# DATA BANDWIDTH REDUCTION TECHNIQUES FOR DISTRIBUTED EMBEDDED SIMULATION USING CONCURRENT BEHAVIOR MODELS

by

HUBERT ARTHUR BAHR
B.S.E.,University of Oklahoma, 1972
M.S. Cp.E., University Central Florida, 1994

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2004

Major Professor: Ronald F. DeMara

# ABSTRACT

Maintaining coherence between the independent views of multiple participants at distributed locations is essential in an Embedded Simulation environment. Currently, the Distributed Interactive Simulation (DIS) protocol maintains coherence by broadcasting the entity state streams from each simulation station. In this dissertation, a novel alternative to DIS that replaces the transmitting sources with local sources is developed, validated, and assessed by analytical and experimental means.

The proposed *Concurrent Model* approach reduces the communication burden to transmission of only synchronization and model-update messages. Necessary and sufficient conditions for the correctness of Concurrent Models in a discrete event simulation environment are established by developing *Behavioral Congruence* $\Psi_B(E_L, E_R)$ and *Temporal Congruence* $\Psi_T(t, E_R)$ functions. They indicate model discrepancies with respect to the simulation time $t$, and the local and remote entity state streams $E_L$ and $E_R$, respectively. Performance benefits were quantified in terms of the bandwidth reduction ratio $B_R=N/I$ obtained from the comparison of the OneSAF Testbed Semi-Automated Forces (OTBSAF) simulator under DIS requiring a total of $N$ bits and a testbed modified for the Concurrent Model approach which required $I$ bits. In the experiments conducted,

a range of $100 \leq B_R \leq 294$ was obtained representing two orders of magnitude reduction in simulation traffic.

Investigation showed that the models rely heavily on the priority data structure of the discrete event simulation and that performance of the overall simulation can be enhanced by an additional 6% by improving the queue management. A low run-time overhead, self-adapting storage policy called the *Smart Priority Queue (SPQ)* was developed and evaluated within the Concurrent Model. The proposed SPQ policies employ a low-complexity linear queue for near head activities and a rapid-indexing variable binwidth calendar queue for distant events. The SPQ configuration is determined by monitoring queue access behavior using cost scoring factors and then applying heuristics to adjust the organization of the underlying data structures. Results indicate that optimizing storage to the spatial distribution of queue access can decrease HOLD operation cost between 25% and 250% over existing algorithms such as calendar queues. Taken together, these techniques provide an entity state generation mechanism capable of overcoming the challenges of Embedded Simulation in harsh mobile communications environments with restricted bandwidth, increased message latency, and extended message drop-outs.

