfile.open(EMMOUTFILE, ios::app);
if(!outfile){
   perror("EMMain outfile open error");
}

//Initialize configuration variables
char CommType = ' ';        // 3 valid types: m, s, v
int  MaxMsgs = 0;
char nrisk = ' ';
int  simrate = 0;
int  nnodes = 0;
int  physconf = 0;
int  MsgHz = 1;
int  ChkHz = 1;

int  ArrSize = 4;        // no longer an input var
int  Interval = 100000;
int  ChkPeriod = 10000;

//Read config file
ifstream configfile;
configfile.open(CONFIGFILE);
if(!configfile){
   cerr << "EMMain configuration file read error\n";
}
else {
   configfile >>  CommType;
   configfile >>  MaxMsgs;
   configfile >>  nrisk;
   configfile >>  simrate;
   configfile >>  physconf;
   configfile >>  nnodes;
   configfile >>  MsgHz;
   configfile >>  ChkHz;

   configfile.close();
   if (MsgHz != 0) Interval = 1000000 / MsgHz;
   if (ChkHz != 0) ChkPeriod = 1000000 / ChkHz;
}

// Timing variables & sim state variables
char ttag[16]="00:00:00.000";  //timetag string
char datastring[60]=" ";
int  dl = strlen(datastring);
long msg = 0;
long lastmsg = 0;

outfile << endl << TmTag(ttag) << " External Module running " << endl;
cout << setprecision(6);

//--- Connect with Speedes Server
outfile << TmTag(ttag) << " Connecting to Speedes... " << endl;
cout    << ttag        << " Connecting to Speedes... " << endl;
double        timeLag      = TIMELAG;
SpDataParser  speedesDotPar ("speedes.par");
SpStateMgr    stateManager(&speedesDotPar, timeLag);

stateManager.GoToTime(0.0);  //Blocking call
outfile << TmTag(ttag)<< " ExternalModule finished GoToTime(0.0)" << endl;
cout   <<        ttag << " ExternalModule finished GoToTime(0.0)" << endl;

// Instantiate class, do initial synchronization
ExternalModuleMgr emm(CommType, Interval, ChkPeriod, ArrSize, 0, 0);

ofstream Gofile (GOFILE, ios::out);
if(!Gofile){
   cerr << "Couldn't create OK2Go.txt " << endl;
   return(-1);
}else {
   Gofile << TmTag(ttag) << endl;
   Gofile.close();
```

```
}

// check for message 1 before start of loop
msg = emm.get_msg(datastring);
outfile << "first read returned msg number " << msg
     << ", datastring: " << datastring << " dl:" << dl << endl;

// make sure sim is ready
if (stateManager.SpeedesExecuting() != 1) {
   outfile << "Speedes quit - exiting" << endl;
   cerr    << "Speedes quit - exiting" << endl;
   exit(0);
}

//--- Loop till Sim ends or no more messages
while (lastmsg < MaxMsgs +1){

  // If we have an unprocessed message, forward data to sim
  if (msg != 0) {
     dl = strlen(datastring);

     stateManager.SendCommand("Control_SeeAlien",
                         "S_Control_MGR 0", datastring, dl) ;

     // Compute delay for hop1, write data record
     emm.write_record();

     // Send update to console & get next Sim time if still running
     if (stateManager.SpeedesExecuting() == 1) {
        outfile << "\nEMMain: Update Message " << msg << " sent at "
             << TmTag(ttag)  << endl
             << "  Current " << stateManager.GetCurrentTime()
             << " Granted " << stateManager.GetGrantedTime() << endl;

        cout << "\nEMMain: Update Message " << msg << " sent at "
             << ttag  << endl
             << "  Current " << stateManager.GetCurrentTime()
             << " Granted " << stateManager.GetGrantedTime() << endl;

        stateManager.GoToTime(stateManager.GetCurrentTime()+ timeLag);
     }

     lastmsg = msg;
     msg = 0;
  }
  else { // No message, just advance time
     if (stateManager.SpeedesExecuting() != 1) {
        outfile << "Speedes quit - exiting" << endl;
        cerr    << "Speedes quit - exiting" << endl;
         exit(0);
     }

     stateManager.GoToTime(stateManager.GetCurrentTime() + timeLag);

  }

  //debug
  if(msg != 0) {
      cerr << "ERROR in EMMain, resetting msg to 0!!\n";
     msg = 0;
  }

  //If not at max msgs, check for message till read or timeout
  if (lastmsg < MaxMsgs) {

    int waittime = 0;
     while ((msg == 0) && (waittime < Interval)) {
        usleep(ChkPeriod);
        msg = emm.get_msg(datastring);
        waittime = waittime + ChkPeriod;
     }
```

```
    }

    // If we are at max messages, close comm channel
    if (lastmsg >= MaxMsgs) {
       emm.end_msgs();
       lastmsg = MaxMsgs + 1;

       ofstream Endfile (ENDFILE, ios::out);
       if(!Endfile){
          cerr << "Couldn't create OK2End.txt \n";
       }else {
          Endfile << "Done" << endl;
          Endfile.close();
       }
    }
    // go check in with sim, msg or not ...

  } // end primary while loop

   return 0;
}

// ********************************************************************
// Name: ExternalModuleMgr.H                          for Sim version 12
//    External Module Manager class to receive external data and to
//    send data into the SPEEDES simulation
// Author: Cathy Hoaglund                    Last modified: 12 Feb 05
//
//*********************************************************************

#ifndef EXTERNALMODULEMGR_H
#define EXTERNALMODULEMGR_H

#include <fstream>
#include <string.h>
#include <iostream.h>
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

class ExternalModuleMgr
{
   public:
                  ExternalModuleMgr(char inmode, int intv, int ckper,
                                    int arrsize, int intid, int inas_tid);
                    ~ExternalModuleMgr();
      int             get_msg(char* instring);
      void            end_msgs();
      void            write_record();
                                    //CommType-specific
      int             init_fifo();
      int             read_fifo();
      void            close_fifo();

      int             init_shm();
      int             read_shm();
      void            close_shm();

      int             init_vm();
      int             read_vm();
      void            close_vm();

   private:

      char            CommType;  //m = Message Passing via rpc & fifo
                                 //s = Single system Shared Memory
                                 //v = Virtual Machine
                                 //r = Reflective Shared Memory
      int             msgintv;   // message interval
      int              msgckper;  // message check per sec
```

```
        int             msgarr_size; //message array size

        ofstream                EMfile;
        ofstream                outfile;
        int             fd1, fd2;  //fifo file descriptors
        long            EMflag[5]; //flag array

        long            hh0,hh1;        //msg time elapsed hrs
        long            mm0,mm1;        //msg time elapsed mins
        long            ss0,ss1;        //msg time elapsed secs
        long            uu0,uu1;        //msg time elapsed usec
        long            tsec0,tsec1;//msg time total sec since midnight
        double          delay01;    //msg delay in seconds
        long            alien;          //vars[1]
        long            secs0;          //vars[2]
        long             usecs0;                //vars[3]
        long            secs1;          //msg time secs
        long             usecs1;                //msg time usecs

        char            timetag[16];
        char            ttag[16];
        long            h, m, s, u;
        char            datastring[45];
        int             dl;
        long            msg;
        long             lastmsg;
        int             sentack;
        int             initstat;

        // shm variables
        int             semid;          //semaphore id
        int             shmid;          //shm segment id
        char*           shmaddr;
        int             semheld;
        long                    *buffer;

         // pvm variables
        int             tid;            // own task id
        int             as_tid;         //task id of alarm sensor


};

#endif
```

```
// ************************************************************************
// Name: ExternalModuleMgr.C                          for Sim version 12
//    External Module Manager class to receive external data and to
//    send data into the SPEEDES simulation
// Author: Cathy Hoaglund              Last modified: 4 Mar 06
//
// This class sets up the communication architecture based on the mode
// with which it was invoked. It then reads data from the appropriate
// location, packages up the data into a string to get passed in to
// the simulation, hiding the data source.
//
// Specific architectural modes are in the file EMComm.C
// ************************************************************************

#include "ExternalModuleMgr.H"
#include "../utils/TimeUtil.h"
#include "/home/pvm/pvm3/include/pvm3.h"

#include <stdlib.h>
#include <errno.h>
```

```
#define EMMOUTFILE "/home/choaglun/asdf/emm.out"
#define EMMDATFILE "/home/choaglun/asdf/data/EM.dat"


//=============================================================================
// ExternalModuleMgr::ExternalModuleMgr
//=============================================================================

ExternalModuleMgr::ExternalModuleMgr(char inmode, int intv, int ckper,
                                     int arrsize, int intid, int inas_tid)
{
   // Clear out any leftover data in output file & debug file
   EMfile.open(EMMDATFILE);
   if(!EMfile){
      perror("EMM constructor EM.dat file error");
   }
   else {
      EMfile.close();
   }

   outfile.open(EMMOUTFILE, ios::app);
   if(!outfile){
      perror("EMM outfile emm.out open error");
   }

   // Initialize data members
   // Initialize timing variables & sim state variables
   hh0 =0; hh1 =0;     //msg time elapsed hrs
   mm0 =0; mm1 =0;     //msg time elapsed mins
   ss0 =0; ss1 =0;     //msg time elapsed secs
   uu0 =0; uu1 =0;     //msg time elapsed usec
   tsec0 =0; tsec1 =0;//msg time total sec since midnight
   delay01 = 0.0;     //msg delay in seconds
   alien = 1;

   strcpy(timetag,"00:00:00.000");
   strcpy(ttag,"00:00:00.000");
   strcpy(datastring,"00,00,00,00,000000,0,0000000000,000000");

   msg = 0;
   lastmsg = 0;
   sentack = 0;
   initstat = 0;
   semid = shmid = -1;
   semheld = 0;

   // Initialize time
   DoTimetag (&hh1,&mm1,&ss1,&uu1, timetag);
   tsec1 = Convert2Sec(hh1,mm1,ss1);

   // Copy input parameters to data members
   CommType = inmode;          // m, r, s, d
   msgintv = intv;             //msg interval
   msgckper = ckper;           //msg check per sec
   msgarr_size = arrsize;      //message array size
   tid = intid;                        // pvm id
   as_tid = inas_tid;          // pvm id for alarmsensor

   // Select correct init method
   int initstat = -1;
   if (CommType == 'm') {
     initstat = init_fifo();
     //outfile << TmTag(ttag) << "EMM constructor: init_fifo() done" << endl;
   }
   else if (CommType == 's'){
     initstat = init_shm();
     //outfile << TmTag(ttag) << "EMM constructor: init_shm() done" << endl;
   }
   else if (CommType == 'v'){
     initstat = init_vm();
    // outfile << TmTag(ttag) << "EMM constructor: init_vm() done" << endl;
```

```
    }
    else {
      cerr << "invalid CommType" << endl;
      initstat = -1;
    }


    if (initstat == -1) {
       cerr << " *** EMM init error ***\n";
       outfile << TmTag(ttag) << " *** EMM init error *** " << initstat << endl;
       return;
    }
    else {
       outfile <<  TmTag(ttag) << " *** EMM init complete ***\n" << endl;
    }
}


//=============================================================================
// ExternalModuleMgr::~ExternalModuleMgr
//  Destructor - close debug file
//=============================================================================

ExternalModuleMgr::~ExternalModuleMgr()
{
  outfile.close();
}


//=============================================================================
// ExternalModuleMgr::get_msg
//   Decides where to read from based on mode value
//=============================================================================

int ExternalModuleMgr::get_msg(char* instring)
{
    int errstat = -1;
    if (CommType == 'm') {
        errstat = read_fifo();
    }
    else if (CommType == 's') {
        errstat = read_shm();
    }
    else if (CommType == 'v') {
        errstat = read_vm();
    }


    if (!errstat) {
       strcpy(instring, datastring);
    }
    else {
       msg = 0;
    }
    return (msg);
}


//=============================================================================
// ExternalModuleMgr::end_msgs
//   Decides what to close (FIFO, shmid, vm) based on mode
//=============================================================================

void ExternalModuleMgr::end_msgs()
{
    if (CommType == 'm') {
        close_fifo();
    }
    else if (CommType == 's') {
        close_shm();
    }
```

```
   else if (CommType == 'v') {
       close_vm();
   }

}


//=====================================================================
// ExternalModuleMgr::write_record
//  Write out most recently read values to data file
//=====================================================================

void ExternalModuleMgr::write_record()
{
  // Write data record to EM.dat

   EMfile.open(EMMDATFILE, ios::app);
   if (!EMfile) {
      perror ("ERROR: EMM data file not open");
      cerr << "EMM write_record: data file error \n";
   }
   else {

      EMfile << msg << " " << secs0 << " " << usecs0 << " "
             << secs1 << " " << usecs1 << " " << delay01 <<endl;
      EMfile.close();
   }
}


// ***********************************************************************
// Name: EMComm.C                          for Sim version 12
// Executable: ExtMod1 or VM_Starter
//    Methods in the External Module Manager class which receive external
// data in a manner specific to the comm mode selected
// Author: Cathy Hoaglund                Last modified: 4 Mar 06
//
//---------------------------------------------------------------------
// This class sets up the communication architecture based on the mode
// with which it was invoked. It then reads data from the appropriate
// location, packages up the data into a string to get passed in to
// the simulation, hiding the data source.
//
// In the message passing/fifo configuration (m) this class reads data
// from a FIFO named pipe filled by Listener.
//      methods: init_fifo(), read_fifo(), close_fifo()
//
// In the Single system shared memory configuration (s) it waits for data
// to be written by AlarmSensor to a shm area.
//      methods: init_shm(), read_shm(), close_shm()
//
// In the Virtual Machine configuration (v) it checks for data to be
// written by AlarmSensor via PVM
//      methods: init_vm(), read_vm(), close_vm()
//
// ***********************************************************************

#include "ExternalModuleMgr.H"

#include <stdlib.h>
#include <errno.h>

#include "../utils/SemUtil.h"
#include "../utils/TimeUtil.h"
#include "/home/pvm/pvm3/include/pvm3.h"

#define EMMOUTFILE "/home/choaglun/asdf/emm.out"
#define EMMDATFILE "/home/choaglun/asdf/data/EM.dat"
#define SEMFILE    "/home/choaglun/asdf/semid_file"

//************************//
```

```
// MODE-SPECIFIC METHODS  //
//************************//

//============================================================================
// ExternalModuleMgr::init_fifo
// There are 2 FIFOs (named pipes) that must exist in filesystem:
//   fifo1 is written by this process, holds EMflag array, is non-blocking
//   fifo2 is written by Listener, holds data message, is blocking
// Listener opens and closes FIFOs but the EMM end is open for
//  the entire run. This function performs initial handshake.
// Sequence:
//    1. Blocking read from fifo2
//    2. Non-blocking write to fifo1 to signal EM done
// Returns 0 on success, -1 on error
//============================================================================


int ExternalModuleMgr::init_fifo()
{
   int n;                              // number of bytes read
   int i;

   outfile << TmTag(ttag) << " EMM init_fifo start\n";

   // Set up comm to Listener
   EMflag[0] = 0;  // EM Initialization Done
   EMflag[1] = 0;  // Ready to Read
   EMflag[2] = 0;  // Done Reading
   EMflag[3] = 0;  // Last msg number read
   EMflag[4] = 0;  // Last msg number fully processed

   // 1. Blocking read from fifo2 (ignore contents)
   long vars[4] = {0,0,0,0}; // Message buffer
   if ((fd2 = open("fifo2",O_RDONLY, 0)) < 0)  {
      perror("EMM init_fifo:ERROR: open failed on fifo2");
      return (-1);
   }

   //outfile << TmTag(ttag) << "EMM init_fifo: Reading msg: ";
   for (i = 0; i < 4; i++) {
      if ((n = read(fd2, (void*)&vars[i], sizeof vars[i])) < 0 ) {
         perror("EMM init_fifo: error reading msg from fifo2");
         close_fifo();
         return(-1);
      }
   }

   // 2. Non-blocking write to fifo1 to signal EM done
   int snoozes = 1000;
   while((fd1 = open("fifo1", O_WRONLY|O_NONBLOCK, 0)) == -1) {
      snoozes--;
      if (snoozes == 0) {
         cerr << TmTag(ttag) << " EMM init_fifo:Step 2 - open on fifo1"
              << " timed out\n";
         close_fifo();
         return(-1);
      }
      usleep(20000);
   }

   n = 0;
   EMflag[0] = 1;  // init done
   EMflag[1] = 0;  // not ready for next read
   int numloop = 50;
   while(numloop > 0) {
      n = write(fd1, &EMflag, sizeof EMflag);
      if (n < 0){
         perror("EMM init_fifo: Step 2 - write failure fifo1");
         if (errno == EAGAIN) {
            usleep(1000);
            numloop--;
```

```
            }
            else  {
               cerr << "ERROR EMM init_fifo: exiting\n";
               close_fifo();
               return(-1);
            }
         }
         else if (n == 0) {
               usleep(1000);
               numloop--;
         }
         else {
            break;
         }
      }

      // No errors
      close_fifo();
      return(0);
}


//=========================================================================
// ExternalModuleMgr::read_fifo
//  reads the input array from fifo2, a named pipe
//  Uses fifo1 to prevent one-sided blocking
//  Listener opens and closes FIFOs but the EMM end is open for
//  the entire run
// Sequence:
// 1 .Wait for fifo1, write EMM ready flag - if no reader by timeout
//       set msg = 0 and return
// 2. If other side reads fifo1, read message from fifo2
// Returns  0 for success, -1 on error
//=========================================================================

int ExternalModuleMgr::read_fifo()
{
   if (initstat == -1) {
      cerr << "EMM read_fifo: prior fifo read failed \n";
      return (-1);
   }

   // Set flags
   int prior = lastmsg;
   EMflag[0] = 1; // Init done
   EMflag[1] = 1; // Ready to read
   EMflag[2] = 0; // Done with read
   EMflag[3] = lastmsg; // last message read
   EMflag[4] = prior;   // last message fully processed
   int n;
   int i;

   // 1 .Wait a short time for fifo1, write EMM ready flag
   //    If no reader by timeout return.
   int snoozes = 10;
   while((fd1 = open("fifo1", O_WRONLY|O_NONBLOCK, 0)) == -1) {
      snoozes--;
      if (snoozes == 0) {
         outfile << "EMM read_fifo:Step 1 - Open timed out on fifo1\n";
         return(-1);
      }
      usleep(1000);
   }

   int numloop = 50;
   while(numloop > 0) {
      n = write(fd1, &EMflag, sizeof EMflag) <= 0;
      if (n < 0){
         perror("EMM read_fifo: Step 1 - write failure fifo1");
         if (errno == EAGAIN) {
            usleep(1000);
```

```
            numloop--;
         }
         else  {
            cerr <<"ERROR EMM read_fifo: Step 1 failure, exiting\n";
            close_fifo();
            initstat = -1;
            return(-1);
         }
      }
      else {
         numloop = 0;
      }
   }

   // 2. Other side read fifo1 ok, now read message from fifo2
   long vars[4] = {0,0,0,0};

   if ((fd2 = open("fifo2",O_RDONLY, 0)) < 0)  {
      perror("EMM read_fifo:ERROR: open failed on fifo2");
      return (-1);
   }

   for (i = 0; i < 4; i++) {
      if ((n = read(fd2, (void*)&vars[i], sizeof vars[i])) < 0 ) {
         perror("EMM read_fifo: error reading msg from fifo2");
         close_fifo();
         initstat = -1;
         return(-1);
      }
   }

   // Process data just read, if new
   if(vars[0] != lastmsg) {
      msg = vars[0];  secs0 = vars[1];  usecs0 = vars[2];
      alien = vars[3];
      RawTime(&secs1, &usecs1);
      delay01 = (secs1 + usecs1 * 0.000001) - (secs0 + usecs0 * 0.000001);

      // Build datastring from values just read + own stopwatch vals
      sprintf(datastring,"%ld,%ld,%ld,%ld", msg, secs1, usecs1, alien);
      lastmsg = msg;
   }
   else {
       msg = 0;
   }


   // Success!
   close_fifo();

   return (0);
}


//=============================================================================
// ExternalModuleMgr::close_fifo
//   close FIFOs
//=============================================================================

void ExternalModuleMgr::close_fifo()
{
   close(fd1);
   close(fd2);
}


//=============================================================================
// ExternalModuleMgr::init_shm
//    Aaaumes AlarmSensor has written semaphore id and shared mem id to
//    a file, and uses those values to initialize semaphore and shared
//    memory resources
```

```
// Returns 0 on success, -1 on error
//============================================================================

int ExternalModuleMgr::init_shm()
{
   outfile << TmTag(ttag) << " EMM init_shm start\n";

   // Get semaphore id
   semid = shmid = -1;
   ifstream infile;
   int trys = 0;

   infile.open(SEMFILE);
   while (!infile) {
     if (++trys > 100000){
        cerr << TmTag(ttag) << " semid_file error " << endl;
        outfile << TmTag(ttag) << " semid_file error " << endl;
        trys = 0;
       sleep(1);
     }
     infile.open(SEMFILE);
   }

   infile >> semid >> shmid;
   infile.close();

   // Attach shm
   shmaddr = (char*)shmat(shmid, 0, 0);

   // Query shm
   struct shmid_ds sbuf;
   if (0 == (shmctl(shmid, IPC_STAT, &sbuf))) {
     // cout << " shared memory size:" << sbuf.shm_segsz << endl;
   }
   else perror("shmctl err");

   request_sem(semid, 1);

   return(0);
}


//============================================================================
// ExternalModuleMgr::read_shm
//   Checks shared memory for new values
// Returns  0 for success, -1 on error
//============================================================================

int ExternalModuleMgr::read_shm()
{
   int errstat;
   long vars[500];

   if((semid < 0) || (shmid < 0)) {
      cerr << "bad semid or shmid" << semid << " " << shmid << endl;
      outfile << "bad semid or shmid" << semid << " " << shmid << endl;
      exit(86);
   }

   errstat = request_sem(semid, 0); //blocks if AlarmSensor is writing

   memcpy(vars, shmaddr, 2000);

   if(vars[0] != lastmsg) {
      msg = vars[0];  secs0 = vars[1]; usecs0 = vars[2];
      alien = vars[3];

      uu0 = usecs0;
      RawTime(&secs1, &usecs1);
      delay01 = (secs1 + usecs1 * 0.000001) - (secs0 + usecs0 * 0.000001);
      uu1 = usecs1;
```