# TABLE OF CONTENTS

LIST OF FIGURES.....viii

LIST OF TABLES.....xi

LIST OF ACRONYMS.....xii

CHAPTER 1 INTRODUCTION.....1

1.1 Embedded Simulation.....1

1.2 Role of Communications in Embedded Simulation.....4

1.3 Motivation for Concurrent Models.....4

1.4 Contribution of Dissertation.....15

1.5 Overview of Dissertation.....16

CHAPTER 2 PREVIOUS WORK.....19

2.1 Overview.....19

2.2 Communication Reduction Strategies.....21

2.2.1 DIS Dead-Reckoning.....22

2.2.2 Vehicle Model Generation and Optimization for Embedded Simulation.....24

2.3 Synchronized Player Models .....26

2.4 Causality and Time Management.....30

2.5 Real-time, simulation-time, Scheduling, and Synchronization.....33

2.6 Queuing Strategies.....37

2.6.1 Linear List Priority Queues....44
2.6.2 Indexed Lists....44
2.6.3 The Calendar Queue....45
2.6.4 Relative Performance....47
CHAPTER 3 TECHNICAL PROBLEM DESCRIPTION ....53
3.1 Background....53
3.2 Operational Constraints....54
3.2.1 Force on Force Training Simulation. ....55
3.2.2 En route Mission Rehearsal....55
3.2.3 Situational Awareness....56
3.3 Computational and Communication Resource Tradeoffs....56
3.4 Situation-related Communication....58
3.5 OTBSAF as a Prototyping Testbed....64
3.5.1 OTBSAF Simulation Framework....65
3.5.2 Application....67
3.6 OTBSAF Scalability and Priority Queue Performance....71
CHAPTER 4 CONCURRENT MODEL APPROACH TO EMBEDED SIMULATION.…
76
4.1 Player Units (PU)....82
4.1.1 Concurrent Remote Model (CRM)....83
4.1.2 Difference Analysis Engine (DAE)....84
4.1.3 Adaptive Reference Model (ARM)....85
4.1.4 Instrumentation for Player Units....86

4.1.5 Situation Database....86
4.2 Processing of Discrepant Results....87
4.3 Concurrent Model Approach Design Criteria....90
4.3.1 General Criteria....91
4.3.2 Remote SAF Operator....93
CHAPTER 5 ANALYTICAL RELATIONSHIPS IN COMMUNICATION MECHANISMS....96
5.1 Correctness Characteristics....97
5.2 Performance Benefit Characteristics....102
CHAPTER 6 CONCURRENT MODEL CONSTRUCTS AND MECHANISMS ....106
6.1 Background....106
6.2 Integrated Model Execution....107
6.3 Isolated Model Functions....111
6.4 Smart Priority Queue Data Structure....113
6.4.1 Activity Feedback Counters....118
6.4.2 Sensing Cost....119
6.4.3 Filtering Costs....121
6.4.4 Correction Costs....123
6.5 Integration of SPQ in OTBSAF....124
CHAPTER 7 EXPERIMENTAL COMPARISON OF ALTERNATIVES ....127
7.1 Experimental Configurations....127
7.1.1 Concurrent SAF....127
7.1.2 Remote SAF Operator....129

7.1.3 SPQ used for Case Study....131
7.1.3.1 Queue Instrumentation....132
7.1.3.2 Statistic Counter Definitions....133
7.1.4 SPQ Evaluation with Statistical Distributions....133
7.2 Description of Scenarios....135
7.3 Presentation of Results....137
7.3.1 Concurrent SAF....137
7.3.2 Remote SAF Operator....138
7.3.3 SPQ....147
7.3.4 SPQ Integrated into OTBSAF....152
CHAPTER 8 CONCLUSIONS....154
8.1 Summary....154
8.2 Major Results....155
8.2.1 Bandwidth Reduction....156
8.2.2 Order(1) Priority Queue....157
8.3 Future Work....159
APPENDIX: EMBEDDED SIMULATION SYSTEMS....161
LIST OF REFERENCES....175

## LIST OF FIGURES

Figure 1. Entity to Observer Communications.....6

Figure 2. Scenario: Four Vehicles En route while Participating in an ES exercise.....7

Figure 3. Scenario: Bridge Crossing Requiring Change in Relative Position.....8

Figure 4. Scenario: Reestablishing Relative Positions Before Approaching Destination Location .....9

Figure 5. Communications Congestion at Time Intervals T1 through T11.....10

Figure 6. Communications Reduction Improving Estimate of Vehicle Location.....12

Figure 7. Communications Message Dropouts.....13

Figure 8. Concurrent Model Reports.....14

Figure 9. WWLAN Data Reduction Pyramid.....20

Figure 10. Dead-Reckoning Example Path Display.....23

Figure 11. SPM Human Surrogate DAE.....28

Figure 12. Scenario demonstrating causal event ordering [Fujimoto 1998].....30

Figure 13. Increase in Computational Load Versus Events Queued.....38

Figure 14. Priority Queue Taxonomy.....43

Figure 15. Calendar Queue.....46

Figure 16. Mean access time for Calendar Queue and Classic Hold Experiments.....49