```
      // Build datastring from values just read + own stopwatch vals
      sprintf(datastring,"%ld,%ld,%06ld,%ld", msg,secs1,usecs1,alien);
      lastmsg = msg;

   }
   else {
      msg = 0;
   }

   errstat = release_sem(semid, 0);
   if (errstat) cout << "error releasing semid " << semid << endl;

   return (errstat);  //msg data member returned by read_msg
}


//=========================================================================
// ExternalModuleMgr::close_shm
//   release resources
//=========================================================================

void ExternalModuleMgr::close_shm()
{
   release_sem(semid, 1);
   int errstat = delete_sem(semid);
   if (errstat) cout << "error deleting semid " << semid << endl;
   shmdt (shmaddr);

   return;
}

//=========================================================================
// ExternalModuleMgr::init_vm
//     Sends ready message to alarm sensor process, a PVM sibling task
// Returns 0 on success, -1 on error
//=========================================================================

int ExternalModuleMgr::init_vm()
{
   outfile << TmTag(ttag) << " EMM init_vm start" << endl;

   // initialize pvm comm
   int info;
   int ready =1;

   // Send a "ready" message to alarm process
   info = pvm_psend(as_tid,0, &ready,1,PVM_INT);
   if(info < 0) { pvm_perror("pvm_psend"), pvm_exit(); return -1;}

   return(0);
}

//=========================================================================
// ExternalModuleMgr::read_vm
//   Checks PVM message queue for new values
// Returns  0 for success, -1 on error
//=========================================================================

int ExternalModuleMgr::read_vm()
{
   int info;
   long vars[500];

   //Get message
   info = pvm_recv(as_tid,0);

   if(info < 0) {
      pvm_perror("EMM");pvm_exit(); exit(1);
   }
```

```
    // unpack message
    info = pvm_upklong(vars, msgarr_size,1);

    if(info < 0) {
       pvm_perror("EMM");pvm_exit(); exit(1);
    }

    if(vars[0] != lastmsg) {
       msg = vars[0];  secs0 = vars[1]; usecs0 = vars[2];
       alien = vars[3];

       uu0 = usecs0;
       RawTime(&secs1, &usecs1);
       delay01 = (secs1 + usecs1 * 0.000001) - (secs0 + usecs0 * 0.000001);
       uu1 = usecs1;

       // Build datastring from values just read + own stopwatch vals
       sprintf(datastring,"%ld,%ld,%06ld,%ld", msg,secs1,usecs1,alien);
       lastmsg = msg;
    }
    else {
       msg = 0;
    }

    return (0);
}


//=============================================================================
// ExternalModuleMgr::close_vm
//   release resources
//=============================================================================

void ExternalModuleMgr::close_vm()
{
    outfile.close();

    return;
}
```

Code used to create Harvester executable

```
============================================================
    MAJOR FILES:   /home/choaglun/asdf/harvester
============================================================
In this section:
-rw-rw-r--    1 choaglun choaglun    25313 Feb 26 12:50 Harvester.cpp
-rw-r--r--    1 choaglun choaglun     2807 Jan 29 21:19 Harvester.h
-rw-r--r--    1 choaglun choaglun      753 Jan 29 21:19 HarvestMain.cpp


// ***************************************************************************
// Name: HarvestMain.cpp                  for Sim  version 8
//     Post-test routine to create merged data file
// Author: Cathy Hoaglund                  Last modified: 30 Jun 03
//
// HarvestMain instantiates the Harvester class,which then opens and reads
// data files created from last Sim run and writes a merged comma delimited
// (csv) file.
//***************************************************************************

#include <fstream>
#include <stdio.h>

#include "Harvester.h"

//===========================================================================
int main (int argc, char *argv[]) {
// command line arguments ignored

   Harvester H;

   return (0);
}


/* ************************************************************************
// Name: Harvester.H  for Sim                              version 11
//     Post-test routine to create merged data files
// Author: Cathy Hoaglund                 Last modified: 27 Feb 05
//
// The Harvester class opens and reads data files created in a Sim run
// and writes merged comma delimited (csv) files.
//************************************************************************
*/

#ifndef HARVESTER_H
#define HARVESTER_H

#include <fstream.h>
#include <iostream.h>
#include <stdio.h>
#include <string>

//===========================================================================

class Harvester
{
   public:
              Harvester();
              ~Harvester();

   private:
        void    open_files();
```

```
        void    merge_data();
        void    write_config();
        void    do_EM();
        void    do_BC();
        void    do_S0();
        void    do_S1();
        void    do_S2();
        void    do_S3();

        long    arr[17];  //arrival times in secs and usecs (7 logical procs)
        double  arrd[6];  // latency increments
        double  arr_at[7]; // inter-message arrival time (delta same LP)
        long    msg1, msg2, msg3s0, msg3s1, msg3s2, msg3s3;
        long    ss0, ss1, ss2, ss3s0, ss3s1, ss3s2, ss3s3;
        long    uu0, uu1, uu2, uu3s0, uu3s1, uu3s2, uu3s3;
        double  delay1, delay2, delay3s0, delay3s1, delay3s2, delay3s3;
        double  this_arrt, AS_last, EM_last, BC_last;
        double  S0_last, S1_last, S2_last, S3_last;
        char    cmode, nrisk;
        char    lrt[7], testnum[4], runday[12];
        int     nummsgs, arrsize, simrate, nnodes, physconf;
        double  Tproc, Tcmt, Nproc, Ncmt;
        int     e, r, m, a;
        double  cpu, wall;
        int     num_drops;
        int     msgHz, chkHz;
        float   PingInt;
        int     dpt1, dpt2, dpt30, dpt31, dpt32, dpt33;
        int     Cntr;
        int     empty[6];
        int     unprocessed_record[6];
        int     done;

        ifstream    EMfile;    // Times AlarmSensor & ExtMod1
        ifstream    BCfile;    // Times BaseControl
        ifstream    S0file;    // Times Ship0
        ifstream    S1file;    // Times Ship1
        ifstream    S2file;    // Times Ship2
        ifstream    S3file;    // Times Ship3
        ifstream    Conffile;   // config.par
        ifstream    Prepfile;  // Prep data gathered after run
        ifstream    LRTfile;   // LastRunTime
        ifstream    TNfile;    // TestNum

        ofstream    outfile1;    // summary csv
        ofstream    outfile2;    // all data, normalized

};

/* LAYOUT OF ARRAYS
   ARRD
   0       1       2         3        4        5
   delay1  delay2  delay3s0  delay3s1 delay3s2 delay3s3

   ARR
   0     1    2    3    4    5    6    7      8      9
   msg1, ss0, uu0, ss1, uu1, ss2, uu2, ss3s0, uu3s0, ss3s1,

   10     11     12     13     14
   uu3s1, ss3s2, uu3s2, ss3s3, uu3s3,

   AAR_AT
   0      1      2     3      4     5      6
   iatAS, iatEM, iatBC, iatS0, iaTS1, iatS2, iatS3,

*/

#endif
```

```
// ***************************************************************************
// Name: Harvester.cpp                                    for Sim  version 12
//     Post-test routine to create merged data files
// Author: Cathy Hoaglund                          Last modified: 26 Feb 06
//
// Harvester opens and reads data files created from last Sim run.
// It writes with all the data to a merged comma delimited (csv) file
// uniquely named with the run time, and also a summary .csv file.
//
// Files expected to be in data directory as input:
//      lastruntime     prep.dat        EM.dat          BC.dat
//      SO.dat          S1.dat          S2.dat          S3.dat
//  Also config.par in config
//
// Class methods:
//      Harvester() -- initializes variables, calls others
//      open_files()
//      write_config()
//      merge_data()
//       do_EM();
//       do_BC();
//       do_SO();
//       do_S1();
//       do_S2();
//       do_S3();

//***************************************************************************

#include "Harvester.h"

//===========================================================================
// Harvester::Harvester
//  Both initializes and starts the action. Expects string to be
//  incorporated in data file names. Does not initialize file pointers.
//===========================================================================

Harvester::Harvester()
{
   // Initialize data members
   int i, j;
   for (i = 0; i < 34; i++) arr[i] =  0;
   for (j = 0; j < 6; j++) arrd[j] =  arr_at[j] = 0.0;
   arr_at[6] = 0.0;
   msg1 = msg2 = msg3s0 = msg3s1 = msg3s2 = msg3s3 = 0;
   ss0 = ss1 = ss2 = ss3s0 = ss3s1 = ss3s2 = ss3s3 = 0;
   uu0 = uu1 = uu2 = uu3s0 = uu3s1 = uu3s2 = uu3s3 = 0;
   delay1 = delay2 = delay3s0 = delay3s1 = delay3s2 = delay3s3 = 0.0;
   this_arrt = AS_last = EM_last = BC_last = 0.0;
   S0_last = S1_last = S2_last = S3_last = 0.0;
   cmode = ' ';
   for (i = 0; i < 12; i++) {
     if (i < 8) lrt[i] = '\0';
     if (i < 5) testnum[i] = '\0';
     runday[i] = '\0';
   }
   nummsgs = arrsize = nrisk = simrate = nnodes = 0;
   e = r = m = a = 0;
   cpu = wall = 0.0;
   num_drops = 0;
   msgHz = chkHz = 0;
   physconf = 0;
   PingInt = 0.0;
   dpt1 = dpt2 = dpt30 = dpt31 = dpt32 = dpt33 = 0;
   Cntr = done = 0;
   for (i = 0; i < 6; i++) {
     empty[i] =  0;
     unprocessed_record[i] = 0;
   }

   // Start processing
   open_files();
```

```
}

//============================================================================
// Harvester::~Harvester - destructor
//============================================================================

Harvester::~Harvester(){ }


//============================================================================
// Harvester::open_files()
//  Opens input and output files. Calls merge_data & write_GVTcsv.
//  Closes files when done
//============================================================================

void Harvester::open_files()
{
   // Get run date & testnum, construct names for output files

   char* LRTptr = "data/lastruntime";
   LRTfile.open(LRTptr);
   if (!LRTfile) {
     cerr << "Error opening lastruntime" << endl;
     exit(1);
   }
   LRTfile >> lrt >> runday;
   LRTfile.close();

   char* TNptr = "data/testnumfile";
   TNfile.open(TNptr);
   if (!TNfile) {
     cerr << "Error opening testnumfile" << endl;
     exit(1);
   }
   TNfile >> testnum;
   TNfile.close();

   char out1[31] = "data/";
   char out2[31] = "data/All_";
   char ext1[] = "_Data.csv";
   char ext2[] = ".csv";
   char ul[] = "_";
   strcat(out1,testnum);
   strcat(out1,ul);
   strcat(out1,lrt);
   strcat(out1,ext1);
   strcat(out2,testnum);
   strcat(out2,ul);
   strcat(out2,lrt);
   strcat(out2,ext2);

   // Open files

   char* EMptr = "data/EM.dat";
   char* BCptr = "data/BC.dat";
   char* S0ptr = "data/S0.dat";
   char* S1ptr = "data/S1.dat";
   char* S2ptr = "data/S2.dat";
   char* S3ptr = "data/S3.dat";
   char* Confptr ="config/config.par";
   char* Prepptr ="data/prep.dat";
   char* OUTptr1 = out1;
   char* OUTptr2 = out2;

   EMfile.open(EMptr);
   if (!EMfile) {
     cerr << "Error opening EM.dat " << endl;
     exit(1);
   }
   BCfile.open(BCptr);
   if (!BCfile) {
```

```
    cerr << "Error opening BC.dat " << endl;
    exit(1);
  }
  S0file.open(S0ptr);
  if (!S0file) {
    cerr << "Error opening S0.dat " << endl;
    exit(1);
  }
  S1file.open(S1ptr);
  if (!S1file) {
    cerr << "Error opening S1.dat " << endl;
    exit(1);
  }
  S2file.open(S2ptr);
  if (!S2file) {
    cerr << "Error opening S2.dat " << endl;
    exit(1);
  }
  S3file.open(S3ptr);
  if (!S3file) {
    cerr << "Error opening S3.dat " << endl;
    exit(1);
  }
  Prepfile.open(Prepptr);
  if (!Prepfile) {
    cerr << "Error opening prep.dat " << endl;
    exit(1);
  }
  Conffile.open(Confptr);
  if (!Conffile) {
    cerr << "Error opening config.par " << endl;
    exit(1);
  }

  outfile1.open(OUTptr1,ios::out);
  if (!outfile1) {
    cerr << "Error opening output summary data file " << endl;
    exit(1);
  }

  outfile2.open(OUTptr2,ios::out);
  if (!outfile2) {
    cerr << "Error opening output normalized full data file " << endl;
    exit(1);
  }


  // Process configuration data
  write_config();

  // Process message data
  merge_data();


  // Close files
  EMfile.close();
  BCfile.close();
  S0file.close();
  S1file.close();
  S2file.close();
  S3file.close();
  Conffile.close();
  Prepfile.close();

  outfile1.close();
  outfile2.close();

}

//=============================================================================
// Harvester::write_config()
```

```
//  Reads prep.dat, writes out as csv file for summary statistics file.
//=========================================================================

void Harvester::write_config()
{
   //cout << "In Harvester::write_config()\n";

   // Previously gathered run id info. Write to output summary file
    outfile1 << "DATE:, " << runday << ",, TIME:, " << lrt
     << " ,RUN ID:, " << testnum << endl;

   // Read config.par file:

   Conffile >> cmode;
   Conffile >> nummsgs;
   Conffile >> nrisk;
   Conffile >> simrate;
   Conffile >> physconf;
   Conffile >> nnodes;
   Conffile >> msgHz;
   Conffile >> chkHz;

   // Read prep file: parse out numerical values for cpu time,
   //   wall time, events, rollbacks from text stat report
   //   Also get committed vs processed numbers just in case
   //   G holds the phrase "GVT=400" which is discarded
   //   If more than 1 node, read and parse Msgs, Anti

   char G[12]; char C[14]; char W[14]; char Tp[12]; char Tc[12];
   char Np[12]; char Nc[12]; char E[12]; char R[12]; char M[12]; char A[12];
   char dum[15];

   Prepfile >> dum >> G >> C >> W >> Tp >> Tc
            >> Np >> Nc >> E >> R;
   if(nnodes > 1) {
      Prepfile >> M >> A;
   }
   else {
      m = a = 0;
   }
   int i;
   double Tproc, Tcmt, Nproc, Ncmt;

   for (i = 0; i < 14; i++) dum[i] = ' ';
   for (i = 4; i <14; i++) dum[i-4] = C[i];
   cpu = atof(dum);
   for (i = 0; i < 14; i++) dum[i] = ' ';
   for (i = 5; i <14; i++) dum[i-5] = W[i];
   wall = atof(dum);
   for (i = 0; i < 14; i++) dum[i] = ' ';
   for (i = 6; i <12; i++) dum[i-6] = Tp[i];
   Tproc = atoi(dum);
   for (i = 0; i < 14; i++) dum[i] = ' ';
   for (i = 5; i <12; i++) dum[i-5] = Tc[i];
   Tcmt = atoi(dum);
   for (i = 0; i < 14; i++) dum[i] = ' ';
   for (i = 6; i <12; i++) dum[i-6] = Np[i];
   Nproc = atoi(dum);
   for (i = 0; i < 14; i++) dum[i] = ' ';
   for (i = 5; i <12; i++) dum[i-5] = Nc[i];
   Ncmt = atoi(dum);
   for (i = 0; i < 14; i++) dum[i] = ' ';
   for (i = 2; i <12; i++) dum[i-2] = E[i];
   e = atoi(dum);
   for (i = 0; i < 14; i++) dum[i] = ' ';
   for (i = 2; i <12; i++) dum[i-2] = R[i];
   r = atoi(dum);

   if(nnodes > 1) {
      for (i = 0; i < 14; i++) dum[i] = ' ';
```

```
      for (i = 2; i <12; i++) dum[i-2] = M[i];
      m = atoi(dum);
      for (i = 0; i < 14; i++) dum[i] = ' ';
      for (i = 2; i <12; i++) dum[i-2] = A[i];
      a = atoi(dum);
   }
   else {
      m = a = 0;
   }

   //cout << "Read SPEEDES statistics: cpu "<< cpu << ", wall "
   //   << wall  << ", events " << e << ", rollbacks " << r
   //   << " msgs " << m << " antimsgs " << a << endl;


   // Write configuration section of output summary file
   // Write header line 2 A1:F1
   outfile1 << " CommMode , Nodes , Nrisk , SimRate ,"
            << "  MsgHz , ChkHz , NumMsgs, Physconf" << endl;

   // Write data line 3 A2..F2 & blank line 4
   outfile1 << cmode << "," << nnodes << ", " << nrisk << ","
            << simrate << ", " << msgHz << "," << chkHz << ", "
            << nummsgs << "," << physconf << ","
            << endl << endl;

   // Write statistics section of output summary file
   // Write header line 5 A5..G5
   outfile1 << " CPU , Wall ,"
            << "  Events , Rbacks , Msgs , Anti, Drops " << endl;

   // Write data line 6 & blank line 7
   int RowEnd = 11 + nummsgs;
   int RowTot = RowEnd + 2;

   outfile1 << cpu << "," << wall << "," << e << "," << r << ","
            << m << "," << a << " ,=COUNTBLANK(B12:B" << RowEnd
            << ") ,"  << endl << endl;

   // Write header line 8
   outfile1 << "TIMES- ,AvgHop1 ,AvgHop2 ,AvgHop3 ,AvgE2E " << endl;

   // Write data line 9 & blank line 10
   outfile1 << " ,=SUM(B12:B" << RowEnd << ")/B" << RowTot ;
   outfile1 << " ,=SUM(C12:C" << RowEnd << ")/C" << RowTot ;

   outfile1 << " ,=SUM(D12:G" << RowEnd << ")/SUM(D" << RowTot
            <<":G" << RowTot << ") ";

   outfile1 << ",=SUM(B9:D9)" << " , " << endl << endl;

}

//=============================================================================
// Harvester::merge_data
//  Reads data files, writes merged record header
//  For each message
//     1. Read record for 6 data files into array of sequentially numbered
//        data values
//      For each record in the 6 files there are 3 cases:
//       A) Empty - at EOF, (all done or file may be shorter than others)
//       B) Cntr < Msg - we had a data drop-out (a Bad Thing). Fill array
//          with 0, retain values till Cntr catches up
//       C) Cntr = Msg - normal case. Process values
//       D) Cntr > Msg - something is not right - drop record and try again
//
//    2. May have more records in some .dat files - keep processing records
//        till all 6 data sources report complete
//
//    2.  Write merged records to output files
//
```

```
//=============================================================================

void Harvester::merge_data()
{
   // Write header line for single run summary file
   outfile1 << "msg1,delay1,delay2,delay3s0,delay3s1,delay3s2,delay3s3"
            << ",ss0 ,uu0 ,ss1 ,uu1 ,ss2 ,uu2 ,ss3s0 ,uu3s0"
            << ",ss3s1 ,uu3s1 ,ss3s2 ,uu3s2 ,ss3s3 ,uu3s3";
   outfile1 << endl;

   // Write header line for big normalized file
   outfile2 << " ,msg ,delay1 ,delay2 ,delay3s0 ,delay3s1 ,delay3s2 ,delay3s3"
            << ",iatAS,iatEM ,iatBC ,iats0 ,iats1 ,iats2 ,iats3"
            << ",testnum ,runday ,lrt ,cmode ,nummsgs ,nrisk"
            << ",simrate ,nnodes ,e ,r ,m ,a ,cpu ,wall ,msgHz ,chkHz"
            << ",physconf";
   outfile2 << endl;


   // Read in a message worth of data
   int i;
   Cntr = 1;
   for (i=0; i < 6; i++) {
     empty[i] = unprocessed_record[i] = 0;
   }
   done = 0;

   while (!done) {

      // Assemble the set of data points where msg = Cntr
      do_EM();
      do_BC();
      do_S0();
      do_S1();
      do_S2();
      do_S3();


      //---- Are we done yet? --------------
      int sum = 0;
      for (i = 0; i < 6; i++) {
         sum = sum + empty[i];
      }

      if (sum >= 6) {    //ready to write
         done = 1;
      }
      else {
         // Write merged data record to both files
         outfile1 << Cntr << ",";
         outfile2 << " ," << Cntr << ",";
         arr[0] = 0;

        //arrd[0] = delay1. If negative, message was lost
        int lostmsg = 0;
        if (arrd[0] < 0.0) lostmsg = 1;

         for (i = 0; i < 6; i++) {     // write delays per each hop
             // if lostmsg  write blank, not a number or space
            if(lostmsg) {
               outfile1 << ",";
               outfile2 << ",";
            }
            else  {
               outfile1 << arrd[i] << ", ";
               outfile2 << arrd[i] << ", ";
            }
            arrd[i] = 0.0;
         }

         for (i = 0; i < 7; i++) {    //write intermsg arrival time difference
```

```
            if((lostmsg) || (arr_at[i] == 0.0)) {
               outfile2 << ",";
            }
            else {
               outfile2 << arr_at[i] << ",";
            }
            arr_at[i] = 0.0;
         }

         // write configuration parameters to big normalized file
         outfile2 << testnum  << "," <<  runday << "," << lrt<< ","
              << cmode  << "," << nummsgs  << ","
              << nrisk  << "," << simrate  << ","
              << nnodes  << "," << e  << "," << r  << "," << m  << ","
              << a  << "," << cpu  << "," << wall << "," << msgHz  << ","
              << chkHz << "," << physconf << ",";
         outfile2 << endl;


         for (i = 1; i < 15; i++) {   //write actual times to single test file
            outfile1 << arr[i] << ",";
            arr[i] = 0;
         }
         outfile1 << endl;

      }  //end if

      Cntr++;

   } // end while

   // Write data points counts to summary
   outfile1 << endl;
   outfile1 << " ," << dpt1 << " ," << dpt2 <<" ,"
            << dpt30 << " ," << dpt31 << " ," << dpt32 << " ,"
            << dpt33 << " ," << endl;
   cout << "Harvester found " << dpt1  << " messages" << endl;
}

//============================================================================
// Harvester::do_EM
//  Processes EM.dat as described in merge_data()
//
// 0.EMfile: ------------------------------
// arr  0     1    2    3    4      arrd 0   arr_at 0     1
//     msg1,  s0, uu0, ss1, uu1     delay1          AS iat  EM iat
//============================================================================

void Harvester::do_EM()
{
   if (!unprocessed_record[0]) {
      EMfile >>  msg1 >> ss0 >> uu0 >> ss1 >> uu1 >>  delay1;

      if(EMfile.eof()) {
         empty[0] = 1;  // Case A: EOF
         return;
      }
   }

   int EMdone = 0;
   while (!EMdone){
      if (Cntr < msg1) {          // Case B: lost a message
         cout  << "EMfile. Missing message #" << Cntr << endl;
         arr[0] = Cntr;
        arrd[0] = -9.0;
        arr_at[0] = 0.0;
         unprocessed_record[0] = 1;
         num_drops++;
         EMdone = 1;
      }
      else if (Cntr == msg1){        // Case C: good record, and timely
```

```
        cout << Cntr << " EMfile: " << msg1 << " " << ss0 << " " << uu0
             <<  "  " << ss1 << " " << uu1 << " " << delay1 << endl;

        //compute and capture intermessage arrival time
        this_arrt = ss0 + uu0 *0.000001;
        if (AS_last > 0.0) {
           arr_at[0] = this_arrt - AS_last;
        }
        AS_last = this_arrt;

        this_arrt = ss1 + uu1 *0.000001;
        if (EM_last > 0.0) {
           arr_at[1] = this_arrt - EM_last;
        }
        EM_last = this_arrt;

        arr[0] = msg1; msg1 = 0;
        arr[1] = ss0;  ss0  = 0;
        arr[2] = uu0;  uu0  = 0;
        arr[3] = ss1;  ss1  = 0;
        arr[4] = uu1;  uu1  = 0;
        arrd[0] = delay1; delay1 = 0.0;

        unprocessed_record[0] = 0;
        dpt1++;
        EMdone = 1;
     }
     else  {                        // Case D: out-of-order error
        cout << "EMfile: Dropping out-of-order message.Counter " << Cntr
             << " is greater than Msg # " << msg1 << endl;
        arr[0] = Cntr;
        msg1 = 0;
        ss0  = 0;
        uu0  = 0;
        ss1  = 0;
        uu1  = 0;
        delay1 = this_arrt = AS_last = EM_last = 0.0;
        unprocessed_record[0] = 0;
        num_drops++;
        EMdone = 1;
     }
  }
  return;
}

//=============================================================================
// Harvester::do_BC
//  Processes BC.dat as described in merge_data()
//
// 1.BCfile: ----------------------------------------
// arr   5   6       arrd   1           arr_at 2
//      ss2, uu2           delay2       BC iat
//=============================================================================

void Harvester::do_BC()
{
  if (!unprocessed_record[1]) {
     BCfile >>  msg2 >> ss2 >> uu2 >> delay2;

     if(BCfile.eof()) {
        empty[1] = 1;   // Case A: EOF
        return;
     }
  }

  int BCdone = 0;
  while (!BCdone) {
     if (Cntr < msg2) {       // Case B: lost a message, don't fill
        cout  << " BCfile. Missing message #" << Cntr << endl;
        unprocessed_record[1] = 1;
        num_drops++;
```

```
            BCdone = 1;
         }
         else if (Cntr == msg2){        // Case C: good record, and timely
            cout << Cntr << "  BCfile: " << msg2 << " "  << ss2 << " "
            << uu2 << " " << delay2 << endl;

            //compute and capture intermessage arrival time
            this_arrt = ss2 + uu2 *0.000001;
            if (BC_last > 0.0) {
               arr_at[2] = this_arrt - BC_last;
            }
            BC_last = this_arrt;

            arr[5] = ss2; ss2  = 0;
            arr[6] = uu2; uu2  = 0;
            arrd[1] = delay2; delay2 = 0.0;

            unprocessed_record[1] = 0;
            msg2 = 0;
            dpt2++;
            BCdone = 1;
         }
         else  {                        // Case D: out-of-order error
            cout << "BCfile: Dropping out-of-order message.Counter " << Cntr
                 << " is greater than Msg # " << msg1 << endl;

            msg2 = 0;
            ss2  = 0;
            uu2  = 0;
            delay2 =  this_arrt = BC_last = 0.0;
            unprocessed_record[1] = 0;
            num_drops++;
            BCdone = 1;
         }
      }
      return;
   }

   //=============================================================================
   // Harvester::do_S0
   //  Processes S0.dat as described in merge_data()
   // 2.S0file: ----------------------------------------
   // arr   7      8          arrd  2          arr_at 3
   //       ss3s0, uu3s0            delay3s0    S0 iat
   //=============================================================================

   void Harvester::do_S0()
   {
      if (!unprocessed_record[2]) {
         S0file >>  msg3s0 >>  ss3s0 >> uu3s0 >> delay3s0;

         if(S0file.eof()) {
            empty[2] = 1;  // Case A: EOF
            return;
         }
      }

      int S0done = 0;
      while (!S0done) {
         if (Cntr < msg3s0) {  // Case B: lost a message, don't fill
            cout  << "  S0file. Missing message #" << Cntr << endl;
            unprocessed_record[2] = 1;
            num_drops++;
            S0done = 1;
         }
         else if (Cntr == msg3s0){      // Case C: good record, and timely
            cout << Cntr << "   S0file: " << msg3s0 << " " << ss3s0 << " "
                 << uu3s0 << " " << delay3s0  << endl;

            //compute and capture intermessage arrival time
            this_arrt = ss3s0 + uu3s0 *0.000001;
```

```
         if (S0_last > 0.0) {
            arr_at[3] = this_arrt - S0_last;
         }
         S0_last = this_arrt;

         arr[7] = ss3s0;  ss3s0  = 0;
         arr[8] = uu3s0;  uu3s0  = 0;
         arrd[2] = delay3s0; delay3s0 = 0.0;

         unprocessed_record[2] = 0;
        msg3s0 = 0;
         dpt30++;
         S0done = 1;
      }
      else  {                    // Case D: out-of-order error
         cout << "S0file: Dropping out-of-order message.Counter " << Cntr
              << " is greater than Msg # " << msg1 << endl;

         msg3s0 = 0;
         ss3s0  = 0;
         uu3s0  = 0;
         delay3s0 =  this_arrt = S0_last = 0.0;
         unprocessed_record[2] = 0;
         num_drops++;
        S0done = 1;
      }
   }
   return;
}

//============================================================================
// Harvester::do_S1
//  Processes S1.dat as described in merge_data()
// 3.S1file: ----------------------------------------
// arr   9      10         arrd   3         arr_at 4
//       ss3s1, uu3s1             delay3s1   S1_iat
//============================================================================

void Harvester::do_S1()
{
   if (!unprocessed_record[3]) {
      S1file >>  msg3s1 >> ss3s1 >> uu3s1 >> delay3s1;

      if(S1file.eof()) {
         empty[3] = 1; // Case A: EOF
         return;
      }
   }

   int S1done = 0;
   while (!S1done) {
      if (Cntr < msg3s1) {  // Case B: lost a message, don't fill
         cout  << "    S1file. Missing message #" << Cntr << endl;
         unprocessed_record[3] = 1;
         num_drops++;
         S1done = 1;
      }
      else if (Cntr == msg3s1){     // Case C: good record, and timely
         cout << Cntr << "    S1file: " << msg3s1 << " " << ss3s1 << " "
              << uu3s1 << " " << delay3s1  << endl;

        //compute and capture intermessage arrival time
        this_arrt = ss3s1 + uu3s1 *0.000001;
         if (S1_last > 0.0) {
           arr_at[4] = this_arrt - S1_last;
         }
        S1_last = this_arrt;

         arr[9] = ss3s1;  ss3s1  = 0;
         arr[10] = uu3s1;  uu3s1  = 0;
         arrd[3] = delay3s1; delay3s1 = 0.0;
```

```
            unprocessed_record[3] = 0;
            msg3s1 = 0;
            dpt31++;
            S1done = 1;
         }
         else  {                    // Case D: out-of-order error
            cout << "S1file: Dropping out-of-order message.Counter " << Cntr
                 << " is greater than Msg # " << msg1 << endl;

            msg3s1 = 0;
            ss3s1  = 0;
            uu3s1  = 0;
            delay3s1 =  this_arrt = S1_last = 0.0;
            unprocessed_record[3] = 0;
            num_drops++;
            S1done = 1;
         }
      }
      return;
   }

   //=============================================================================
   // Harvester::do_S2
   //  Processes S2.dat as described in merge_data()
   // 4.S2file: ----------------------------------------
   // arr   11     12     arrd  4          arr_at 5
   //       ss3s2, uu3s2        delay3s2     S2 iat
   //=============================================================================

   void Harvester::do_S2()
   {
      if (!unprocessed_record[4]) {
         S2file >>  msg3s2 >> ss3s2 >> uu3s2 >> delay3s2;

         if(S2file.eof()) {
            empty[4] = 1;  // Case A: EOF
            return;
         }
      }

      int S2done = 0;
      while (!S2done) {
         if (Cntr < msg3s2) {  // Case B: lost a message, don't fill
            cout  << "     S2file. Missing message #" << Cntr << endl;
            unprocessed_record[4] = 1;
            num_drops++;
            S2done = 1;
         }
         else if (Cntr == msg3s2){      // Case C: good record, and timely
           cout << Cntr << "     S2file: " << msg3s2 << " " << ss3s2 << " "
                << uu3s2 << " " << delay3s2  << endl;

           //compute and capture intermessage arrival time
           this_arrt = ss3s2 + uu3s2 *0.000001;
           if (S2_last > 0.0) {
              arr_at[5] = this_arrt - S2_last;
           }
           S2_last = this_arrt;

           arr[11] = ss3s2;  ss3s2  = 0;
           arr[12] = uu3s2;  uu3s2  = 0;
           arrd[4] = delay3s2; delay3s2 = 0.0;

           unprocessed_record[4] = 0;
           msg3s2 = 0;
           dpt32++;
           S2done = 1;
         }
         else  {                    // Case D: out-of-order error
            cout << "S2file:  Dropping out-of-order message.Counter " << Cntr
```

```
            << " is greater than Msg # " << msg1 << endl;

         msg3s2 = 0;
         ss3s2  = 0;
         uu3s2  = 0;
         delay3s2 =  this_arrt = S2_last = 0.0;
         unprocessed_record[4] = 0;
         num_drops++;
         S2done = 1;
      }
   }
   return;
}

//=============================================================================
// Harvester::do_S3
//  Processes S3.dat as described in merge_data()
// 5. S3file:-------------------------------------------
// arr  13     14    arrd   5          arr_at 6
//      ss3s3, uu3s3       delay3s3       S3 iat
//=============================================================================

void Harvester::do_S3()
{
   if (!unprocessed_record[5]) {
      S3file >>  msg3s3 >>  ss3s3 >> uu3s3 >> delay3s3;

      if(S3file.eof()) {
         empty[5] = 1;    // Case A: EOF
         return;
      }
   }

   int S3done = 0;
   while (!S3done) {
      if (Cntr < msg3s3) {  // Case B: lost a message, don't fill
         cout  << "      S3file. Missing message #" << Cntr << endl;
         unprocessed_record[5] = 1;
         num_drops++;
         S3done = 1;
      }
      else if (Cntr == msg3s3){     // Case C: good record, and timely
         cout << Cntr << "      S3file: " << msg3s3 << " "  << ss3s3 << " "
              << uu3s3 << " " << delay3s3  << endl;

        //compute and capture intermessage arrival time
        this_arrt = ss3s3 + uu3s3 *0.000001;
        if (S3_last > 0.0) {
           arr_at[6] = this_arrt - S3_last;
        }
        S3_last = this_arrt;

         arr[13] = ss3s3;  ss3s3  = 0;
         arr[14] = uu3s3;  uu3s3  = 0;
         arrd[5] = delay3s3; delay3s3 = 0.0;

         unprocessed_record[5] = 0;
         msg3s3 = 0;
         dpt33++;
         S3done = 1;
      }
      else  {                   // Case D: out-of-order error
         cout << "S3file: Dropping out-of-order message.Counter " << Cntr
              << " is greater than Msg # " << msg1 << endl;

         msg3s3 = 0;
         ss3s3  = 0;
         uu3s3  = 0;
         delay3s3 = this_arrt = S3_last = 0.0;
         unprocessed_record[5] = 0;
         num_drops++;
```

```
        S3done = 1;

      }
   }
   return;
}
```

Code used to create Ships executable

```
=============================================================
    MAJOR FILES:   /home/choaglun/asdf/mysim
=============================================================
In This Section
-rw-r--r--    1 choaglun choaglun      842 Feb 17 09:55 SimMain.C
-rw-r--r--    1 choaglun choaglun     6531 Feb 17 10:15 S_Control.C
-rw-r--r--    1 choaglun choaglun     2090 Jan 29 21:20 S_Control.H
-rw-r--r--    1 choaglun choaglun    11470 Jan 29 21:20 S_Ship.C
-rw-r--r--    1 choaglun choaglun     3737 Jan 29 21:20 S_Ship.H
-------------------------------------------------------------------------------


//*********************************************************************
// SimMain.C                                        For Sim Version 12
// Driver for SPEEDES simulation
// Author: Cathy Hoaglund    Last modified: 17 Feb 06
//
//  SimMain uses the PlugIn definitions created in each SimObj class
// (currently Ship and Control) to instantiate the objects
// and their associated events. The ExecuteSpeedes call starts
// the Init routines in all SimObj classes. The simulation ends at
// the sim time given in parameter tend in speedes.par
//
// Command line input: number of nodes Speedes should use [optional]
//*********************************************************************

#include "SpMainPlugIn.H"
#include <unistd.h>

void PlugInShips();
void PlugInControl();
void signal_mgr(int sigval);

int main(int argc, char** argv) {
  PlugInShips();
  PlugInControl();

  ExecuteSpeedes(argc, argv);
}
=============================================================


//*********************************************************************
// Name: S_Control.H                                 for Sim Version 12
//    Base Control Class Definition for SPEEDES Simulation Object
// Author: Cathy Hoaglund
//                                                 Last modified: 16 Feb 05
//
//    The Base Control station is located at the center of gaming area, but
// that fact is known only by the spaceships. It sends transmit request
// signals at regular intervals, for a heartbeat function.
//
//    It also receives notice when and where an alien ship appears out of
// hyperspace from the external module and sends the information to all
// ships. It writes the message sent and received times to a file.
//*********************************************************************

#ifndef S_Control_H
#define S_Control_H
```

```
#include "SpSimObj.H"
#include "SpDefineSimObj.H"
#include "SpDefineEvent.H"

#include <time.h>
#include <sys/time.h>
#include <unistd.h>
#include <fstream>


//-------------------------------------------------------------------------

class S_Control : public SpSimObj {
  public:
    S_Control() {}
    virtual ~S_Control() {}
    virtual void Init();
    void        Starter();
    void        Closer();
    void         Req();
    void         SeeAlien();

  protected:
  private:
    char         timetag[16];        // form hh:mm:ss.uuuuuu
    char         ttag[16];           // form hh:mm:ss.uuuuuu

    // Rollbackable variables used in output
    RB_int      msg;
    RB_int     hh1;
    RB_int     mm1;
    RB_int     ss1;
    RB_int     uu1;
    RB_int     alienTF;
    RB_int     alienX;
    RB_int     alienY;
    RB_int     hh2;
    RB_int      mm2;
    RB_int     ss2;
    RB_int     uu2;
    RB_double  delay;

};

DEFINE_SIMOBJ(S_Control, 1, BLOCK);
DEFINE_SIMOBJ_EVENT_0_ARG(Control_StartUp, S_Control, Starter);
DEFINE_SIMOBJ_EVENT_0_ARG(Control_CloseUp, S_Control, Closer);
DEFINE_SIMOBJ_EVENT_0_ARG(Control_ReqTX, S_Control, Req);
DEFINE_SIMOBJ_EVENT_0_ARG(Control_SeeAlien, S_Control, SeeAlien);

#endif


//*************************************************************************
// Nome: S_Ship.H                                        for Sim Version 12
// Spaceship Class Definition for SPEEDES simulation objects
// Author: Cathy Hoaglund                          Last modified: 17 Feb 05
//
//    Spaceships cruise back and forth over the 2D gaming area, emitting
//  position signals via point-to-point events, and calculating the distance
//  from Base Control in the center, and from other detected ships. They
//  also respond to periodic requests to report their status, requests
//  sent from the base control object as Ship_Transmit_Prep events.
//    The number of ship object instances is set by NumShips.
//
//*************************************************************************

#ifndef S_Ship_H
#define S_Ship_H

#include "SpSimObj.H"
```

```
#include "SpDefineSimObj.H"
#include "SpDefineEvent.H"

#include <time.h>
#include <sys/time.h>
#include <unistd.h>
#include <fstream>

//-------------------------------------------------------------------------
const int NumShips = 4;  // Multiples of 4 work best
const int Half = 2;


class S_Ship : public SpSimObj {
  public:
    S_Ship() {}
    virtual ~S_Ship() {}
    virtual void Init();
    void         Go();
    void         Alert(long msg,long s2,long u2,long aTF);

    void         TransmitPrep(double transmitTime);
    void         Transmit();
    void         Position_Send();
    void         Position_Recv(double SentTime, int otherShip,
                               double otherXpos, double otherYpos,
                               int otherXdir, int otherYdir);

  protected:
  private:
   //Vars that do not change in a given run:
    double       Velocity;           // Ship Velocity
    int          MyId;               // Sim Object Global Id
    time_t       startsec;           // Time structure filled by ctime
    int          startusec;          // Time value from gettimeofday()
    char         sdatafile[10];      // Name of own data file
    ofstream     Sfile;              // stream for diagnostics

  // Ship SimObj State vars that must be rollbackable:
    RB_double    Xpos;               // Ship Position S->N
    RB_double    Ypos;               // Ship Position W->E
    RB_int       Xdir;               // 1 going north, -1 going south
    RB_int       Ydir;               // 1 going east, -1 going west
    RB_double    LastTimePosUpdate;  // Time at last position check
    RB_double    LastTimeTX;         // Time at last telemetry transmit
    RB_int       TXState;            // 1 = Transmitting; 0 = Silent
    RB_int       TimesTransmitted;   // Number of times ship has reported
    RB_int       NumShipGoEvents;    // Number of Go events
    RB_int       RedAlert;           // 1 if alien in vicinity, 0 if not


 // Utility functions
    void         FindPos(double numsecs); //Update coords & dirs
    double       FindDist(double X1, double Y1,
                          double X2, double Y2); //distance formula
    void         UpdateClockTime();    // Update timetag
    char         timetag[16];      // Timetag
    char         ttag[16];         // Timetag
};

//Macros created by SPEEDES framework:
// Maximum number of arguments is 8
DEFINE_SIMOBJ(S_Ship, NumShips, BLOCK);
DEFINE_SIMOBJ_EVENT_0_ARG(Ship_Go, S_Ship, Go);
DEFINE_SIMOBJ_EVENT_4_ARG(Ship_Alert, S_Ship, Alert, long, long, long,long);
DEFINE_SIMOBJ_EVENT_1_ARG(Ship_TransmitPrep, S_Ship, TransmitPrep, double);
DEFINE_SIMOBJ_EVENT_0_ARG(Ship_Transmit, S_Ship, Transmit);

DEFINE_SIMOBJ_EVENT_0_ARG(Ship_PosSend, S_Ship, Position_Send);
DEFINE_SIMOBJ_EVENT_6_ARG(Ship_PosRecv, S_Ship, Position_Recv, double,
                          int, double, double, int, int);
```

```
#endif
//*************************************************************************
// Program name:S_Ship.C                                  for Sim Version 12
//   Spaceship Class Implementation for SPEEDES simulation objects
// Author: Cathy Hoaglund                              Last modified: 17 Feb 05
//
//   The Ship objects share ownership of static data regarding gaming area,
// but run as independent processes with unique SimObjGlobalIds. There are
// no explicit synchronization points, but events are guaranteed to be in
// logical order. Transmit (TX) requests are corrected if for a past time,
// and an "internal" log entry written to match the "Transmit" entry when
// the actual Transmit request is sent. Position broadcasts are scheduled
// regularly at custom intervals.
//    The SPEEDES framework controls event list processing and SimTime,and
// defines PlugInShips macro, which enable the SCHEDULE_ macros
//    A straggler position or TX request event could cause rollback so all
// state variables that could change are rollbackable.RB_ostream provides
// rollbackable file output, unlike cout.
//
//*************************************************************************

#include "SpGlobalFunctions.H"
#include "SpMainPlugIn.H"
#include "RB_ostream.H"

#include "S_Ship.H"
#include "../utils/TimeUtil.h"

//-------------------------------------------------------------------------
// Shared Simulation Gaming Area Data
static int BorderLen = 3000;   //units: thousand kilometers
static int W_edge = 0;
static int S_edge = 0;
static int N_edge = BorderLen;
static int E_edge = BorderLen;