Figure 17. Mean access time for Calendar Queue Up/Down experiments.....50

Figure 18. Distribution of search Length in DDC using a Linear Queue.....51

Figure 19. Search Length Distribution in a Calendar Queue.....51

Figure 20. Congruence Transfer Function.....60

Figure 21. Congruence Karnaugh Map.....61

Figure 22. Example Graphics Illustrating Tolerances.....63

Figure 23. SAFstation and SAFsim Components.....67

Figure 24. WWLAN Data Reduction Pyramid with Data Rates.....70

Figure 25. Dead-Reckoning.....77

Figure 26. Concurrent Model.....78

Figure 27: CRM.....82

Figure 28: Collective Interaction under Concurrent Model approach.....90

Figure 29. Concurrent Model Analysis.....96

Figure 30. Real-Time and simulation-time Relationship.....109

Figure 31: PO object Modification.....112

Figure 32. Smart Priority Queue (SPQ).....115

Figure 33. SPQ Insert Operation.....117

Figure 34. Revised SPQ for OTBSAF.....125

Figure 35: Concurrent SAF.....129

Figure 36: Remote SAF Operator.....130

Figure 37. Screen Shot 1.....136

Figure 38. Screen Shot 10.....142

Figure 39. Behavioral Congruence for Vehicle A11.....143

Figure 40. Behavioral Congruence for Vehicle A12.....143

Figure 41. Behavioral Congruence for Vehicle A13.....144
Figure 42. Behavioral Congruence for Vehicle A14.....144
Figure 43. Congruence Histogram for Vehicle A11.....145
Figure 44. Congruence Histogram for Vehicle A12.....146
Figure 45. Congruence Histogram for Vehicle A13.....146
Figure 46. Congruence Histogram for Vehicle A14.....147
Figure 47. Search Length Distribution in a SPQ.....149
Figure 48. SPQ Performance for Classic Hold.....150
Figure 49. Calendar Queue for Classic Hold.....150
Figure 50. SPQ Performance In Up/Down.....151
Figure 51. Calendar Queue Performance for Up/Down.....151
Figure 52. SPQ Integer Performance for Classic Hold.....153
Figure 53. OTBSAF PQ Performance for Classic Hold.....153
Figure 54. Screen Shot 2.....167
Figure 55. Screen Shot 3.....168
Figure 56. Screen Shot 4.....169
Figure 57. Screen Shot 5.....170
Figure 58. Screen Shot 6.....171
Figure 59. Screen Shot 7.....172
Figure 60. Screen Shot 8.....173
Figure 61. Screen Shot 9.....174

# LIST OF TABLES

Table 1. Queue Percent of Computation for Given Queue Sizes.....40

Table 2. Prioritized Execution.....41

Table 3. Results of Jones Study.....48

Table 4. Situation Database.....59

Table 5. Current Communications Packets.....69

Table 6. ModSAF 3.0 Benchmark Results [Roberts 1998].....71

Table 7. OTBSAF Benchmark Results.....72

Table 8. OTBSAF Benchmark Results - Part II.....74

Table 9. Comparison Equivalent for Optimization Calculations.....123

Table 10. Statistical Counter definitions.....132

Table 11. Message Counts for Remote Operator.....138

Table 12. Message counts for Benchmark.....140

Table 13. Path Counts and Comparison Equivalents.....148

# LIST OF ACRONYMS

| | |
|---|---|
| ARM | Adaptive Reference Model |
| CCTTSAF | Close Combat Tactical Trainer Semi-Automated Force |
| CM | Concurrent Model |
| CRM | Concurrent Remote Model |
| DAE | Difference Analysis Engine |
| DES | Discrete Event Simulation |
| DIS | Distributed Interactive Simulation |
| ES | Embedded Simulation |
| FIFO | First-In First-Out |
| FSM | Finite State Machine |
| GMT | Greenwich Meridian Time |
| GPS | Global Positioning System |
| GPS | Inter Vehicle Simulation Technology |
| LAN | Local Area Network |
| M&S | Modeling and Simulation |
| ModSAF | Modular Semi-Automated Force |
| OneSAF | One Semi-Automated Force |
| OTBSAF | OneSAF Testbed Semi-Automated Force |
| PDU | Protocol Data Unit |
| PO | Persistent Object |

| | |
|---|---|
| PVD | Plan View Display |
| SAF | Semi-Automated Force |
| SME | Subject Matter Expert |
| STO | Science and Technology Objective |
| UDP | User Datagram Protocol |
| UTM | Universal Transverse Mercator |
| WAN | Wide Area Network |
| WWAN | Wireless Wide Area Network |