//Shared output file for ship log events
char* fileptr = "logs/Logfile";
RB_ostream   RBout(fileptr);

char* sfptr0 = "data/S0.dat";
RB_ostream  Sdata0(sfptr0);
char* sfptr1 = "data/S1.dat";
RB_ostream  Sdata1(sfptr1);
char* sfptr2 = "data/S2.dat";
RB_ostream  Sdata2(sfptr2);
char* sfptr3 = "data/S3.dat";
RB_ostream  Sdata3(sfptr3);

//-------------------------------------------------------------

// Plug Objects and Events into Framework
void PlugInShips() {
  PLUG_IN_SIMOBJ(S_Ship);
  PLUG_IN_EVENT(Ship_Transmit);
  PLUG_IN_EVENT(Ship_TransmitPrep);
  PLUG_IN_EVENT(Ship_Go);
  PLUG_IN_EVENT(Ship_Alert);
  PLUG_IN_EVENT(Ship_PosSend);
  PLUG_IN_EVENT(Ship_PosRecv);
}




//=========================================================================
// S_Ship::Init
// Psuedo constructor, runs once before Time Zero
//=========================================================================
```

```
void S_Ship::Init() {

   MyId              = SpGetSimObjGlobalId();
   float Seg         = (float) BorderLen / (float) (Half - 1);
   Velocity          = ((MyId + 1.0) * 5.0); //kiloklicks per sec
   LastTimePosUpdate = 0.0;
   LastTimeTX        = 0.0;
   TXState           = 1;
   TimesTransmitted  = 0;
   NumShipGoEvents   = 1;

   // Determine starting place in gaming area
   Xpos = (MyId % Half) * Seg;  //space evenly on W and E edge
   if (Xpos > (BorderLen/2))
      Xdir = -1;  //heading up N
   else
      Xdir =1;    // heading down S

   if (MyId < Half)  {
      Ypos = W_edge;
      Ydir = 1;         // heading E
   }
   else {
      Ypos = E_edge;
      Ydir = -1;    // heading W
   }


   //set up non-rollbackable data recording file - clean out leftovers
   sprintf(sdatafile,"data/S%d.dat",MyId);
   Sfile.open(sdatafile);
   if(!Sfile) {
      cout << "Ship " << MyId << " file error\n";
   }else {
      Sfile.close();
   }
   TmTag(ttag);

   // Initialization message with wallclock time
   cout << "S_Ship Init " << MyId<< ": Local time: " << ttag << endl;
   cout << setprecision(6);
   Sfile << "S_Ship Init " << MyId<< ": Local time: " << ttag << endl;
   Sfile << setprecision(6);

   // write rollbackable entry to logfile, schedule next events
   RBout << "START:Ship " << MyId << ": Position: ("
        << Xpos << ", " << Ypos << "), Velocity: "
        << Velocity << " kiloclick/sec" <<  " Heading: ";
   if(Xdir == 1) RBout << "N"; else RBout << "S";
   if(Ydir == 1) RBout << "E"; else RBout << "W";
   RBout <<" Clock:" << ttag << endl;


   SCHEDULE_Ship_Go(0.0, SpGetObjHandle());
   SCHEDULE_Ship_PosSend((MyId + 10.0), SpGetObjHandle());
}



//////////////////////
/// event handlers ///
//////////////////////

//=====================================================================
// S_Ship::Go
// Handle Ship_Go events.  Update history, wait for next event
// Pretty useless, included just to increase event density
//=====================================================================

void S_Ship::Go() {
```

```
    if (TXState == 1) {
      LastTimeTX = SpGetTime();
      NumShipGoEvents++;
      TXState       = 0;
    }
}

//=================================================================
// S_Ship::Alert
// Handle Ship_Alert events.  Write record of delay to
// data file, change alert state.
//=================================================================

void S_Ship::Alert(long msg,long s2, long u2,long aTF){

   // Get time, calculate delay since Base Control got message
   long s3, u3;
   RawTime(&s3, &u3);
   double delay = (s3 + u3 * 0.000001) - (s2 + u2 * 0.000001);

   switch (MyId) {
     case 0:
       Sdata0 << msg << " " << s3 << " " << u3 << " " << delay << endl;
       break;
     case 1:
       Sdata1 << msg << " " << s3 << " " << u3 << " " << delay << endl;
       break;
     case 2:
       Sdata2 << msg << " " << s3 << " " << u3 << " " << delay << endl;
       break;
     default:
       Sdata3 << msg << " " << s3 << " " << u3 << " " << delay << endl;
       break;
   }

    // Change state. Write to Logfile, schedule Ship Go
    RedAlert = aTF;

    RBout << MyId << ": " << TmTag(ttag) << " Ship_Alert event:"
          << msg << ", "  << s3 << ", " << u3 << ", "
       << " Alert status: " << RedAlert << ", " << SpGetTime() << endl;

    SCHEDULE_Ship_Go(SpGetTime() + 1.0, SpGetObjHandle());

}




//=====================================================================
// S_Ship::TransmitPrep
//  Handle Ship_TransmitPrep events
//  Adjust to prevent negative waits, annotate ship log, schedule TX
//=====================================================================

void S_Ship::TransmitPrep(double TXReqTime) {

    UpdateClockTime();

    double nextTransmitWaitTime = TXReqTime - SpGetTime();
    if (nextTransmitWaitTime < 0.0) nextTransmitWaitTime = 0.0;

    RBout << "LOG Ship " << SpGetSimObjGlobalId()
          << ": Recd TX request " << TXReqTime << " from Control"
          << " at Sim" << SpGetTime() << ", " << timetag << endl;
```

```
    SCHEDULE_Ship_Transmit(SpGetTime() + nextTransmitWaitTime,
                           SpGetObjHandle()); //schedule on self
}


//=====================================================================
// S_Ship::Transmit
//  Handle Ship_Transmit events
//  Update state variables, "transmit" position & heading to Base
//=====================================================================

void S_Ship::Transmit() {

    UpdateClockTime();

    double elapsedTime = SpGetTime() - LastTimePosUpdate;
    FindPos(elapsedTime);
    double dist = FindDist(Xpos, Ypos, BorderLen/2.0, BorderLen/2.0);
    TXState = 1;  //transmitting
    ++TimesTransmitted;

    RBout << "TX #"<< TimesTransmitted
          << ": Ship " << SpGetSimObjGlobalId()
          << " " << SpGetTime() << ", " << " Position: ("
          << Xpos << ", " << Ypos << ") Heading: ";
    if(Xdir == 1) RBout << "N"; else RBout << "S";
    if(Ydir == 1) RBout << "E"; else RBout << "W";
    RBout << "   Distance from base: " << dist << " " << timetag << endl;

   SCHEDULE_Ship_Go(SpGetTime()+ 1.0, SpGetObjHandle());//schedule on self
}


//=====================================================================
// S_Ship::Position_Send
//  Handle Ship_PosSend events
//  Update state variables, transmit position & heading to other ships
//=====================================================================

void S_Ship::Position_Send() {

   UpdateClockTime();

   double elapsedTime = SpGetTime() - LastTimePosUpdate;
   FindPos(elapsedTime);
   int MyId = SpGetSimObjGlobalId();

   RBout << "POSITION BROADCAST: " << ": Ship " << MyId
         << " " << SpGetTime()   << " Position: (" << Xpos << ", "
         << Ypos << ") Heading: ";
   if(Xdir == 1) RBout << "N"; else RBout << "S";
   if(Ydir == 1) RBout << "E"; else RBout << "W";
   RBout  << ", " << timetag << endl;


   for (int i = 0; i < NumShips; ++i) {
      SpObjHandle objHandle = SpGetObjHandle("S_Ship_MGR", i);
      if(i != MyId) {
         SCHEDULE_Ship_PosRecv(SpGetTime(), objHandle, SpGetTime(),
                               MyId, Xpos, Ypos, Xdir, Ydir);
    }
  }

  SCHEDULE_Ship_PosSend(SpGetTime()+ 100, SpGetObjHandle()); //self sche
}


//=====================================================================
// S_Ship::Position_Recv
// Handle Ship_PosRecv events
// Update state variables, calculate distance
```

```
//=============================================================================

void S_Ship::Position_Recv(double SentTime, int otherShip,
                           double otherXpos, double otherYpos,
                           int otherXdir, int otherYdir) {
   UpdateClockTime();

   double elapsedTime = SpGetTime() - LastTimePosUpdate;
   FindPos(elapsedTime);
   double dist = FindDist(Xpos, Ypos, otherXpos, otherYpos);

   RBout << "-Ship " << SpGetSimObjGlobalId()
         << ": " << SpGetTime() << " Data from Ship" << otherShip
         << " at Pos: (" <<otherXpos << ", " << otherYpos << ")"
         << ", Sent T:" << SentTime;
   RBout  << " Own Pos (" << Xpos << ", "
         << Ypos <<"), Distance: " << dist
          << ", " << timetag << endl;

   SCHEDULE_Ship_Go(SpGetTime(), SpGetObjHandle()); //schedule on self
}

//////////////////
/// utilities  ///
//////////////////

//=============================================================================
// S_Ship::FindPos
// Update state variables, reverse course if outside gaming area
//=============================================================================

void S_Ship::FindPos(double numsecs) {

   Xpos = Xpos + (Xdir * Velocity * numsecs);
   Ypos = Ypos + (Ydir * Velocity * numsecs);
   LastTimePosUpdate = SpGetTime();

   if (Xpos > N_edge) Xdir = -1;
   if (Xpos < S_edge) Xdir = 1;
   if (Ypos > E_edge) Ydir = -1;
   if (Ypos < W_edge) Ydir = 1;
}

//=============================================================================
// S_Ship::FindDist
// find distance between two points
//=============================================================================

double S_Ship::FindDist(double X1, double Y1, double X2, double Y2) {

   double dist = sqrt(((X1 - X2)*(X1 - X2))
                      +((Y1 - Y2)*(Y1 - Y2)));
   return dist;
}

//=============================================================================
// S_Ship::UpdateClockTime
// update wallclock time data members
//=============================================================================

void S_Ship::UpdateClockTime() {

   time_t       nowsec;
   int          nowusec;
   struct tm    *loc_tm;            // Ptr to tm time structure

   nowsec = time(0);
   struct timeval tv;
   struct timezone tz;
   gettimeofday(&tv, &tz);
   nowsec = tv.tv_sec;
```

```
    nowusec = tv.tv_usec;

    loc_tm = localtime( &nowsec);
    int hh = loc_tm->tm_hour;
    int mm = loc_tm->tm_min;
    int ss = loc_tm->tm_sec;
    long uu = tv.tv_usec;

    FillTimetag(hh, mm, ss, uu, timetag);
}
```

Code used to create VM Starter executable

```
============================================================
  MAJOR FILES:   /home/choaglun/asdf/virtualmach
============================================================
In tHIS Section:
-rw-rw-r--    1 choaglun choaglun     6729 Mar  5 15:42 driver.C
-rw-rw-r--    1 choaglun choaglun     9396 Mar  6 10:52 alarm_top.C
-rw-rw-r--    1 choaglun choaglun     6075 Mar  6 10:53 myalarm_mgrVM.C
-rw-rw-r--    1 choaglun choaglun     8342 Mar  4 16:51 extmgr.C
-rw-rw-r--    1 choaglun choaglun     1716 Mar  5 10:53 myalarm_mgrVM.H
------------------------------------------------------------------------------

/*********************************************************************/
// Program Name: driver.C                          for Sim Ver 12
// Executable: VM_starter
// Command: VM_starter
// Author: Cathy Hoaglund
//                                    Date Last Modified: 26 Feb 06
/*********************************************************************/
/* Description:
     This is the main program that sets up the Virtual Machine (v)
     comm mode using Parallel Virtual Machine (PVM). This code does
     the "parent" initial section, and forks off two child processes.
     one of which may be on a remote machine. The children execute
     the same code, but skip to the "child" secion. After they each
     unpack the brown bag lunch of data from the parent, the one on
     the local machine calls the external module manager code, and
     the remote one calls the alarm sensor program. The parent collects
     the output for all three and puts it out to stdout, and then
     waits till both children have finished.
*/
/*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <fstream>

#include "/home/pvm/pvm3/include/pvm3.h"

extern int extmgr(int tid, int other_child);

extern int alarm_top(int tid, int other_child, int mxm, int mhz);

extern void read_config_vm(int *mxm, int *mhz );

int  mxm, mhz;  // MaxMsgs, MsgHz

#define VMOUTFILE  "/home/choaglun/asdf/VM.out"
#define VMPROGRAM "/home/choaglun/asdf/VM_starter"


int main(int argc, char *argv[])
{
   int i,  retval;
   int tid;           // task id
   int parent_tid;    // tid of parent process or neg num if none
   ofstream outfile;  //diagnostics & status

   tid = pvm_mytid();  // obtain own task id
   if(tid < 0) {
      pvm_perror(argv[0]);
      return -1;
   }

/*============================ PARENT ============================*/
// get parent_tid while finding out if there is a parent.

   if((parent_tid = pvm_parent()) == PvmNoParent) {
```

```
// If true, this is initial process
outfile.open(VMOUTFILE);
if(! outfile) {
   perror("driver outfile open error on VM.out");
}

cout    << "Virtual Machine Parent has tid " << tid << endl;
outfile << "Virtual Machine Parent has tid " << tid << endl;
outfile.close();

// get config vars
int vals[3];

read_config_vm( &mxm, &mhz);
vals[1]=mxm;
vals[2]=mhz;
//cout << "Parent vals:\n";
//for (i=1; i < 3; i++) cout << vals[i] << " " << endl;

int child_tid;
int em_child, as_child;
int nhost, nprocs, narch;
struct pvmhostinfo *hostp;
int numt;

pvm_catchout(stdout); //redirect output of children to parent stdout

pvm_config(&nhost,&narch,&hostp); // get virtual machine configuration

nprocs = 2;                  // Set number of processes

pvm_initsend(PvmDataRaw);  // Send raw data since all architectures
                           // are compatible

//cout << "Parent about to spawn 2 processes " << endl;

// Spawn each child process
for(i=0; i<nprocs; i++) {
   cout << "Process " << i << "on host " <<(hostp+(i%nhost))->hi_name
        << endl;

   numt = pvm_spawn(VMPROGRAM,argv+1,
         PvmTaskHost,(hostp+(i%nhost))->hi_name,1,&child_tid);

   if(numt < 1) {
      cout << "Error: pvm_spawn returned" << numt << endl;
      exit(1);
   }
   if (i == 0)  em_child = child_tid;
   if (i == 1)  as_child = child_tid;
   vals[0] = i;

   // Send newborn's id number to the spawned process + config vals
   retval = pvm_psend(child_tid,0, vals, 3,PVM_INT);
   if(retval < 0) { pvm_perror("pvm_psend"), pvm_exit(); return -1;}
}

// Give children each other's address
retval = pvm_psend(em_child, 0, &as_child,1,PVM_INT);
if(retval < 0) { pvm_perror("pvm_psend"), pvm_exit(); return -1;}

retval = pvm_psend(as_child,0, &em_child,1,PVM_INT);
if(retval < 0) { pvm_perror("pvm_psend"), pvm_exit(); return -1;}

// Wait till all spawned processes have exited because of pvm_catchout
pvm_exit();

outfile.open(VMOUTFILE, ios::app);
if(! outfile) {
   perror("driver outfile open error on VM.out");
}
```

```
      cout    << "All spawned tasks have ended, Parent exiting" << endl;
      outfile << "All spawned tasks have ended, Parent exiting" << endl;
      outfile.close();
      exit(0);

   }   // end if no parent

/*===========END PARENT PROCESS, START CHILD ======================*/
//If we get here we are in a spawned process - there are 2 children

   outfile.open(VMOUTFILE, ios::app);
   if(! outfile) {
      perror("driver outfile open error on VM.out");
   }

   outfile << "Child: tid " << tid  << endl;
   cout    << "Child: tid " << tid  << endl;

   // Receive the child id number + config params
   int mynum;  // identifier - #0 does extmgr, #1 does alarmsensor
   int invals[4];

   retval = pvm_recv(parent_tid,0);
   if(retval < 0) {
      pvm_perror(argv[0]);pvm_exit(); exit(1);
   }

   // unpack message
   retval = pvm_upkint(invals, 3,1);
   if(retval < 0) {
      pvm_perror(argv[0]);pvm_exit(); exit(1);
   }

   mynum = invals[0];
   mxm   = invals[1];
   mhz   = invals[2];

   int nhost2, narch2;
   struct pvmhostinfo *hostp2;

   pvm_config(&nhost2,&narch2,&hostp2); // get virtual machine configuration

   outfile << tid << ": mynum " << mynum << " mxm " << mxm << " mhz " << mhz
           << " on host " << ((hostp2+(mynum%nhost2))->hi_name) << endl;

   cout    << tid << ": mynum " << mynum << " mxm " << mxm << " mhz " << mhz
           << " on host " <<((hostp2+(mynum%nhost2))->hi_name) << endl;


   // Receive the tid of the other child
   int other_child;

   retval = pvm_recv(parent_tid,0);
   if(retval < 0) {
      pvm_perror(argv[0]);pvm_exit(); exit(1);
   }

   // unpack message
   retval = pvm_upkint(&other_child,1,1);
   if(retval < 0) {
      pvm_perror(argv[0]);pvm_exit(); exit(1);
   }

   outfile << tid << ": " << "the other child tid = " << other_child << endl;
   cout    << tid << ": " << "the other child tid = " << other_child << endl;

   if(mynum == 0) {
      outfile << tid << ": starting extmgr here " << endl;
      retval = extmgr(tid, other_child);
```

```
   } else if (mynum == 1) {
      outfile << tid << ": starting alarm here " << endl;
      retval = alarm_top (tid, other_child, mxm, mhz);

   } else {
      outfile << tid << ": mynum is and I have no job!" << mynum << endl;
   }

   outfile.close();
   return 0;

}
```

```
/*****************************************************************/
// Program Name: alarm_top.C             for Sim Ver 12
// Executable: VM_starter
//
//   This file contains 3 functions that inplement the Alarm Sensor
// data generator for the Virtual Machine configuratiom, where the
// main program is a PVM parent (in driver.C) that spawns two
// child processes, one of which starts the alarm sequence
//   Classes & Functions:
//     alarm_top - equivalent to main in other comm modes
//     timer_mgr - uses signal to wake up and send messages
//             uses class Array_mgrVM
//     read_config_vm - called by parent to read config file
//       and extract relevant parameters
//
// Author: Cathy Hoaglund
//                                    Date Last Modified: 19 Feb 06
/*****************************************************************/

#include <string.h>
#include <stdlib.h>
#include <iostream.h>
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/time.h>

#include "myalarm_mgrVM.H"

#include "../utils/TimeUtil.h"
#include "/home/pvm/pvm3/include/pvm3.h"

#define ASOUTFILE  "/home/choaglun/asdf/AS.out"
#define CONFIGFILE "/home/choaglun/asdf/config/config.par"


// --- Global variables
int  tid;                    // PVM task id
int  em_tid;                // PVM task id for extmgr

// set from config.par for use by timer_mgr
int  MaxMsgs =30;            // number of messages expected
int  MsgHz = 2;               // rate for msg generation
int  Interval = 5;           // in microseconds

// other
int  ArrSize = 4;            // number of array vals to send{4..500)
int prior = -1;             // last message acknowledged
int done = 0;               // flag, 1 when no more msgs

Array_mgrVM am;             // instantiate the Array Manager class
```

```
void timer_mgr(int sigval);


//=========================================================================
// alarm_top replaces main in the alarm sensor program. It gets the
// input task ids from the spawned PVM process, registers timer_mgr
// as the signal manager for SIGALARM. It then raises SIGALRM
// and then waits till timer_mgr says we are done.
//=========================================================================

int alarm_top (int tid, int em_tid, int mxm, int mhz)
{

  cout << "Alarm_top input parameters: mytid = " << tid
            << ", other tid = " << em_tid << endl << "mxm = "
            << mxm << ",  mhz = " << mhz  << endl;

   Interval = 1000000/mhz;
   MaxMsgs = mxm;
   MsgHz   = mhz;

   //open output file
   ofstream outfile;
   outfile.open(ASOUTFILE, ios::out);
   if(! outfile){
      perror("alarm_top outfile open error");
      return(-1);
   }
   else {
      outfile << "Alarm_top configuration parameters: mytid = " << tid
              << ", other tid = " << em_tid << endl << "MaxMsgs = "
             << MaxMsgs << ",  MsgHz = " << MsgHz << ", Interval = "
             <<Interval << endl;

      cout << "Alarm_top configuration parameters: mytid = " << tid
             << ", other tid = " << em_tid << endl << "MaxMsgs = "
             << MaxMsgs << ",  MsgHz = " << MsgHz << ", Interval = "
             <<Interval << endl;
    }

    // Record starting time
    long hh, mm, ss, uu;
    char timetag[16] = "00:00:00.000000";
    DoTimetag(&hh, &mm, &ss, &uu, timetag);
    outfile << timetag << " Alarm_top Starting initialization" << endl;
    outfile.close();

    //Array_mgrVM am() is a class to handle array detail & communication
    // Instantiated globally to be available to signal handler

    am.start_Array_mgr( Interval, tid, em_tid);

    signal(SIGALRM,timer_mgr);
    raise(SIGALRM);

    //Wait till signal processing is over
    while (!done);

    exit (0);
}


//=========================================================================
// timer_mgr: Handles the repetive alarm function by using the
//   ITIMER_REAL to raise SIGALRM for every Interval number of usecs.
//   It uses the Array_mgr class to generate data and take
//   care of transmitting it to the External Module process.
//=========================================================================
```

```
void timer_mgr(int sigval)
{
  static int simtime =0;
  static long int msg_num =0;
  long h, m,s, u;
  char timetag[16]="00:00:00.000";

  struct itimerval t; //declare & initialize time structure
  t.it_interval.tv_usec = Interval;
  t.it_interval.tv_sec = 0;
  t.it_value.tv_usec = Interval;
  t.it_value.tv_sec = 0;

  int maxtime = MaxMsgs * Interval;
  int errstat = 0;