# CHAPTER 1 INTRODUCTION

The Concurrent Model (CM) approach for embedded simulation is a technique for reducing the inter-platform communications requirements of Distributed Interactive Simulation (DIS) when used for Embedded Simulation (ES). This dissertation first identifies critical aspects of performance and fidelity of Embedded Simulation. It then introduces the CM approach as a proposed solution for the restrictive communications requirements of mobile platforms using ES. It then describes a prototype program to test the proposed concept and reports on the results of the first phase of this program.

## 1.1 Embedded Simulation

Historically, computer modeling and simulation have been primarily used for three basic purposes:

1. Analysis of behavior of completed events (i.e., understanding of significant, incompletely understood events),
2. Prediction of future behavior of complex systems (i.e., design), and
3. Training in operation of equipment or mission accomplishment

These traditional applications of Modeling and Simulation are centered around advisory functions, typically performed off-line. On the other hand, ES, derives significant advantages from its use in operational real-time tasks.

*Embedded Simulation* (ES) is the integration of simulation technology with operational systems, allowing the operators of those systems to interact with both the real world and the virtual world as if both were integrated. This provides the operator with capabilities in the real world beyond his/her immediate perception range. Furthermore, it allows for high fidelity predictions of behavior of other elements in the real world, which would improve decision-making in tactical or strategic contexts. The degree of real world replacement by the virtual world is dependent on the application. For example, in a military system application, the envisioned concept would be as follows:

> *Embed the capability in the vehicle to allow a "virtual world" to be displayed to the crew and to have virtual interaction with vehicle subsystems in support of Mission Rehearsal, Battlefield Visualization, Command Coordination, and Training.*

We expand on these below as they form the basis for the applications to be optimized:

- Mission Rehearsal — Rehearsal or simulation of actions prior to undertaking the mission, on the operational equipment to be used in that specific mission, is a highly desirable capability made possible by ES. The extent of rehearsal could range from a single commanders scenario for Course Of Action Analysis, to having all crew members participating in an exercise.
- Environment Visualization — ES can support visualization of the simulation environment, which provides significant advantages in situational awareness. Visualization could be as simple as a display of unit and enemy locations on a two-dimensional map based on the last reported locations/sightings. More ambitiously, it could include a time-updated best estimate of the locations/status on a three-

dimensional display with integrated current sensor data at sufficient resolution for all critical decisions.

- Command Coordination — ES can be used to vary the graphical representation of plans to the automated tracking and notification of coordination events. The simulation of the execution of the commanders intent could be displayed to both the commander and the units executing the commands. If any differences occur between the commanders intent and the execution the units would notify the commander of the actual execution and the commander would either acknowledge their recommendations or modify his commands.
- Training — ES can support having one member of the crew located at his station refining his individual skills with simulated crew members against simulated forces. Furthermore, the operational scenario could be simulated in a stationary vehicle and through repeated executions of this scenario, be used as a drill to refine capabilities.

The study of embedded simulation technology is a refinement and enhancement of simulation concepts for real-time use in the space and power constraints of operational systems. In the operational environment, embedded simulations will strive to convert inputs from multiple sensors into a form to give the operators an enhanced interface with their environment. It also includes the capability to project the future, and serve as a basis for decision aids. Achievement of these objectives will require significant capabilities for high performance communication in a distributed simulation environment.

## 1.2 Role of Communications in Embedded Simulation

Communication mechanisms in an ES system must provide the means for the various vehicles involved in the mission to share the benefits of the distributed simulation resources. This imposes real-time constraints on the bandwidth, latency and connectivity of the data transfer mechanisms. Realism constraints place an additional burden of precise data ordering on this communications subsystem. Current computer generated force systems such as One Semi-Automated Force (SAF) (OneSAF) Test Bed (OTBSAF) and Close Combat Tactical Trainer (CCTT) SAF (CCTTSAF) are designed to provide only one model of each virtual player in the exercise. Each of these models periodically generate state information that is broadcast to all the other participants. Live players interact with these models by operating a vehicle simulator which generates the state information in the same format. In this environment, each entity is represented to all the other entities by the sequence of state messages that are broadcast periodically. As long as the communications bandwidth is sufficient, and has minimal latency, entities can be added to the ES exercise by including additional sources on the network. To date most of the work has been dedicated to providing adequate bandwidth and low latency, using local area network technology and User Datagram Protocol (UDP) packets.

## 1.3 Motivation for Concurrent Models

A critical parameter impacting to the amount of bandwidth required is the frequency of the entity state messages. One method used to reduce this frequency is to use *dead reckoning algorithms*. These techniques have proven sufficient for training with current

systems such as OneSAF and CCTTSAF. However, goals exists to continue to expand in numbers, locations, and types of participants. One expansion is in the number of entities involved in a given training exercise. Another is to have entities participate from locations worldwide. A third is to have participation of crews from their actual vehicles while they are moving.

During the assessment of the bandwidth requirement under current techniques [Goblick 1996][Valle 1997] verses current capacity [Bahr 1994A] it is apparent some significant bandwidth reductions are needed to achieve the goal of ES. The concurrent model approach [Bahr 1996] postulates taking the dead reckoning concept to the limit. It does this by assuming that it has clones of the interacting entities residing at each location. Thereby, given that all clones are receiving the same stimuli at the same time they will react to the stimuli in unison. As long as this condition is maintained, no entity state messages would need to be broadcast. This has the potential to drastically reduce the required update traffic on the network. The network is then used primarily to introduce new common information into the exercise, much as a tactical command and control system is used to specify new objectives to be performed next.

The value of ES is enhanced when the vehicle is free to move. Hence the requirement for wireless communications between the users of the simulation is originated. Wireless communications introduces additional impacts on bandwidth, latency and continuity over the traditional Local Area Network (LAN) used for Distributed Interactive Simulation (DIS) [Fullford 96]. While many new strategies are being developed to share more of the

available radio spectrum with multiple users using space, time, and code division multiplexing techniques, the demand for this spectrum continues to expand even faster.

In addition, the use of wireless communications allows the participants freedom of movement, many times to locations unfavorable for communications. Increased emphasis has been placed on operation in the urban environment, where communications can be problematic. The avoidance of line of site, and the operation in the urban environment with its higher incidence of man made noise all contribute to situations where there can be extensive communications outages, or reduction in the available bandwidth. If a method can be developed that allows minimal disruption of command and control coordination with continued situational awareness in this environment, it would greatly enhance the effectiveness of ES.