   //open output file
   ofstream outfile;
   outfile.open(ASOUTFILE, ios::app);
   if(! outfile){
      perror("timer_mgr outfile open error");
      return;
   }
   outfile << "timer_mgr maxtime:" << maxtime << " Interval "
        << Interval << " MaxMsgs "<< MaxMsgs << endl;

  // *********Do init to allow Sim to get going, init comm *********
  if (simtime == 0) {

    errstat = am.init();

    if (errstat != 0) {
       done = 1;
       return;
    }

    // We have init success, start message processing
    errstat = am.send(++msg_num);

    DoTimetag(&h, &m, &s, &u, timetag);

    if (errstat != 0) {
       outfile << timetag << " timer_mgr: Lost msg " << msg_num << endl;
       cout << timetag << " timer_mgr: Lost msg " << msg_num << endl;
    }

    // Start timer
    simtime = simtime + Interval;
    outfile << timetag << " timer_mgr, after msg " << msg_num
          << " simtime = " << simtime << " Interval = " << Interval << endl;

    static sigset_t sigmask;
    sigemptyset(&sigmask);
    if (sigaddset(&sigmask,SIGALRM)==-1 ||
        sigprocmask(SIG_SETMASK, &sigmask, 0)==-1)
      perror ("set signal mask");

    signal(SIGALRM, timer_mgr);
    setitimer(ITIMER_REAL, &t, 0);
  }

  //********* Normal interval processing till maxtime *************
  else if (simtime < maxtime) {
    errstat = am.send(++msg_num);

    DoTimetag(&h, &m, &s, &u, timetag);

    if (errstat != 0) {
       outfile << timetag << " timer_mgr lost msg "<< msg_num <<endl;
       cout << timetag << " timer_mgr lost msg "<< msg_num <<endl;
    }
```

```
    simtime = simtime + Interval;

    static sigset_t sigmask;
    sigemptyset(&sigmask);
    if (sigaddset(&sigmask,SIGALRM)==-1 ||
        sigprocmask(SIG_SETMASK, &sigmask, 0)==-1)
      perror ("set signal mask");

    signal(SIGALRM, timer_mgr);
    setitimer(ITIMER_REAL, &t, 0);
  }

  //********* Simtime >= maxtime *************
  else {
     DoTimetag(&h, &m, &s, &u, timetag);

     am.endmsg();      //release any resources

     DoTimetag(&h, &m, &s, &u, timetag);
     outfile << timetag << " ** END MESSAGE GENERATION **" <<endl;
     cout << timetag << " ** END MESSAGE GENERATION **" <<endl;

    t.it_interval.tv_usec = 0;
    t.it_value.tv_usec = 0;
    setitimer(ITIMER_REAL, &t, 0);
    done = 1;
  }

  outfile.close();

  return;
}


//=========================================================================
// read_config: get variables of interest to the alarm sensor
// process from the config.par file
//=========================================================================

void read_config_vm( int *mxm, int *mhz)
{
    // Local -read and discard
    char CommType;           // comm mode (m, s, v, r)
    char nrisk;              // L, M, or H
    int  simrate;           // not used by alarmsensor
    int  physconf;          // not used by alarmsensor
    int  nnodes;             // not used by alarmsensor
    int  ChkHz;                    // not used by alarmsensor

    // Global -read and store for use by timer
    //int  MaxMsgs;          //number of messages expected
    //int  ArrSize;         // number of array vals to send{4..500)
    //int  MsgHz;          // rate for msg generation
    //int  Interval;        // in microseconds

   // Read config file
   ifstream configfile;
   configfile.open(CONFIGFILE, ios::in);
   if(! configfile){
      perror("alarm_top configuration file read error\n");
   }
   else {
      configfile >>  CommType;
      configfile >>  MaxMsgs;
      configfile >>  nrisk;
      configfile >>  simrate;
      configfile >>  physconf;
      configfile >>  nnodes;
      configfile >>  MsgHz;
```

```
        configfile >>  ChkHz;

        cout << "alarm_top: config.par - CommType: " << CommType
             << " MaxMsgs: " << MaxMsgs
             << " nrisk:"    << nrisk << endl
             << " simrate: " << simrate
             << " physconf: " << physconf
             << " nnodes: "  << nnodes
             << " MsgHz: " << MsgHz
             << " ChkHz: "   << ChkHz << endl ;

        configfile.close();

        Interval = -1;
        if (MsgHz != 0) Interval = 1000000 / MsgHz;

        // Also return parameters
        *mxm = MaxMsgs;
        *mhz = MsgHz;

    }
}
//-------------------------------------------------------



/*********************************************************************/
// Program Name: myalarm_mgrVM.cpp             for Sim Ver 12
// Executable: VMStarter
// Class: Array_Mgr
// Author: Cathy Hoaglund
//
//                                    Date Last Modified: 5 Mar 06
/*********************************************************************/
/* Description:
    The alarm process alarm_top uses the Array_Mgr class to fill
    and transmit a message array, usinga virtual machine under PVM,

   Classes & functions in AlarmSensor application:
       Signals are handled by timer_mgr, which uses the setitimer
       interval timer to create a cycle with microsecond resolution.

       Class Array_mgr member functions:
        fill_array() fills an array with counter,time, and data values
        init() calls  init_vm()
        send() calls send_vm()
        endmsg() does nothing

*/
/*********************************************************************/

#include <ctype.h>
#include <fcntl.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#include <sys/time.h>
#include <sys/ipc.h>
#include <sys/shm.h>


#include "myalarm_mgrVM.H"
#include "../utils/TimeUtil.h"
#include "/home/pvm/pvm3/include/pvm3.h"
```

```
#define ASOUTFILE  "/home/choaglun/asdf/AS.out"

//====================================================================
// Constructor & Destructor
//====================================================================

Array_mgrVM::Array_mgrVM()
{
   h = m = s = u = 0;
   for (int i = 0; i < 500; i++)  arr[i] = 0;
   arr_size = 4;
   msg = secs = usecs = alien = 0;
}

Array_mgrVM::~Array_mgrVM()
{
   outfile.close();
}

//====================================================================
// Array_mgrVM:: start_Array_mgr
//     initializes data members associated with config files,
//     opens diagnostic log file
//====================================================================

void Array_mgrVM::start_Array_mgr( int inMsgIntv, int intid, int inem_tid )
{
   interval = inMsgIntv;
   tid      = intid;
   em_tid   = inem_tid;

   outfile.open(ASOUTFILE, ios::app);
   if(! outfile) {
      perror("start_Array_mgr outfile open error on AS.out");
   }

}

//====================================================================
// Array_mgrVM::fill_array:
//     fills an array with id, time, and a state values, then
//     pads balance
//====================================================================

void Array_mgrVM::fill_array()
{
   alien = msg % 2;
   TellTime(&h, &m, &s, &u);
   RawTime(&secs, &usecs);

   arr[0] = msg;
   arr[1] = secs;
   arr[2] = usecs;
   arr[3] = alien;
   for (int i = 4; i < 500; i++) arr[i] = i;

}

//====================================================================
// Array_mgrVM::init
//  Chooses correct init actions based on Comm Type
//  returns 0 on success, -1 on error
//====================================================================
int Array_mgrVM::init()
{
   int errstat;

   errstat = init_vm();

  return(errstat);
```

```
}

//=====================================================================
// Array_mgrVM::send
//  Chooses correct send actions based on Comm Type
//  returns 0 on success, -1 on error
//=====================================================================
int Array_mgrVM::send(long msg_num)
{
   int errstat;

      errstat = send_vm(msg_num);

   return(errstat);
}


//=====================================================================
// Array_mgrVM::endmsg
//  Close outfile
// returns 0 on success, -1 on error
//=====================================================================
int Array_mgrVM::endmsg()
{
   outfile.close();

   return(0);
}


//=====================================================================
// Array_mgrVM::init_vm
//
//  returns 0 on success, -1 on error
//=====================================================================

int Array_mgrVM::init_vm()
{
   outfile << tid << ": " << TmTag(ttag) <<" *** AM init_vm ***" << endl;

   //wait for ok from extmgr
   int info;
   int val;

   info = pvm_recv(em_tid,0);
   if(info < 0) {
      pvm_perror("init_vm");pvm_exit(); exit(1);
   }

   // unpack message
   info = pvm_upkint(&val,1,1);
   if(info < 0) {
      pvm_perror("init_vm");pvm_exit(); exit(1);
   }

   outfile << tid << ": " << TmTag(ttag) << " *** AM init complete" << endl;

   return(0);
}


//=====================================================================
// Array_mgrVM::send_vm
//    Uses pvm_psend, which packs and sends all in one action.
//    pvm_psend is non-blocking and does not requre an ack.
//  returns 0 on success, -1 on error
//=====================================================================

int Array_mgrVM::send_vm(long msg_num)
{
```

```
   int info;

   msg = msg_num;

   fill_array();

   // send array values using asynchronous, non-blocking send

   long arr2[4];  int i;
   for (i=0; i < 4; i++){
       arr2[i] = arr[i];
   }

   info = pvm_psend(em_tid,0, &arr2, 4, PVM_LONG);
   if(info < 0) { pvm_perror("send_vm pvm_psend"), pvm_exit(); return -1;}

   return(0);
}
```

```
// ********************************************************************
// Name: extmgr                                    for Sim version 12
//    External Module to send data into SPEEDES simulation under PVM
// Author: Cathy Hoaglund                Last modified: 17 Feb 06
//
// Executable used in virtual machine comm mode as a spawned PVM task
// that reads data from messages sent by a sibling task runnung thr
// AlarmSensor process. It then packages up the data
// into a string to get passed in when send_command schedules an event.
//
// The main process manages the connection with the SPEEDES Host Router
// and SpeedesServer. It leaves the incoming messages to emm,an
// instance of class ExternalModuleMgr, which also writes data to a
// file with times.
// ********************************************************************

#include <fstream>

#include <SpIostream.H>
#include <SpStateMgr.H>
#include <SpDataParser.H>
#include <SpStateMgrEvent.H>
#include <SpFreeObjProxy.H>

#include "../external/MyReflect.H"
#include "../utils/TimeUtil.h"
#include "../external/ExternalModuleMgr.H"

//minimal constructor to connect to GVT
 SpFreeObjProxy::SpFreeObjProxy(int n) {set_ntypes(n);}

#define WAIT4MSG 1.0    // Time to wait for message each cycle
#define TIMELAG  10.0   // How far behind the Sim in virtual seconds
                        // we can lag. Effectively, virtual frame length

#define EMMOUTFILE "/home/choaglun/asdf/emm.out"
#define EMMDATFILE "/home/choaglun/asdf/data/EM.dat"
#define CONFIGFILE "/home/choaglun/asdf/config/config.par"
#define GOFILE     "/home/choaglun/asdf/OK2Go.txt"
#define ENDFILE    "/home/choaglun/asdf/OK2End.txt"


//--------------------------------------------------------------------
// For virtual machine case replaced main in extmod1 with extmgr
//--------------------------------------------------------------------

int extmgr(int tid, int as_tid ) {
   // tid = pvm task id, as_tid is the tid for Alarm_sensor

   ofstream outfile;
```

```
  outfile.open(EMMOUTFILE);
  if(!outfile){
     perror("extmgr outfile creation error 1st open");
  }
  else {
     outfile.close();
  }
  outfile.open(EMMOUTFILE, ios::app);
  if(!outfile){
     perror("extmgr outfile open error 2nd open");
  }

  //Initialize configuration variables
  char CommType = ' ';        // 3 valid types: m, s, v
  int  MaxMsgs = 0;
  char nrisk = ' ';
  int  simrate = 0;
  int  nnodes = 0;
  int  physconf = 0;
  int  MsgHz = 1;
  int  ChkHz = 1;

  int  ArrSize = 4;                // no longe input
  int  Interval = 100000;
  int  ChkPeriod = 10000;

  // Read config file
  ifstream configfile;
  configfile.open(CONFIGFILE);
  if(!configfile){
     cerr << "extmgr configuration file read error\n";
  }
  else {
     configfile >>  CommType;
     configfile >>  MaxMsgs;
     configfile >>  nrisk;
     configfile >>  simrate;
     configfile >>  physconf;
     configfile >>  nnodes;
     configfile >>  MsgHz;
     configfile >>  ChkHz;

/*
     outfile << "extmgr: config.par- CommType: " << CommType
             << ", MaxMsgs: " << MaxMsgs << ", nrisk:" << nrisk
             << ", simrate: " << simrate  << ", physconf " << physconf
             << ", nnodes: "  << nnodes  << ", MsgHz: " << MsgHz
             << ", ChkHz: "   << ChkHz  << endl;
*/
     configfile.close();
     if (MsgHz != 0) Interval = 1000000 / MsgHz;
     if (ChkHz != 0) ChkPeriod = 1000000 / ChkHz;
/*
     outfile << "extmgr: Message Interval " << Interval << endl;
     outfile << "extmgr: Message check period = " << ChkPeriod << endl;
*/
  }

  // Timing variables & sim state variables
  char ttag[16]="00:00:00.000";  //timetag string
  char datastring[60]="  ";
  int  dl = strlen(datastring);
  long msg = 0;
  long lastmsg = 0;

  outfile << endl << TmTag(ttag) << " External Module running " << endl;
 // cout    << endl << ttag        << " External Module running " << endl;
  cout << setprecision(6);

  //--- Connect with Speedes Server
  outfile << TmTag(ttag) << " Connecting to Speedes... " << endl;
```

```
cout    << ttag        << " Connecting to Speedes... " << endl;
double        timeLag      = TIMELAG;
SpDataParser  speedesDotPar("speedes.par");
SpStateMgr    stateManager(&speedesDotPar, timeLag);

stateManager.GoToTime(0.0);  //Blocking call
outfile << TmTag(ttag) << " ExternalModule finished GoToTime(0.0)" << endl;
cout    <<       ttag  << " ExternalModule finished GoToTime(0.0)" << endl;

// Instantiate class, do initial synchronization
ExternalModuleMgr emm(CommType, Interval, ChkPeriod, ArrSize, tid, as_tid);

ofstream Gofile (GOFILE, ios::out);
if(!Gofile){
   cerr << "Couldn't create OK2Go.txt " << endl;
   return(-1);
}else {
   Gofile << TmTag(ttag) << endl;
   Gofile.close();
}

// check for message 1 before start of loop
msg = emm.get_msg(datastring);
outfile << "first read returned msg number " << msg
     << ", datastring: " << datastring << " dl:" << dl << endl;

// make sure sim is ready
if (stateManager.SpeedesExecuting() != 1) {
   outfile << "Speedes quit - exiting" << endl;
   cerr    << "Speedes quit - exiting" << endl;
   exit(0);
}

//--- Loop till Sim ends or no more messages
while (lastmsg < MaxMsgs+1){

  // If we have an unprocessed message, forward data to sim
  if (msg != 0) {
     dl = strlen(datastring);
     //outfile << "unprocesed msg " << msg << " datastring: "
     //        << datastring << " dl:" << dl << endl;

     stateManager.SendCommand("Control_SeeAlien",
                          "S_Control_MGR 0", datastring, dl) ;

     // Compute delay for hop1, write data record
     emm.write_record();

     // Send update to console & get next Sim time if still running
     if (stateManager.SpeedesExecuting() == 1) {
        outfile << "\n extmgr: Update Message " << msg << " sent at "
             << TmTag(ttag)  << endl
             << "  Current " << stateManager.GetCurrentTime()
             << " Granted " << stateManager.GetGrantedTime() << endl;

        cout << "\n extmgr: Update Message " << msg << " sent at "
             <<  ttag  << endl
             << "  Current " << stateManager.GetCurrentTime()
             << " Granted " << stateManager.GetGrantedTime() << endl;

         stateManager.GoToTime(stateManager.GetCurrentTime()+ timeLag);
     }

     lastmsg = msg;
     msg = 0;
  }
  else { // No message, just advance time
     if (stateManager.SpeedesExecuting() != 1) {
        outfile << "Speedes quit - exiting" << endl;
        cerr    << "Speedes quit - exiting" << endl;
        exit(0);
```

```
        }

        //outfile << "\n" << TmTag(ttag) << " extmgr: No new message.\n "
        //     <<      "\tSim Time Current " << stateManager.GetCurrentTime()
        //     << " Granted " << stateManager.GetGrantedTime() << endl;
        //cout << "\n" << TmTag(ttag) << " extmgr: No new message." << endl;

        stateManager.GoToTime(stateManager.GetCurrentTime() + timeLag);

      }

      //debug
      if(msg != 0) {
          cerr << "ERROR in extmgr, resetting msg to 0!!\n";
          msg = 0;
      }

      //If not at max msgs, check for message till read or timeout
      if (lastmsg < MaxMsgs) {

        int waittime = 0;
        while ((msg == 0) && (waittime < Interval)) {
           usleep(ChkPeriod);
           msg = emm.get_msg(datastring);
          // outfile << "read returned msg number " << msg
          //         << ", datastring: " << datastring << endl;

           waittime = waittime + ChkPeriod;
        }
      }

      // If we are at max messages, close comm channel
      if (lastmsg >= MaxMsgs) {
         emm.end_msgs();
         lastmsg = MaxMsgs + 1;

         ofstream Endfile (ENDFILE, ios::out);
         if(!Endfile){
            cerr << "Could not create OK2End.txt \n";
         }else {
            Endfile << "Done" << endl;
            Endfile.close();
         }
      }
      // go check in with sim, msg or not ...

   } // end primary while loop

    return 0;

}
```

```
// ********************************************************************
// Name: myalarm_mgrVM.H                                for Sim version 12
//
// Author: Cathy Hoaglund                  Last modified: 18 Feb 06
//
// ********************************************************************

#ifndef MYALARMMGRVM_H
#define MYALARMMGRVM_H

#include <fstream.h>

//=====================================================================
// Array_mgrVM
//=====================================================================
```

```
class Array_mgrVM
{
   public:
                  Array_mgrVM();
                  ~Array_mgrVM();
       void       start_Array_mgr(int inMsgIntv, int intid, int inem_tid);
                    //set data members per config

       void       fill_array();          //timetag and fill
       int        init();                  //choose correct init
       int        send(long msg_num);      //choose correct send
       int        endmsg();                //choose correct end

       int        init_vm();
       int        send_vm(long msg_num);

   private:
       ofstream   outfile;            //diagnostics & status
       int        interval;           //microsecs between messages
       char       ttag[16];           //timetag
        long       h, m, s, u;        //hrs, mins, secs, microsecs


     //sim-specific

        long       arr[500];
       int        arr_size;       //how many elements to use [4..500]
                                  //determines packet size in mode m

       long       msg;            // arr[0], msg_num
       long       secs;           // arr[1], message time pt 1
       long       usecs;          // arr[2], message time pt 2
       long       alien;          // arr[3], alien detection T/F

    // vm variables
        int        tid;            // PVM task id
       int        em_tid;         // PVM task id of extmgr

};

#endif
// MyReflect.H
#ifndef MyReflect_H
#define MyReflect_H

#include "SpStateMgrEvent.H"

class MyReflect: public SpStateMgrEvent {
  public:
    MyReflect() {}
    virtual ~MyReflect() {}
    virtual void Process() {
      cout << "MyReflect Process at " << GetTimeTag()
           << ", Global Id: " << GetSimObjGlobalId()
           << ", Event Name: " << GetEventName() << endl;
    }
};

//static void* NewMyReflect(int n) {
//  return new MyReflect[n];
//}
#endif
```

D2. Supporting Source Code Files

```
============================================================
    asdf source code, config, makefile, and script files
    TOP LEVEL:   /home/choaglun/asdf
============================================================

drwxrwxr-x    2 choaglun choaglun     4096 Apr 22 16:39 alarmsensor
drwxrwxr-x    2 choaglun choaglun     4096 Mar 10 11:15 config
drwxrwxr-x    2 choaglun choaglun     4096 Apr 22 16:42 data
drwxrwxr-x    2 choaglun choaglun     4096 Jan 22 16:48 external
drwxrwxr-x    2 choaglun choaglun     4096 Jan 22 16:48 harvester
-rw-rw-r--    1 choaglun choaglun      508 Feb 11 09:35 Makefile
drwxrwxr-x    2 choaglun choaglun     4096 Jan 22 16:48 mysim
-rwxrwxr-x    1 choaglun choaglun     3321 Mar 24 15:46 paramscript1
-rwxr-xr-x    1 choaglun choaglun      520 Feb 17 15:19 runSS
drwxrwxr-x    2 choaglun choaglun     4096 Apr 22 16:47 utils
drwxrwxr-x    2 choaglun choaglun     4096 Mar 10 09:19 virtualmach


------------------------------------------------------------------
# Makefile for ASDF executables
# modified by Cathy Hoaglund                    11 Feb 06

PVM_ROOT = /home/pvm3
PVM_ARCH = LINUX

APPS       = alarmsensor external harvester mysim virtualmach

APP_MFLAGS =  $(MFLAGS) PVM_ROOT=$(PVM_ROOT) \
          PVM_ARCH=$(PVM_ARCH)

all:
       @for i in $(APPS) ;\
       do \
       (cd $$i ; echo -- making all in $$i; \
       $(MAKE) $(APP_MFLAGS) all; echo ""); \
       done

clean:
       @for i in $(APPS) ;\
       do \
       (cd $$i ; echo -- making clean in $$i; \
       $(MAKE) $(APP_MFLAGS) clean; echo ""); \
       done
------------------------------------------------------------

#!/bin/bash
#=====================================================================
# Name: paramscript1                              for Sim Version 12
# Command line:
#   paramscript1 tdir cmode nnodes nrisk msgHz chkHz tnum
#
#   Bash script to execute sim on laptop & prep files for Harvester
#   Uses command line parameters for frequently changed parameters
#   in lieu of the text editing mode in goscript or just1script
#   variables are entered on the command line.
#
#   Assumes SpeedesServer (runSS) is running for all modes, that
#   Listener, the rpc server, is running in message passing mode, and
#   a PVM Virtual machine is running for the virtual machine
#   configuration.
#
#                                 Basic design last modified 24 Mar 06
#   comm type can be [m, v, s]
#     If m, need to start EMMain locally and AlarmSensor remotely
#     If v, start the PVM-based version
#     If s, all executables are on local machine
#   nrisk time mgmt code can be [L = BTB, M = BTW, H = TW]
#   simrate can be 50, 100, 150, 200, or -1 to have it be a random
```