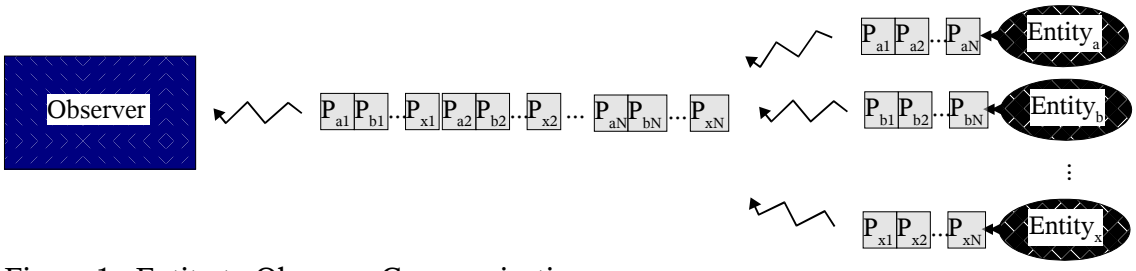



Figure 1. Entity to Observer Communications

Figure 1 provides a sketch showing the relationship between multiple entities labeled **a** through **x**, who, while moving away from the stationary observer, are returning their state message packets labeled $\mathbf{P_{a1}}$ through $\mathbf{P_{xN}}$ over a shared wireless communications link. $\boldsymbol{P_{ij}}$ is the packet with $\boldsymbol{i}$ denoting the originating entity IDs ***a*** through ***x,*** and $\boldsymbol{j}$ denoting the sequence numbers of the packets labeled ***1*** to ***N***. Since this communications link is

shared, if it is already in use by another entity, each remaining entity must queue up their packets until the link becomes available. The packet stream that is depicted in Figure 1 indicates a round-robin protocol in which each entity transmits its next packet in turn. For ease of analysis, we will assume this type of channel sharing unless otherwise stated throughout this dissertation. Unless otherwise stated the observer is assumed to be some finite distance away from the entities reporting their current state.

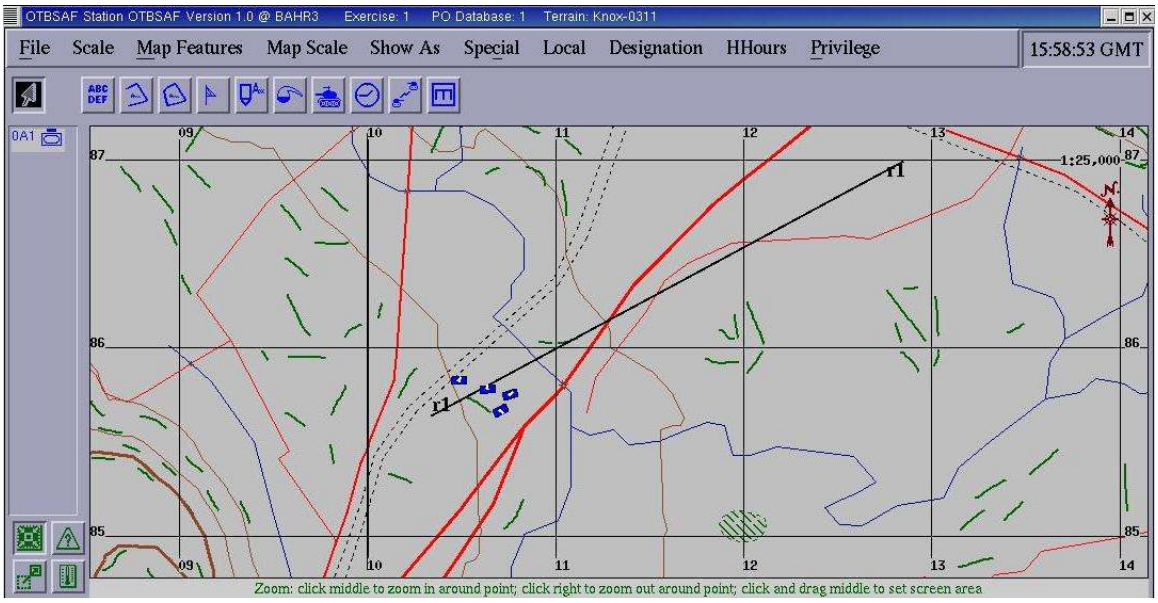


Figure 2. Scenario: Four Vehicles En route while Participating in an ES exercise

Figures 2, 3, and 4 display snapshots of a short scenario as displayed in the Plan View Display (PVD) of OTBSAF [SAIC 2001B]. The title bar of the window gives information about the computer running the simulation, the version of OTBSAF, exercise and Persistent Object (PO) database numbers and the terrain database used. The next bar has a set of pulldown menus and a display of the current wall clock time. The next bar select various modes of use for this display. The section on the left of the screen provides buttons for the various units available for display on this station. The major section of