```
#          number between 10 and 100 if it is to be variable.
#   number of sim nodes can be [1,2,5]
#   physconf code [10: Dell only, 30: raven3 only; 40: raven4 only
#         50: raven5 only
#         54: raven5 sending, raven4 recving, with EM & sim on 4]
#         45: raven4 sending, raven5 recving, with EM & sim on 5]
#
#=====================================================================


if [ $1 = "-h" ]; then
   echo "Usage:"
   echo "paramscript1 TestDir CommType NumNodes Nrisk MsgHz ChkHz TestRunid"
   exit
elif [ $1 = "-H" ] ; then
   echo "Usage:"
   echo "paramscript1 TestDir CommType NumNodes Nrisk MsgHz ChkHz TestRunid"
   echo "TestDir is a directory name like Feb20"
   echo "CommType may be m, s, or v"
   echo "NumNodes may be 1 or 2 "
   echo "Nrisk may be L (BTB), M (BTW), or H (TW)"
   echo "MsgHz may be 1, 2, 10"
   echo "ChkHz may be 10 or 50"
   exit  :
fi


tdir=$1
cmode=$2
nnodes=$3
nrisk=$4
msgHz=$5
chkHz=$6
tnum=$7

#--SETUP: static over this set of runs ----
physconf=10
msgnum=30
simrate=10

utils/staticscript $nrisk

thename=$HOSTNAME


if   [ $thename = "raven3" ] && [ $cmode = "s" ]; then
  physconf=30
  utils/runscript1 $cmode $msgnum $nrisk $simrate $physconf $tdir $tnum $nnodes $msgHz
$chkHz |tee sim.out
elif [ $thename = "raven3" ]; then
  physconf=43
  utils/runscript $cmode $msgnum $nrisk $simrate $physconf $tdir $tnum $nnodes $msgHz
$chkHz  |tee sim.out
elif [ $thename = "raven4" ] && [ $cmode = "s" ]; then
  physconf=40
  utils/runscript1 $cmode $msgnum $nrisk $simrate $physconf $tdir $tnum $nnodes $msgHz
$chkHz |tee sim.out
elif [ $thename = "raven4" ]; then
  physconf=54
  utils/runscript $cmode $msgnum $nrisk $simrate $physconf $tdir $tnum $nnodes $msgHz
$chkHz |tee sim.out
elif [ $thename = "raven5" ] && [ $cmode = "s" ]; then
  physconf=50
  utils/runscript1 $cmode $msgnum $nrisk $simrate $physconf $tdir $tnum $nnodes $msgHz
$chkHz |tee sim.out
elif [ $thename = "raven5" ]; then
  physconf=45
  utils/runscript $cmode $msgnum $nrisk $simrate $physconf $tdir $tnum $nnodes $msgHz
$chkHz |tee sim.out
else
  physconf=10
```

```
  utils/runscript1 $cmode $msgnum $nrisk $simrate $physconf $tdir $tnum $nnodes $msgHz
$chkHz |tee sim.out
fi


------------------------------------------------------------------------
#=======================================================================
# Name: runSS                          for MySim - all versions
#   Bash script to execute SpeedesServer
# Author: Cathy Hoaglund               Last modified 16 Dec 05
#=======================================================================

# invoke as a background task
#xterm -geometry 40x10+0+0 -e
/home/speedes/speedes2.0.1/exe/ArchitectureDirs/Linux_i686/SpeedesServer
/home/speedes/speedes2.0.1/exe/ArchitectureDirs/Linux_i686/SpeedesServer




============================================================
    SUPPORTING FILES:   /home/choaglun/asdf/alarmsensor
============================================================
In Major Routines Section:
-rw-rw-r--    1 choaglun choaglun    10584 Feb 25 17:30 listener.cpp
-rw-rw-r--    1 choaglun choaglun     7910 Mar  6 10:46 myalarm_main.cpp
-rw-rw-r--    1 choaglun choaglun    13464 Mar  6 11:01 myalarm_mgr.cpp
-rw-rw-r--    1 choaglun choaglun     2138 Feb 25 05:29 myalarm_mgr.h
In This Section:
-rw-rw-r--    1 choaglun choaglun     1914 Feb 25 06:25 makefile
-rw-r--r--    1 choaglun choaglun      562 Dec 17 14:56 myalarm_clnt.cpp
-rw-r--r--    1 choaglun choaglun      981 Dec 17 14:57 myalarm.h
-rw-r--r--    1 choaglun choaglun     2121 Dec 17 14:56 myalarm_svc.cpp
-rwx------    1 choaglun choaglun      519 Dec 17 14:57 myalarm.x
-rw-r--r--    1 choaglun choaglun      324 Dec 17 14:56 myalarm_xdr.cpp
------------------------------------------------------------------------


# makefile for alarmsensor directory
# This was originally generated by rpcgen but was
# substantially modified by Cathy Hoaglund - last changed 4/6/05

# Parameters
CLIENT = ../AlarmSensor
SERVER = ../Listener

SOURCES.x = myalarm.x

UTILDIR = /home/speedes/speedes2.0.1/cathy/utils

TARGETS_CLNT.cpp = myalarm_clnt.cpp myalarm_main.cpp myalarm_mgr.cpp
TARGETS_SVC.cpp = myalarm_svc.cpp listener.cpp
TARGETS_MISC.cpp = myalarm_xdr.cpp
TARGETS_TIME.cpp = $(UTILDIR)/TimeUtil.cpp

HEADERS = myalarm.h  myalarm_mgr.h $(UTILDIR)/ShmUtil.h $(UTILDIR)/TimeUtil.h

OBJECTS_CLNT = myalarm_clnt.o myalarm_main.o myalarm_mgr.o
OBJECTS_SVC = myalarm_svc.o listener.o
OBJECTS_MISC = myalarm_xdr.o
OBJECTS_TIME = TimeUtil.o

# Compiler flags

CFLAGS += -g
LDLIBS += -lnsl
RPCGENFLAGS =
```

```
# Targets

all : $(CLIENT) $(SERVER)

#$(TARGETS) : $(SOURCES.x) ---- if ever needed reconstruct
#       rpcgen $(RPCGENFLAGS) $(SOURCES.x)

$(OBJECTS_TIME) : $(TARGETS_TIME.cpp) $(UTILDIR)/TimeUtil.h
        i386-redhat-linux7-g++ -c $(TARGETS_TIME.cpp)
        @echo "made TimeUtil.o"

$(OBJECTS_MISC) : $(TARGETS_MISC.cpp) $(HEADERS)
        i386-redhat-linux7-g++ -c $(TARGETS_MISC.cpp)

$(OBJECTS_CLNT) : $(TARGETS_CLNT.cpp) $(HEADERS)
        i386-redhat-linux7-g++ -c $(TARGETS_CLNT.cpp)

$(OBJECTS_SVC) : $(TARGETS_SVC.cpp) $(TARGETS_MISC.cpp) $(HEADERS)
        i386-redhat-linux7-g++ -c $(TARGETS_SVC.cpp) $(TARGETS_MISC.cpp)

$(CLIENT) : $(OBJECTS_CLNT) $(OBJECTS_MISC) $(OBJECTS_TIME)
        i386-redhat-linux7-g++ -o $(CLIENT) $(OBJECTS_CLNT) $(OBJECTS_MISC)
$(OBJECTS_TIME) \
                                /usr/lib/libpthread.so  $(LDLIBS)

$(SERVER) : $(OBJECTS_SVC) $(OBJECTS_MISC) $(OBJECTS_TIME)
        i386-redhat-linux7-g++ -o $(SERVER) $(OBJECTS_SVC) $(OBJECTS_MISC) $(OBJECTS_TIME)
\
                                /usr/lib/libpthread.so  $(LDLIBS)

 clean:
         $(RM) core $(OBJECTS_CLNT) $(OBJECTS_SVC) $(OBJECTS_MISC) $(OBJECTS_TIME)
$(CLIENT) $(SERVER)
```

---

myalarm_clnt.cpp

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <memory.h> /* for memset */
#include "myalarm.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

values *
myalarm_repl_1(values *argp, CLIENT *clnt)
{
        static values clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, MYALARM_REPL,
                (xdrproc_t) xdr_values, (caddr_t) argp,
                (xdrproc_t) xdr_values, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

---

my_alarm.h

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _MYALARM_H_RPCGEN
```

```
#define _MYALARM_H_RPCGEN

#include <rpc/rpc.h>


#ifdef __cplusplus
extern "C" {
#endif


typedef struct {
       u_int values_len;
       long int *values_val;
} values;

#define MYALARM 0x40000000
#define MYALARM_VER 1

#if defined(__STDC__) || defined(__cplusplus)
#define MYALARM_REPL 1
extern  values * myalarm_repl_1(values *, CLIENT *);
extern  values * myalarm_repl_1_svc(values *, struct svc_req *);
extern int myalarm_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
#define MYALARM_REPL 1
extern  values * myalarm_repl_1();
extern  values * myalarm_repl_1_svc();
extern int myalarm_1_freeresult ();
#endif /* K&R C */

/* the xdr functions */

#if defined(__STDC__) || defined(__cplusplus)
extern  bool_t xdr_values (XDR *, values*);

#else /* K&R C */
extern bool_t xdr_values ();

#endif /* K&R C */

#ifdef __cplusplus
}
#endif

#endif /* !_MYALARM_H_RPCGEN */
```

---

myalarm_svc.cpp

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "myalarm.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifndef SIG_PF
#define SIG_PF void(*)(int)
#endif

static void
myalarm_1(struct svc_req *rqstp, register SVCXPRT *transp)
```

```
{
        union {
                values myalarm_repl_1_arg;
        } argument;
        char *result;
        xdrproc_t _xdr_argument, _xdr_result;
        char *(*local)(char *, struct svc_req *);

        switch (rqstp->rq_proc) {
        case NULLPROC:
                (void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
                return;

        case MYALARM_REPL:
                _xdr_argument = (xdrproc_t) xdr_values;
                _xdr_result = (xdrproc_t) xdr_values;
                local = (char *(*)(char *, struct svc_req *)) myalarm_repl_1_svc;
                break;

        default:
                svcerr_noproc (transp);
                return;
        }
        memset ((char *)&argument, 0, sizeof (argument));
        if (!svc_getargs (transp, _xdr_argument, (caddr_t) &argument)) {
                svcerr_decode (transp);
                return;
        }
        result = (*local)((char *)&argument, rqstp);
        if (result != NULL && !svc_sendreply(transp, _xdr_result, result)) {
                svcerr_systemerr (transp);
        }
        if (!svc_freeargs (transp, _xdr_argument, (caddr_t) &argument)) {
                fprintf (stderr, "%s", "unable to free arguments");
                exit (1);
        }
        return;
}

int
main (int argc, char **argv)
{
        register SVCXPRT *transp;

        pmap_unset (MYALARM, MYALARM_VER);

        transp = svcudp_create(RPC_ANYSOCK);
        if (transp == NULL) {
                fprintf (stderr, "%s", "cannot create udp service.");
                exit(1);
        }
        if (!svc_register(transp, MYALARM, MYALARM_VER, myalarm_1, IPPROTO_UDP)) {
                fprintf (stderr, "%s", "unable to register (MYALARM, MYALARM_VER, udp).");
                exit(1);
        }

        transp = svctcp_create(RPC_ANYSOCK, 0, 0);
        if (transp == NULL) {
                fprintf (stderr, "%s", "cannot create tcp service.");
                exit(1);
        }
        if (!svc_register(transp, MYALARM, MYALARM_VER, myalarm_1, IPPROTO_TCP)) {
                fprintf (stderr, "%s", "unable to register (MYALARM, MYALARM_VER, tcp).");
                exit(1);
        }

        svc_run ();
        fprintf (stderr, "%s", "svc_run returned");
        exit (1);
        /* NOTREACHED */
}
```

```
-----------------------------------------------------------------------


/*************************************************************/
/* Program:
   Program Name: myalarm.x
   Author: Cathy Hoaglund
   Date Started:
   Description:
     RPC protocol definition file used by rpcgen to generate
     header and source code files and makefile.

     values is an XDR variable length array
*/
/*************************************************************/


typedef int values<>;

program MYALARM {
        version MYALARM_VER {
              values MYALARM_REPL(values v) = 1;
        } = 1;
} = 0x40000000;

-------------------------------------------------------------------------
myalarm.xdr
-------------------------------------------------------------------------

/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "myalarm.h"

bool_t
xdr_values (XDR *xdrs, values *objp)
{
      register int32_t *buf;

       if (!xdr_array (xdrs, (char **)&objp->values_val, (u_int *) &objp->values_len,
~0,
             sizeof (int), (xdrproc_t) xdr_int))
               return FALSE;
        return TRUE;
}

=============================================================
   SUPPORTING FILES:   /home/choaglun/asdf/config
=============================================================
-rw-r--r--    1 choaglun choaglun     2994 Mar 10 10:31 speedes.BTB
-rw-r--r--    1 choaglun choaglun     3051 Mar 10 10:31 speedes.BTW
-rw-r--r--    1 choaglun choaglun     3055 Mar 10 10:31 speedes.TW

-------------------------------------------------------------------------
// speedes.BTB - to be copied to speedes.par for conservative test runs
// speedes.par - all values are optional, used to override defaults
// ver 8, modified 6/28/03

parameters {
  string   mode        BREATHING_TIME_WARP  // Time management mode
  int      n_nodes     1
                                          // Number of nodes
  float    tend        400.0                // End time for simulation
  logical  statistics  T                                // Use statistics sectiom
  logical  optimize_sequential F      // Prevent sequential mode
}

gvt_parameters {
  float    Tgvt                    1.0           // Min time interval for GVT updates
```

```
  float    Tasb                    1.0     // Time betwn async broadcasts event horizon
  int      Ngvt              100           // Max Events before node seeks GVT
  int      Nrisk             0
                                 // Max Events node can process at risk
                                           // If 0, using BTB algorithm, If infinity TW
  int      Nopt         1000
                                 // Max Events node can process past GVT
}

statistics {
  logical  BTW           T        // Add times for BTW to statistics file
  logical  STAR          F        // SimTime Advance Rate since last display
  logical  PROC          T        // Tot time processing events (Tproc)
  logical  COMMIT        T        // Tot time committing events (Tcmt)
  logical  PROCEFF       F        // Ratio committed to processed events(Eff)
  logical  EVENTS        T        // Tot events since sim start(e)
  logical  EVENTSCYCLE   F        // Tot events during last cycle (e/c)
  logical  EVTGVT        F        // Tot events during GVT cycle (eg)
  logical  ROLLBACKS     T        // Tot rollback since sim start(r)
  logical  MESSAGES      T        // Tot msgs between nodes since sim start
  logical  ANTIMESSAGES  T        // Tot antimsgs btwn nodes since sim start
  logical  CANCELS       F        // Tot events cancelled since sim start(c)

  int     ReportTime         5      // Write statistics every cycle
  logical WriteGvtStatistics F              // Write GVT stats for each node

  //string  FileName           GvtStat

  string  Timer                             CPU           // Event proc timing method
  logical MemoryUsage        F              // Memory usage statistics
  logical MessageSending     T              // Message statistics (if node > 1)
  logical ObjectProcessing   F      // Object processing statistics
  logical EventProcessing    F      // Event processing statistics
  logical CriticalPath       F              // Critical path and maximum speedup

//  IntervalOutputTime{            // Interval for MemoryUsage, MessageSending
//                                          //  ObjectProcessing, EventProcessing
//     float SimTime                20.0
//     OutputFileName {
//        string StatisticsFileName intervalstats
//     }
//  }

  string Output_Method        File   // stderr, File, or Socket
  string Stat_Output_Filename statq // filename for data


 // ObjNamesByNode {                   // Print Sim Object names, handles, kind ids
 //   string fileName Objectmap
 // }
}

//------ end speedes.par --------------

--------------------------------------------------------------------------------

// speedes.BTW - to be copied to speedes.par for balanced test runs
//  where the Breathing Time Warp algorithm has normal risk
// speedes.par - all values are optional, used to override defaults
// ver 9, modified 9/7/03

parameters {
  string   mode       BREATHING_TIME_WARP  // Time management mode
  int      n_nodes    1
                                          // Number of nodes
  float    tend       400.0                 // End time for simulation
  logical  statistics T                              // Use statistics sectiom
  logical  optimize_sequential F      // Prevent sequential mode
}
```

```
gvt_parameters {
  float    Tgvt                    1.0            // Min time interval for GVT updates
  float    Tasb                    1.0     // Time betwn async broadcasts event horizon
  int      Ngvt              100           // Max Events before node seeks GVT
  int      Nrisk             500
                               // Max Events node can process at risk
                                          // If 0, using BTB algorithm, If infinity TW
  int      Nopt         1000
                               // Max Events node can process past GVT
}

statistics {
  logical  BTW           T         // Add times for BTW to statistics file
  logical  STAR          F         // SimTime Advance Rate since last display
  logical  PROC          T         // Tot time processing events (Tproc)
  logical  COMMIT        T         // Tot time committing events (Tcmt)
  logical  PROCEFF       F         // Ratio committed to processed events(Eff)
  logical  EVENTS        T         // Tot events since sim start(e)
  logical  EVENTSCYCLE   F         // Tot events during last cycle (e/c)
  logical  EVTGVT        F         // Tot events during GVT cycle (eg)
  logical  ROLLBACKS     T         // Tot rollback since sim start(r)
  logical  MESSAGES      T         // Tot msgs between nodes since sim start
  logical  ANTIMESSAGES  T         // Tot antimsgs btwn nodes since sim start
  logical  CANCELS       F         // Tot events cancelled since sim start(c)

  int      ReportTime          5      // Write statistics every cycle
  logical WriteGvtStatistics   F              // Write GVT stats for each node

  //string  FileName            GvtStat

  string  Timer                            CPU            // Event proc timing method
  logical MemoryUsage          F           // Memory usage statistics
  logical MessageSending       T           // Message statistics (if node > 1)
  logical ObjectProcessing     F       // Object processing statistics
  logical EventProcessing      F       // Event processing statistics
  logical CriticalPath         F           // Critical path and maximum speedup

//  IntervalOutputTime{             // Interval for MemoryUsage, MessageSending
//                                          //  ObjectProcessing, EventProcessing
//     float SimTime                 20.0
//     OutputFileName {
//        string StatisticsFileName intervalstats
//     }
//  }

  string Output_Method        File   // stderr, File, or Socket
  string Stat_Output_Filename statq // filename for data


 // ObjNamesByNode {                   // Print Sim Object names, handles, kind ids
 //   string fileName Objectmap
 // }
}


//------ end speedes.par --------------
-------------------------------------------------------------------------

// speedes.TW - to be copied to speedes.par for max risk test runs
//   where a max risk causes BTW to behave like Time Warp
// speedes.par - all values are optional, used to override defaults
// ver 8, modified 6/28/03

parameters {
  string   mode        BREATHING_TIME_WARP  // Time management mode
  int      n_nodes     1
                                          // Number of nodes
  float    tend        400.0                // End time for simulation
  logical  statistics  T                                  // Use statistics sectiom
  logical  optimize_sequential F      // Prevent sequential mode
}
```

```
gvt_parameters {
  float    Tgvt                      1.0            // Min time interval for GVT updates
  float    Tasb                      1.0     // Time betwn async broadcasts event horizon
  int      Ngvt                100            // Max Events before node seeks GVT
  int      Nrisk       2147483647
                                 // Max Events node can process at risk
                                              // If 0, using BTB algorithm, If infinity TW
  int      Nopt         1000
                                 // Max Events node can process past GVT
}

statistics {
  logical  BTW           T         // Add times for BTW to statistics file
  logical  STAR          F         // SimTime Advance Rate since last display
  logical  PROC          T         // Tot time processing events (Tproc)
  logical  COMMIT        T         // Tot time committing events (Tcmt)
  logical  PROCEFF       F         // Ratio committed to processed events(Eff)
  logical  EVENTS        T         // Tot events since sim start(e)
  logical  EVENTSCYCLE   F         // Tot events during last cycle (e/c)
  logical  EVTGVT        F         // Tot events during GVT cycle (eg)
  logical  ROLLBACKS     T         // Tot rollback since sim start(r)
  logical  MESSAGES      T         // Tot msgs between nodes since sim start
  logical  ANTIMESSAGES  T         // Tot antimsgs btwn nodes since sim start
  logical  CANCELS       F         // Tot events cancelled since sim start(c)

  int     ReportTime          5      // Write statistics every cycle
  logical WriteGvtStatistics  F              // Write GVT stats for each node

  //string  FileName            GvtStat

  string  Timer                              CPU            // Event proc timing method
  logical MemoryUsage         F              // Memory usage statistics
  logical MessageSending      T              // Message statistics (if node > 1)
  logical ObjectProcessing    F       // Object processing statistics
  logical EventProcessing     F       // Event processing statistics
  logical CriticalPath        F              // Critical path and maximum speedup

//  IntervalOutputTime{             // Interval for MemoryUsage, MessageSending
//                                                //  ObjectProcessing, EventProcessing
//     float SimTime                  20.0
//     OutputFileName {
//        string StatisticsFileName intervalstats
//     }
//  }

  string Output_Method        File   // stderr, File, or Socket
  string Stat_Output_Filename statq // filename for data


 // ObjNamesByNode {                    // Print Sim Object names, handles, kind ids
 //   string fileName Objectmap
 // }
}


//------ end speedes.par --------------


============================================================
    SUPPORTING FILES:    /home/choaglun/asdf/data
============================================================

-rwxrwxr-x    1 choaglun choaglun     1566 Feb 20 11:43 bundlescript

#============================================================
# bundle_script                            20 Feb 06
# Cathy Hoaglund
#     Takes all data from a run and makes it into a
# single file named grande.csv. Bundles the used files
```

```
# into a tar named todos.tar. Bundles the short summary
# files into a tar file named pocos.tar.
#     If run from the laptop offers to scp files to the
# XP machine and cleanout the upload directory. On a raven
# copies to /home0/choaglun/data/
#    Assumes a file named grande_header.txt exists, created by
#     grep msg All_somefile.csv > grande.csv
#
#============================================================
#
# Seek self-knowledge...
echo "I am " $UID
if test $UID -eq 500; then
  LOC="Home"
else
  LOC="CSUSB"
fi


# ...find our place in the universe...
echo " therefore I am at " $LOC ", about to tidy "
cd /home/choaglun/asdf/data/upload
pwd

#... assemble the information...
cp ../grande_header grande
grep -v msg All* >> grande
tar  -cf todos.tar All_*
rm All_*.csv
tar -cf summary.tar *.csv
rm *.csv
mv grande  grande.csv
ls -l

# ... and humbly step aside when the job is done
if test $UID -eq 500; then

  echo "Press enter to transfer files to DELL8600 on 192.168.0.3 or ^C to exit"
  read
  ls
  scp $PWD/grande.csv  "choaglun@192.168.0.3:/simdata"
  scp $PWD/*.tar "choaglun@192.168.0.3:/simdata"

else
  echo "Press enter to transfer data files to home0 or ^C to exit"
  read
  cp * /home0/choaglun/data
fi

# One last offer
echo "Press enter to clean out the directory or ^C to exit"
read
rm -i *
```

```
============================================================
    SUPPORTING FILES:   /home/choaglun/asdf/external
============================================================
In Major Routines Section:
-rw-r--r--    1 choaglun choaglun     8200 Mar  4 16:51 EMMain.C
-rw-rw-r--    1 choaglun choaglun     2699 Feb 12 14:45 ExternalModuleMgr.H
-rw-r--r--    1 choaglun choaglun     5395 Mar  4 17:01 ExternalModuleMgr.C
-rw-rw-r--    1 choaglun choaglun    15537 Mar  4 17:28 EMComm.C
In This Section:
-rw-r--r--    1 choaglun choaglun     2146 Feb 12 15:12 Makefile
-rw-r--r--    1 choaglun choaglun      466 Jan 29 21:19 MyReflect.H
------------------------------------------------------------------
------------------------------------------------------------------
makefile for ExtMod1
```

```
-----------------------------------------------------------------------

TARGET = ../ExtMod1

all: ${TARGET}

# Use := here instead of = so that time is not wasted by repeated
# execution of the the uname command:
ARCH := $(subst -,_,$(shell uname -s))
MACH := $(subst /,_,$(shell uname -m))

ARCHDIR = ArchitectureDirs/${ARCH}_${MACH}${EXTRA_ARCH_DIR}

UTILDIR = /home/speedes/speedes2.0.1/cathy/utils


CXX             = i386-redhat-linux7-g++

#CXX_FLAGS      = -g -DLinux -Wall
CXX_FLAGS       = -DLinux -Wall

# SPEEDES_DIR   = /hpcd/cm/speedes/delivery

SPEEDESLIBDIR   = $(SPEEDES_DIR)/lib/$(ARCHDIR)

PTHREAD_LIBDIR  = /usr/lib
PVM_ROOT        = /home/pvm/pvm3
PVM_ARCH        = LINUX
PVMLIBS         = -lpvm3 -lgpvm3

SRCS            = EMMain.C      \
                  ExternalModuleMgr.C \
                  EMComm.C

SRC_TIME = $(UTILDIR)/TimeUtil.cpp

HEADERS         = ExternalModuleMgr.H \
                  $(UTILDIR)/SemUtil.h $(UTILDIR)/TimeUtil.h

OBJS            = ${SRCS:.C=.o}
DEPENDS         = ${SRCS:.C=.d}

OBJ_TIME = TimeUtil.o


SPEEDES_INCLUDES = $(SPEEDES_DIR)/include
#COMPAT_INCLUDES = /usr/lib/gcc-lib/i386-redhat-linux7/2.96/include

#INCLUDES = -I$(SPEEDES_INCLUDES) $(COMPAT_INCLUDES)
INCLUDES = -I$(SPEEDES_INCLUDES)

ifneq ($(MAKECMDGOALS),clean)
-include ${DEPENDS}
endif

%.d:    %.C
        @echo Making dependencies for $<
        @$(CXX) -M $(INCLUDES) $< |                             \
                sed 's;\(^..*\.o\): ;\1 $@ ${@:d=f}: ;'         > $@

$(OBJ_TIME):    $(SRC_TIME)
        @echo Compiling TimeUtil.cpp
        $(CXX) $(CXX_FLAGS) -c $(SRC_TIME)

%.o:    %.C
        @echo Compiling $<
        @$(CXX) ${INCLUDES} ${CXX_FLAGS} -c   $<

${TARGET} :     ${SRCS}                                 \
                ${HEADERS}                              \
                ${OBJS}                                 \
```

```
                $(OBJ_TIME)                             \
                ${SPEEDESLIBDIR}/libSpStateMgr.so       \
                ${SPEEDESLIBDIR}/libSpEngine.so                 \
                ${SPEEDESLIBDIR}/libSpShMemTCP.so       \
                ${SPEEDESLIBDIR}/libSpUtil.so           \
                ${SPEEDESLIBDIR}/libSpHLA.so            \
                Makefile
                @echo Linking $@
                $(CXX) ${CXX_FLAGS} $(OBJS) $(OBJ_TIME) -o $@           \
                ${SPEEDESLIBDIR}/libSpStateMgr.so       \
                ${SPEEDESLIBDIR}/libSpEngine.so                 \
                ${SPEEDESLIBDIR}/libSpShMemTCP.so       \
                ${SPEEDESLIBDIR}/libSpUtil.so           \
                ${SPEEDESLIBDIR}/libSpHLA.so            \
                /usr/lib/libpthread.so                    \
                -L /home/pvm/pvm3/lib/$(PVM_ARCH)         \
                -lm -lpvm3 -lgpvm3


clean:
        @rm -rf *.o ${TARGET} *.d
-------------------------------------------------------------------------
MyReflect.H
-------------------------------------------------------------------------

// MyReflect.H
#ifndef MyReflect_H
#define MyReflect_H

#include "SpStateMgrEvent.H"

class MyReflect: public SpStateMgrEvent {
  public:
    MyReflect() {}
    virtual ~MyReflect() {}
    virtual void Process() {
      cout << "MyReflect Process at " << GetTimeTag()
           << ", Global Id: " << GetSimObjGlobalId()
           << ", Event Name: " << GetEventName() << endl;
    }
};

#endif

-------------------------------------------------------------------------

=============================================================
    SUPPORTING FILES:    /home/choaglun/asdf/harvester
=============================================================
In Major Routines Section:
-rw-rw-r--    1 choaglun choaglun    25313 Feb 26 12:50 Harvester.cpp
-rw-r--r--    1 choaglun choaglun     2807 Jan 29 21:19 Harvester.h
-rw-r--r--    1 choaglun choaglun      753 Jan 29 21:19 HarvestMain.cpp
In This Section:
-rw-r--r--    1 choaglun choaglun      635 Jan 29 21:19 makefile

-------------------------------------------------------------------------

#-----------------------------------------
# makefile for Harvester     11/21/04
#-----------------------------------------
# Parameters
HARVESTER = ../Harvester

#-----------------------------------------
# This section is for building applications.
#-----------------------------------------

all :   $(HARVESTER)

$(HARVESTER) : HarvestMain.o Harvester.o
```

```
	g++ HarvestMain.cpp Harvester.cpp -o ../Harvester -Wno-deprecated

HarvestMain.o: HarvestMain.cpp Harvester.h
	g++ -c HarvestMain.cpp -Wno-deprecated

Harvester.o: Harvester.cpp Harvester.h
	g++ -c Harvester.cpp -Wno-deprecated


clean:
	rm -f core *.o *.a *.trace

-----------------------------------------------------------------------



=================================================================
    SUPPORTING FILES:   /home/choaglun/asdf/mysim
=================================================================
In Major Routines Section
-rw-r--r--    1 choaglun choaglun      842 Feb 17 09:55 SimMain.C
-rw-r--r--    1 choaglun choaglun     6531 Feb 17 10:15 S_Control.C
-rw-r--r--    1 choaglun choaglun     2090 Jan 29 21:20 S_Control.H
-rw-r--r--    1 choaglun choaglun    11470 Jan 29 21:20 S_Ship.C
-rw-r--r--    1 choaglun choaglun     3737 Jan 29 21:20 S_Ship.H


In This Section:

-rw-r--r--    1 choaglun choaglun     1457 Jan 29 21:20 Makefile
-----------------------------------------------------------------------
Makefile for mysim directory
-----------------------------------------------------------------------


TARGET = ../Ships

all: ${TARGET}

# Use := here instead of = so that time is not wasted by repeated
# execution of the the uname command:
ARCH := $(subst -,_,$(shell uname -s))
MACH := $(subst /,_,$(shell uname -m))

ARCHDIR = ArchitectureDirs/${ARCH}_${MACH}${EXTRA_ARCH_DIR}

CXX             = i386-redhat-linux7-g++

#CXX_FLAGS      = -g -DLinux -Wall
CXX_FLAGS       =  -DLinux -Wall

SPEEDESLIBDIR   = $(SPEEDES_DIR)/lib/$(ARCHDIR)

SRCS            =                                   \
                  SimMain.C                         \
                  S_Ship.C                          \
                  S_Control.C                       \
                   TimeUtil.C     \

HEADERS         =

OBJS            = ${SRCS:.C=.o}
DEPENDS         = ${SRCS:.C=.d}

SPEEDES_INCLUDES = $(SPEEDES_DIR)/include
#COMPAT_INCLUDES = /usr/lib/gcc-lib/i386-redhat-linux7/2.96/include

#INCLUDES = -I$(SPEEDES_INCLUDES) $(COMPAT_INCLUDES)
INCLUDES = -I$(SPEEDES_INCLUDES)
```

```
ifneq ($(MAKECMDGOALS),clean)
-include ${DEPENDS}
endif

%.d:    %.C
        @echo Making dependencies for $<
        @$(CXX) -M $(INCLUDES) $< |                              \
                sed 's;\(^..*\.o\): ;\1 $@ ${@:d=f}: ;'          > $@

%.o:    %.C
        @echo Compiling $<
        @$(CXX) ${INCLUDES} ${CXX_FLAGS} -c $<

${TARGET} :     ${SRCS}                                  \
                ${HEADERS}                               \
                ${OBJS}                                  \
                ${SPEEDESLIBDIR}/libSpEngine.so                  \
                ${SPEEDESLIBDIR}/libSpShMemTCP.so        \
                ${SPEEDESLIBDIR}/libSpUtil.so            \
                Makefile
                @echo Linking $@
                $(CXX) ${CXX_FLAGS} $(OBJS) -o $@        \
                ${SPEEDESLIBDIR}/libSpEngine.so                  \
                ${SPEEDESLIBDIR}/libSpShMemTCP.so        \
                ${SPEEDESLIBDIR}/libSpUtil.so            \
                -lm

clean:
        @rm -rf *.o ${TARGET} *.d
```

```
================================================================
    SUPPORTING FILES:   /home/choaglun/asdf/utils
================================================================
-rwxrwxr-x    1 choaglun choaglun      810 Feb 17 09:38 datadirscript
-rwxrwxr-x    1 choaglun choaglun      360 Mar 24 16:20 goASscript
-rwxr-xr-x    1 choaglun choaglun      337 Mar 24 16:21 goEMscript
-rwxrwxr-x    1 choaglun choaglun      403 Mar 24 16:21 goVMscript
-rwxr-xr-x    1 choaglun choaglun     4267 Mar 24 16:11 runscript
-rwxr-xr-x    1 choaglun choaglun     3925 Mar 24 16:25 runscript1
-rw-r--r--    1 choaglun choaglun     4362 Jan 29 21:21 SemUtil.h
-rwxrwxr-x    1 choaglun choaglun      877 Feb 25 08:49 staticscript
-rw-r--r--    1 choaglun choaglun     2928 Jan 29 21:21 TimeUtil.cpp
-rw-r--r--    1 choaglun choaglun     1164 Jan 29 21:21 TimeUtil.h

--------------------------------------------------------------------------------
#!/bin/bash
#====================================================================
# Name: datadirscript                          for Sim Version 12
#   manages creation of data file directories
#     tdir = Test Series location
#     sdir = subdirectory for 1 run's worth of data files
#   uses bash test to see if necessary to create a data directory
# Author: Cathy Hoaglund                last modified 17 Feb 06
#====================================================================
#

cd /home/choaglun/asdf/data

tdir=$1
tnum=$2

sdir=$tdir"/"$tnum

if [ -e $tdir ]; then
```

```
   echo "using test directory" $tdir;
else
   echo "making test directory " $tdir;
   mkdir $tdir;
fi

if [ -e $sdir ]; then
   echo "using save directory" $sdir;
else
   echo "making save directory " $sdir;
   mkdir $sdir;
fi
```

```
# ********************************************************************
# Name: goASscript                                  for Sim version 12
#  Start AlarmSensor in xterm
#
# Author: Cathy Hoaglund                  Last modified: 25 Feb 06
#*********************************************************************

AlarmSensor $1 $2 $3  $4
#echo "Enter "
#read
```

```
# ********************************************************************
# Name: goEMscript                                  for Sim version 11
#  Start ExtMod1 in xterm
#
# Author: Cathy Hoaglund                  Last modified: 21 Jan 05
#*********************************************************************

ExtMod1
#echo "enter"
#read
```

```
#!/bin/bash
# ********************************************************************
# Name: goVMscript                                  for Sim version 12
#  execute VM_starter
# Author: Cathy Hoaglund                  Last modified: 16 Feb 06
#*********************************************************************


cd /home/choaglun/asdf
/home/choaglun/asdf/VM_starter

#echo "enter to exit"
#read
```

```
#!/bin/bash
#====================================================================
# Name: runscript   FOR DISTRIBUTED RUNS          for Sim Version 12
#   Assumes raven5 is sending if run from raven4
#   The middle layer - does one run with a given configuration
#   - Writes a config file with run-by-run parameters
#   - Opens xterms for executables that run the sim
#   - Preps the data from the many little files
#   - Calls Harvester
#   - Tidies up files
#
# Command line:
#    runscript cmode nummsgs nrisk simrate physconf
#    testdir testnum nnodes msgHz chkHz
# Author: Cathy Hoaglund               Last modified 24 Mar 06
```

```
#=====================================================================

cmode=$1
nummsgs=$2
nrisk=$3
simrate=$4
physconf=$5
testdir=$6
testnum=$7
nnodes=$8
msgHz=$9
chkHz=${10}

cd /home/choaglun/asdf/
#---------------------------------------------------------------
savedir="data""/"$testdir"/"$testnum"/"

echo  "RUNSCRIPT - cmode " $cmode "nummsgs "  $nummsgs " nrisk " $nrisk
echo  " simrate " $simrate "physconf " $physconf " nnodes " $nodes

utils/datadirscript $testdir $testnum

echo  $cmode    > config/config.par
echo  $nummsgs  >> config/config.par
echo  $nrisk    >> config/config.par
echo  $simrate  >> config/config.par
echo  $physconf >> config/config.par
echo  $nnodes   >> config/config.par
echo  $msgHz    >> config/config.par
echo  $chkHz    >> config/config.par
#echo "-------------------------------------------------"
#echo "config.par: cmode msgnum nrisk simrate"
#echo "            physconf nnodes msgHz chkHz"
#cat config/config.par
#echo "-------------------------------------------------"

host=$HOSTNAME

if [ $physconf = "10" ] || [ $physconf = "30" ]; then
   echo "Physical Config: " ${physconf} " = wrong script for " $host
   exit
elif [ $physconf = "40" ] || [ $physconf = "50" ]; then
   echo "Physical Config: " ${physconf} " = wrong script for " $host
   exit
elif [ $physconf = "43" ]; then
   echo "Physical Config: 43 = raven 4 sending to " $host
elif [ $physconf = "45" ]; then
   echo "Physical Config: 45 = raven 4 sending to " $host
elif [ $physconf = "54" ]; then
   echo "Physical Config: 54 = raven 5 sending to " $host
else
   echo "Physical Config not supported - quitting"
   exit
fi

echo "files initialized"

#---------------------------------------------------------------
#--RUN:

#echo "enter to run or ^c to exit"
#read  $ans

# Tha actual commands to start executables are in yet another script so the
# xterm subshell won't disappear without user input.
# can't run Ships in xterm or sim.out lacks needed data
#notes:. xterm columns x row + pixels from left + pixels from top
#OR . xterm columns x row - pixels from right - pixels from bottom


if [ $cmode = v ]; then
```

```
  cd /home/choaglun/asdf
  (xterm -geometry 80x40+0+0 -e utils/goVMscript ) &
   /home/choaglun/asdf/Ships $nnodes   &


elif [ $cmode = m ]; then
  cd /home/choaglun/asdf
  (xterm -geometry 80x20+0+0 -e utils/goEMscript &)
  rsh raven4 "~/asdf/AlarmSensor $host $cmode $nummsgs $msgHz " &
  /home/choaglun/asdf/Ships $nnodes    &

elif [ $cmode = s ]; then
  echo "wrong script"

else
   echo "unrecognized comm mode, exiting runscript "
   exit 1
fi

# Make sure all background processes have finished
wait
echo "Sim done"


# collect remote file
rsh raven5 cp /home/choaglun/asdf/AS.out /home0/choaglun/asdf
cp /home0/choaglun/asdf/AS.out /home/choaglun/asdf

#echo "enter to continue with post-test or ^c to exit"
#read  $ans

#--POSTTEST: Prep data files
date +%H%M%S > data/lastruntime

date +%D >> data/lastruntime
echo $testnum >> data/testnumfile

grep "GVT=400" sim.out >> data/prep.dat

grep "GVT" sim.out | wc > data/GVT_count.dat

echo "Data files prepped"

Harvester

echo "Data files merged"

mv logs/* $savedir
mv data/*.dat $savedir
mv *.out OK* $savedir
if [ $cmode = s ]; then
  rm semid_file
fi
mv speedes.par $savedir
mv config/config.par $savedir
cp data/*_Data.csv  data/upload/$testnum".csv"
cp data/All*  data/upload/"All_"$testnum".csv"

mv data/*.csv data/$testdir/$testnum
rm data/lastruntime
rm data/testnumfile

echo "Data files copied and moved. Done with $testnum"
echo "-----------------------------------------------"


-------------------------------------------------------------------------------

#!/bin/bash
#=============================================================================
# Name: runscript1   NON-DISTRIBUTED          for Sim Version 12
```

```
#   The middle layer - does one run with a given configuration
#   - Writes a config file with run-by-run parameters
#   - Opens xterms for executables that run the sim
#   - Preps the data from the many little files
#   - Calls Harvester
#   - Tidies up files
#
# Command line:
#    runscript1; cmode nummsgs nrisk simrate physconf
#    testdir testnum nnodes msgHz chkHz
# Author: Cathy Hoaglund                Last modified 24 Mar 06
#=============================================================================

cmode=$1
nummsgs=$2
nrisk=$3
simrate=$4
physconf=$5
testdir=$6
testnum=$7
nnodes=$8
msgHz=$9
chkHz=${10}

cd /home/choaglun/asdf/
#-----------------------------------------------------------------
savedir="data""/"$testdir"/"$testnum"/"

echo  "RUNSCRIPT1 - cmode " $cmode "nummsgs "  $nummsgs " nrisk " $nrisk
echo  " simrate " $simrate "physconf " $physconf " nnodes " $nodes

utils/datadirscript $testdir $testnum

echo  $cmode    > config/config.par
echo  $nummsgs  >> config/config.par
echo  $nrisk    >> config/config.par
echo  $simrate  >> config/config.par
echo  $physconf >> config/config.par
echo  $nnodes   >> config/config.par
echo  $msgHz    >> config/config.par
echo  $chkHz    >> config/config.par
#echo "-------------------------------------------------"
#echo "config.par: cmode msgnum nrisk simrate"
#echo "            physconf nnodes msgHz chkHz"
#cat config/config.par
#echo "-------------------------------------------------"

host=$HOSTNAME

if [ $physconf = "10" ]; then
   echo "Physical Config: " ${physconf} " = All on Dell Hostname=" $host
elif [ $physconf = "30" ]; then
   echo "Physical Config: " ${physconf} " = All on raven 3 Hostname" $host
elif [ $physconf = "40" ]; then
   echo "Physical Config: " ${physconf} " = All on raven 4 Hostname " $host
elif [ $physconf = "50" ]; then
   echo "Physical Config: " ${physconf} " = All on raven 5 Hostname " $host
else
   echo "Physical Config unsupported - quitting"
   exit
fi

echo "files initialized"

#-----------------------------------------------------------------
#--RUN:

#echo "enter to run or ^c to exit"
#read  $ans

# Tha actual commands to start executables are in yet another script so the
```

```
# xterm subshell won't disappear without user input.
# can't run Ships in xterm or sim.out lacks needed data
#notes:. xterm columns x row + pixels from left + pixels from top
#OR . xterm columns x row - pixels from right - pixels from bottom


if [ $cmode = v ]; then
  cd /home/choaglun/asdf
  (xterm -geometry 80x40+0+0 -e utils/goVMscript ) &
   /home/choaglun/asdf/Ships $nnodes   &


elif [ $cmode = m ] || [ $cmode = s ]; then
  cd /home/choaglun/asdf
  (xterm -geometry 80x20+0+0 -e utils/goEMscript &)
  (xterm -geometry 80x20-0+0 -e utils/goASscript $host $cmode $nummsgs $msgHz &)

  /home/choaglun/asdf/Ships $nnodes   &

else
   echo "unrecognized comm mode, exiting runscript1 "
   exit 1
fi

# Make sure all background processes have finished
wait
echo "Sim done"

#echo "enter to continue with post-test or ^c to exit"
#read  $ans

#--POSTTEST: Prep data files
date +%H%M%S > data/lastruntime

date +%D >> data/lastruntime
echo $testnum >> data/testnumfile

grep "GVT=400" sim.out >> data/prep.dat

grep "GVT" sim.out | wc > data/GVT_count.dat

echo "Data files prepped"

Harvester

echo "Data files merged"

mv logs/* $savedir
mv data/*.dat $savedir
mv *.out OK* $savedir
rm semid_file

mv speedes.par $savedir
mv config/config.par $savedir
cp data/*_Data.csv  data/upload/$testnum".csv"
cp data/All*  data/upload/"All_"$testnum".csv"

mv data/*.csv data/$testdir/$testnum
rm data/lastruntime
rm data/testnumfile

echo "Data files copied and moved. Done with $testnum"
echo "-----------------------------------------------"

-------------------------------------------------------------------------

// ********************************************************************
//  SemUtil.H
//    Semaphore utility functions
//  Cathy Hoaglund
//                                          Last modified 5 Feb 05
```

```
// ************************************************************************

#ifndef SEMUTIL_H
#define SEMUTIL_H

#include <stdio.h>
#include <iostream>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <signal.h>

//========================================
// Global variables & function prototypes
//========================================

int start_sem(int numsem);
int test_sem(int id, int indx);
int release_sem(int id, int indx);
int request_sem(int id, int indx);
int delete_sem(int id);


#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
/* union semun is defined by including <sys/sem.h> */
#else
/* according to X/OPEN we have to define it ourselves */
union semun {
  int val;                    /* value for SETVAL */
  struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
  unsigned short int *array;  /* array for GETALL, SETALL */
  struct seminfo *__buf;      /* buffer for IPC_INFO */
};
#endif

//=============================================================================
// start_sem
//   Gets a unique semid
//   Returns semid, which may be 0 or greater, or -1 on error
//=============================================================================

int start_sem(int numsem) {

  union semun sunion;
  int id = 1;

  /* Request a new semaphore set with numsem read-write semaphores*/
  id = semget(IPC_PRIVATE, numsem, SHM_R | SHM_W);


  if (id != -1) {
     /* Initialize resource count to number of semaphores requested */
     for (int i = 0; i < numsem; i++){
        sunion.val = (unsigned)1;

        if (semctl(id, i, SETVAL, sunion) == -1) {
            printf ("- semctl SETVAL failed: %s\n", strerror(errno));
            return(-1);
        }

       //printf( "set %d = %d\n", i, semctl(id, i, GETVAL, sunion));
     }
  }
  else  {
     printf(" - Semaphore request failed: %s.\n", strerror(errno));
  }
```

```
  return (id);
}

//============================================================================
// test_sem
//  returns 0 or 1 on success, -1 on error
//============================================================================

int test_sem(int id, int indx) {
  struct semid_ds mbuf;
  int semv = -1;
  union semun arg;
  arg.buf = &mbuf;

  if((semctl(id, 0, IPC_STAT, arg) < 0)){
    printf(" IPC_STAT error: %s\n", strerror(errno));
    return(-1);
  }

  if(indx >= (int)mbuf.sem_nsems){
    printf(" index error: %s\n", strerror(errno));
    return(-1);
  }

  semv = (int) (semctl(id, indx, GETVAL, arg));


  return(semv);
}

//============================================================================
// release_sem -
//  returns 0 on success, -1 on error
//============================================================================

int release_sem(int id, int indx) {
  struct sembuf sb;


  sb.sem_num = indx;          //semaphore index
  sb.sem_op = 1;              //sem oper - increase free count
  sb.sem_flg = SEM_UNDO;      //operation flags

  if (semop(id, &sb, 1) == -1) {
    //printf("- semop release error: %s\n", strerror(errno));
    //exit(255);
    return(-1);
  }

  return(0);
}

//============================================================================
// request_sem
//  returns 0 on success, -1 on error
//============================================================================

int request_sem(int id, int indx) {
  struct sembuf sb;

  sb.sem_num = indx;
  sb.sem_op = -1;
  sb.sem_flg = SEM_UNDO;

  //printf("- Requesting resource for %d...", id);
  fflush(stdout);

  if (semop(id, &sb, 1) == -1) {
   // printf(" - semop request error: %s\n", strerror(errno));
  //  exit(255);
```

```
    return(-1);
  }

  //printf(" request_sem done - got it.\n");
  return(0);
}


//============================================================================
// delete_sem
//  returns 0 on success, -1 on error
//============================================================================

int delete_sem(int id) {
  //printf("- Deleting semaphore.\n");
  if (semctl(id, 0, IPC_RMID, 0) == -1) {
    //printf("- Error releasing semaphore.\n");
    return(-1);
  }
  return(0);
}



#endif

-------------------------------------------------------------------------

#!/bin/bash
#=========================================================================
# Name: staticscript                              for Sim Version 12
#   Bash script to set up .par files for a series of runs
#   Sets up only the files read by simulation on local server
#                                      Last modified 27 Feb 06
#
# Command line: staticscript nrisk
# Author: Cathy Hoaglund
#=========================================================================

nrisk=$1

cd /home/choaglun/asdf
#-------------------------------------------------------------

case $nrisk in
  L)
    cp config/speedes.BTB    speedes.par
#    echo "BTB"
  ;;
  M)
    cp config/speedes.BTW    speedes.par
#    echo "BTW"
  ;;
  H)
    cp config/speedes.TW     speedes.par
#    echo "TW"
  ;;
  *)
     echo "staticscript - huh?"
  ;;
  esac

echo $simrate > config/ping.par

-------------------------------------------------------------------------


//*************************************************************
// TimeUtil.cpp                              for Sim Version 11
//  Common utility functions for dealing with Unix time
// Author: Cathy Hoaglund        Last modified: 30 Jan 05
```

```
//
//
//*****************************************************************

#include <time.h>
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>
#include "TimeUtil.h"


void TellTime(long* hh, long* mm, long* ss, long* uu) {
// Get sec & microsec, do conversions,

   time_t timv = time(0);          // Clear prior vals
   struct tm    *loc_tm;           // Ptr to tm struct for localtime
   struct timeval tv;              // time value in seconds & usecs
   struct timezone tz;
   gettimeofday(&tv, &tz);
   timv = tv.tv_sec;

   loc_tm = localtime(&timv);
   *hh = loc_tm->tm_hour;
   *mm = loc_tm->tm_min;
   *ss = loc_tm->tm_sec;
   *uu = tv.tv_usec;
}


void ConvertTime(long inTime, long* hh, long* mm, long* ss) {
// Get time in unix seconds & turn into hh,mm,ss

   time_t timv = time(0);          // Clear prior vals
   struct tm    *loc_tm;           // Ptr to tm struct for localtime        ;
   timv = inTime;

   loc_tm = localtime(&timv);
   *hh = loc_tm->tm_hour;
   *mm = loc_tm->tm_min;
   *ss = loc_tm->tm_sec;
}

void RawTime(long* secs, long* usecs){
// Get sec & microsec,

   struct timeval tv;              // time value in seconds & usecs
   struct timezone tz;
   gettimeofday(&tv, &tz);
   *secs = tv.tv_sec;
   *usecs = tv.tv_usec;
}

void DoTimetag(long* hh, long* mm, long* ss, long* uu, char* timetag){
// Get time & fill char array of length 16
   TellTime(hh, mm, ss, uu);
   sprintf(timetag, "%02ld:%02ld:%02ld.%06ld", *hh, *mm, *ss, *uu);
}

char* TmTag( char* timetag){
// Get time & fill char array of length 16
   long hh, mm, ss, uu;
   TellTime(&hh, &mm, &ss, &uu);
   sprintf(timetag, "%02ld:%02ld:%02ld.%06ld", hh, mm, ss, uu);
   return timetag;
}

long Convert2Sec(long hh, long mm, long ss) {
// Use input timevals to compute time as seconds since midnight

//   cout << "Convert2Sec: " << hh << ":" << mm << ":" << ss << endl;
   long mins = (hh * 60) + mm;
```

```
   long secs = (mins * 60) + ss;
//   cout << "Convert2Sec: secs " << secs << endl;

   return secs;
}

double CalcDelay (long tsecA,long tsecB,long uuA,long uuB) {
// Use input timevals to compute time difference in seconds

   // cout << "CalcDelay: " << tsecA << "." << uuA << " - " << tsecB
   //           << "." << uuB << endl;
   double secsA = tsecA + (uuA / 1000000.0);
   double secsB = tsecB + (uuB / 1000000.0);

   //cout << "CalcDelay: Delay in secs.usecs " << secsA - secsB << endl;

   return (secsA - secsB );

}


void FillTimetag(long hh, long mm, long ss, long uu, char* timetag) {
// Use input timevals to fill char array of length 16

   sprintf(timetag, "%02ld:%02ld:%02ld.%06ld", hh, mm, ss, uu);
}
```

```
// ********************************************************************
// TimeUtil.h                              for Sim Version 11
//  Common utility functions for dealing with Unix time
// Author: Cathy Hoaglund        Last modified: 28 Jan 05
//
//
// ********************************************************************

#ifndef TimeUtil_H
#define TimeUtil_H


void TellTime(long* hh, long* mm, long* ss, long* uu) ;
// Get sec & microsec, do conversions

void ConvertTime(long inTime, long* hh, long* mm, long* ss);
// Get time in unix seconds & turn into hh,mm,ss

void RawTime(long* secs, long* usecs) ;
// Get sec & microsec

void DoTimetag(long* hh, long* mm, long* ss, long* uu, char* timetag);
// Get time & fill char array of length 16

char* TmTag(char* timetag);
// Return a timetag

long Convert2Sec(long hh, long mm, long ss);
// Use input timevals to compute time as seconds since midnight


double CalcDelay (long tsecA,long tsecB,long uuA,long uuB);
// Use input timevals to compute time difference in seconds


void FillTimetag(long hh, long mm, long ss, long uu, char* timetag);
// Use input timevals to fill char array of length 16


#endif
```

```
------------------------------------------------------------------------------


============================================================
  SUPPORTING FILES:   /home/choaglun/asdf/virtualmach
============================================================
In Major Routines Section:
-rw-rw-r--    1 choaglun choaglun     6729 Mar  5 15:42 driver.C
-rw-rw-r--    1 choaglun choaglun     9396 Mar  6 10:52 alarm_top.C
-rw-rw-r--    1 choaglun choaglun     6075 Mar  6 10:53 myalarm_mgrVM.C
-rw-rw-r--    1 choaglun choaglun     8342 Mar  4 16:51 extmgr.C
-rw-rw-r--    1 choaglun choaglun     1716 Mar  5 10:53 myalarm_mgrVM.H

In This Section:
-rw-r--r--    1 choaglun choaglun     2660 Feb 19 16:22 Makefile
------------------------------------------------------------------------


# Makefile for virtual_mach directory              updated 19 Feb 06

TARGET = ../VM_starter

all: ${TARGET}

# Use := here instead of = so that time is not wasted by repeated
# execution of the the uname command:
ARCH := $(subst -,_,$(shell uname -s))
MACH := $(subst /,_,$(shell uname -m))

ARCHDIR = ArchitectureDirs/${ARCH}_${MACH}${EXTRA_ARCH_DIR}

EXTDIR    = /home/choaglun/asdf/external
UTILDIR   = /home/choaglun/asdf/utils

#CXX           = g++
CXX           = i386-redhat-linux7-g++

CXX_FLAGS     = -DLinux -Wall -Wno-deprecated

SPEEDESLIBDIR  = $(SPEEDES_DIR)/lib/$(ARCHDIR)

PTHREAD_LIBDIR  = /usr/lib
PVM_ROOT        = /home/pvm/pvm3
PVM_ARCH        = LINUX
PVMLIBS         = -lpvm3 -lgpvm3


SRCS          = driver.C      \
                  extmgr.C      \
                alarm_top.C   \
                myalarm_mgrVM.C

SRC_TIME = $(UTILDIR)/TimeUtil.cpp

SRC_EMM =  $(EXTDIR)/ExternalModuleMgr.C
SRC_EMC =  $(EXTDIR)/EMComm.C

HEADERS       = myalarm_mgrVM.H       \
                $(EXTDIR)/ExternalModuleMgr.H \
                $(UTILDIR)/TimeUtil.h

OBJS          = ${SRCS:.C=.o}
DEPENDS       = ${SRCS:.C=.d}

OBJ_TIME = TimeUtil.o
OBJ_EMM  = ExternalModuleMgr.o
OBJ_EMC  = EMComm.o

SPEEDES_INCLUDES = $(SPEEDES_DIR)/include

INCLUDES = -I$(SPEEDES_INCLUDES) -I$(PVM_ROOT)/include
```

```
ifneq ($(MAKECMDGOALS),clean)
-include ${DEPENDS}
endif

%.d:    %.C
        @echo --- Making dependencies for $<
        @$(CXX) -M $(INCLUDES) $< |                          \
                sed 's;\(^..*\.o\): ;\1 $@ ${@:d=f}: ;'          > $@

$(OBJ_TIME):   $(SRC_TIME)
        @echo --- Compiling TimeUtil.cpp ---
        $(CXX) $(CXX_FLAGS) -c $(SRC_TIME)

$(OBJ_EMM):    $(SRC_EMM)
        @echo --- Compiling External modules locally ---
        $(CXX) $(CXX_FLAGS) -c $(SRC_EMM)

$(OBJ_EMC):    $(SRC_EMC)
        @echo --- Compiling External modules locally ---
        $(CXX) $(CXX_FLAGS) -c $(SRC_EMC)


%.o:    %.C
        @echo --- Compiling $< ---
        @$(CXX) ${INCLUDES} ${CXX_FLAGS} -c   $<

${TARGET} :    ${SRCS}                                  \
               ${HEADERS}                               \
               ${OBJS}                                  \
               $(OBJ_TIME)                              \
                $(OBJ_EMM) $(OBJ_EMC)                     \
               ${SPEEDESLIBDIR}/libSpStateMgr.so        \
               ${SPEEDESLIBDIR}/libSpEngine.so               \
               ${SPEEDESLIBDIR}/libSpShMemTCP.so        \
               ${SPEEDESLIBDIR}/libSpUtil.so            \
               ${SPEEDESLIBDIR}/libSpHLA.so             \
               Makefile
               @echo --- Linking $@ ---
               $(CXX) ${CXX_FLAGS} $(OBJS) $(OBJ_TIME)\
                     $(OBJ_EMM) $(OBJ_EMC)    -o $@  \
               ${SPEEDESLIBDIR}/libSpStateMgr.so        \
               ${SPEEDESLIBDIR}/libSpEngine.so               \
               ${SPEEDESLIBDIR}/libSpShMemTCP.so        \
               ${SPEEDESLIBDIR}/libSpUtil.so            \
               ${SPEEDESLIBDIR}/libSpHLA.so             \
               /usr/lib/libpthread.so                     \
               -L /home/pvm/pvm3/lib/$(PVM_ARCH)          \
               -lm -lpvm3 -lgpvm3


clean:
        @rm -rf *.o ${TARGET} *.d

-------------------------------------------------------------------------
```

REFERENCES

1. R.L. Bagrodia,"Perils and Pitfalls of Parallel Discrete-Event Simulation", Proceedings of the 1996 Winter Simulation Conference, ACM Press, New York, NY, 1996, pp. 136-143.

2. C.J.M. Booth, et al., "Dynamic memory usage in parallel simulation: a case study of a large-scale military logistics application", Proceedings of the Winter Simulation Conference November 1996, ACM Press, New York, NY, 1996, pp. 975-982.

3. C.D. Carothers, et al., "Visualizing Parallel Simulations in Network Computing Environments: A Case Study", Proceedings of the 1997 Winter Simulation Conference, ACM Press,New York, NY, USA, pp. 110-117.

4. S. Chandra, J.R. Larus, and A. Rogers, "Where is Time Spent in Message-Passing and Shared-Memory Programs", *Proceedings Sixth International Conference on Architectural Support for Programming Languages and Operating Systems,* ACM Press, New York, NY, pp. 61 - 73, 1994.

5. T.J. Croak, "Application of Capacity Planning Techniques As Architectural Design Decision Aids For 3-Tier And N-Tier Software Architectures", Dissertation for DCS, Colorado Technical University, Colorado Springs, Colorado, May 2000.

6. B.P. Douglass, *Real-time UML Second Edition,* Addison Wesley Longman, Reading, MA, 2000.

7. S.L. Ferenci, K.S. Perumalla, R.M. Fujimoto, "An Approach for Federating Parallel Simulators", Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation, IEEE Computer Society, Washington, DC, USA, pp. 63-70, 2000.

8. R.M. Fujimoto and M. Hybinette, "Computing global virtual time in shared-memory multiprocessors ", *Transactions on Modeling and Computer Simulation (TOMACS)*,Volume 7, Issue 4, pp. 425-446, ACM Press, New York, NY, USA, October 1997.

9. R.M. Fujimoto, *Parallel and Distributed Simulation Systems*, John Wiley & Sons, Inc., New York, 2000.

10. A. Geist, et al., *PVM: Parallel Virtual Machine*, MIT Press, Cambridge, MA., 1997

11. R. Hillson, "Design and Implementation of an MPI (Message Passing Interface) Interface for the SPEEDES Simulation Framework", High Performance Computing Modernization Program 1999 User Group Conference Proceedings (UGC99), DoD High Performance Computing Modernization Program, 1999, http:// www.hpcmo.hpc.mil/Htdocs/UGC/UGC99/papers/fms2-2/

12. D.R. Jefferson, "Virtual Time", ACM Transactions on Programming Languages and Systems, Vol. 7, No. 3, July 1985, Pages 404-425.

13. D. Kranz, et al., " Integrating message-passing and shared-memory", ACM SIGPLAN Notices, Proceedings of the Fourth ACM SIGPLAN symposium on Principles & Practice of Parallel Programming, Vol. 28, Issue 7, July 1993. pp. 54-63

14. M. Jovanovic, and V. Milutinovic, "An Overview of Reflective Memory Systems", IEEE Concurrency, Vol. 7, No. 2; April-June 1999, pp. 56-64.

15. A.M. Law, and W. D. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, New York, 2000.

16. J.W.S. Liu, *Real-Time Systems*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

17. R.M. McGraw, et al., "Integration Of SPEEDES Into The JMASS Architecture", Summer Computer Simulation Conference 2000 , Society for Computer Simulation, 2000.

18. T. McGuinness et al., "Executing Independent Parallel Applications Using the SPEEDES Communications Library.", *Proceedings of the High Performance Computing Modernization Program Users Group Conference, 2001,* http://www.hpcmo.hpc.mil/ Htdocs/UGC/UGC01/thursday.html.

19. Metron, Inc., *SPEEDES User's Guide, The Synchronous Parallel Environment for Emulation and Discrete-Event Simulation, Revision Number: 3*, Solana Beach CA, 4 October 2001, http://www.speedes.com

20. D.M. Nichol, and X. Liu,"The Dark Side of Risk (What your mother never told you about Time Warp)", Proceedings of the 1997 Workshop on Parallel and Distributed Simulation, IEEE Computer Society Press Los Alamitos, CA, 1997, pp. 188 - 195.

21. C.D. Pham, R.L. Bagrodia. "Building parallel time-constrained HLA federates: a case study with the Parsec parallel simulation language", Proceedings of 1998 conference on Winter simulation , 1998 , Washington, D.C , pp. 1555 - 1562

22. M. Pullen, M. Myjak, and C.Bouwens, *Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment,* IETF RFC 2502, February 1999; http://www.rfc-editor.org/rfc/rfc2502.txt.

24. J.S. Steinman, "Breathing Time Warp", Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS93), ACM Press, New York, NY, USA, 1994, pp. 109-118.

25. J.S. Steinman et al., "Global Virtual Time and distributed synchronization", Proceedings of the 9th Workshop on Parallel and Distributed Simulation July 1995, Volume 25, Issue 1, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 139-148

26. J.S. Steinman, "Scalable Parallel and Distributed Military Simulations using the Speedes Framework", Electronic Conference on Scalability in Training Simulation, 1995. http://www.scs.org/oldConferences/elecsim/elecsim95/papers/steinman/steinman-main.html

27. A.S. Tanenbaum, and M. Van Steen, Maarten, *Distributed Systems, Principles and Paradigms*, Prentice Hall, Upper Saddle River, N.J, 2001

28. J. Weber, 2002, Telephone interview with author, University of Dayton, Department of Electrical and Computer Engineering, 16 Mar,

29. F. Weiland, E. Blair, and T. Zukas, "Parallel Discrete-Event Simulation (PDES): A Case Study in Design, Development, and Performance Using SPEEDES", Proceedings of the Ninth Workshop on Parallel and Distributed Simulation (PADS'95), pp. 103 - 110, IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

30. G.A. Whitted, R.J. Meidenbauer, S.E. Louton, "Simulation Control for an Integrated Multispectral Environment", HPC Workshop 1999. http://www.dtc.army.mil/hpcw/1999/whitted/

31. D.L. Wright. "The Utility of Advanced Distributed Simulation for Electronic Warfare Systems Testing (JADS JT&E-TR-99-017)", Joint Advanced Distributed Simulation Joint Test Force, Kirtland Air Force Base, N.M., 19 November 1999, http://akss.dau.mil/docs/041EVDOC.doc

32. B.P. Zeigler, H. Praehofer, T.G. Kim, *Theory of Modeling and Simulation 2nd Edition*, Academic Press, San Diego, CA., 2000.

33. J. L. Zhang and C. Tropper, "The Dependence List in Time Warp", Proceedings of the fifteenth workshop on Parallel and Distributed Simulation, IEEE Computer Society, 2001, pp.35-45.