the display shows a topographical map based on the identified terrain database at a selected Map Scale. In Figure 2, this scale is 1:25,000. Around the sides of the map display the grid is labeled with Universal Transverse Mercator (UTM) map grid coordinates. In this scenario a platoon designated "100A1" has been told to move "cross-country" in a wedge formation along the route indicated by the line designated "r1." "Cross-country" is a term that signifies a movement forward on the closest route along the path depicted by the line allowed by the terrain as opposed to "road march" that signifies a movement forward on roads closest to the path depicted by the line. Figure 2 shows the platoon in wedge formation approximately two minutes into the exercise.

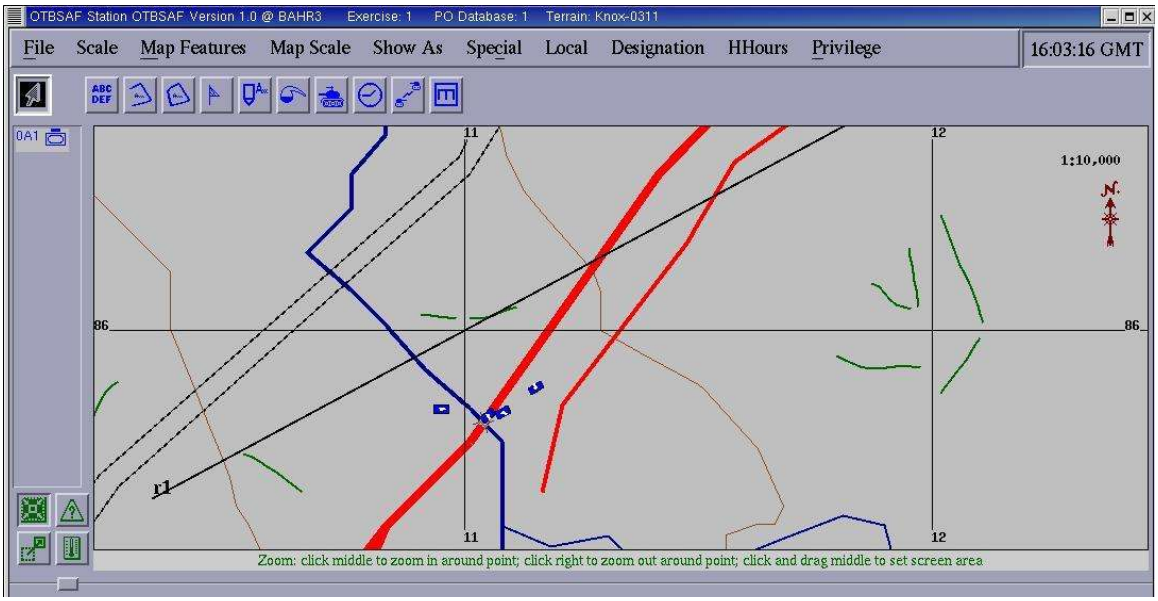



Figure 3. Scenario: Bridge Crossing Requiring Change in Relative Position

Each vehicle's position is indicated by a small blue icon approximating the shape of vehicle but at an enlarged scale sufficient to be easily viewed on the display. It's orientation approximates the vector of the last movement. Even though the route is indicated as a straight line, the platoon must navigate the terrain as depicted on the

topographical map. Between the platoons current location and its objective indicated as the terminal point of "r1" there is a river that must be crossed. Figure 3 shows that the platoon is using a bridge to cross the river and is continue toward its objective. It shows the map scale changed to 1:10,000 to show more detail about the bridge crossing. Figure 4 shows the platoon about a minute from its objective, again at a zoomed in scale of 1:10,000. Each grid line represents 1 kilometer. This scenario will be used throughout this dissertation to illustrate proposed communication techniques.

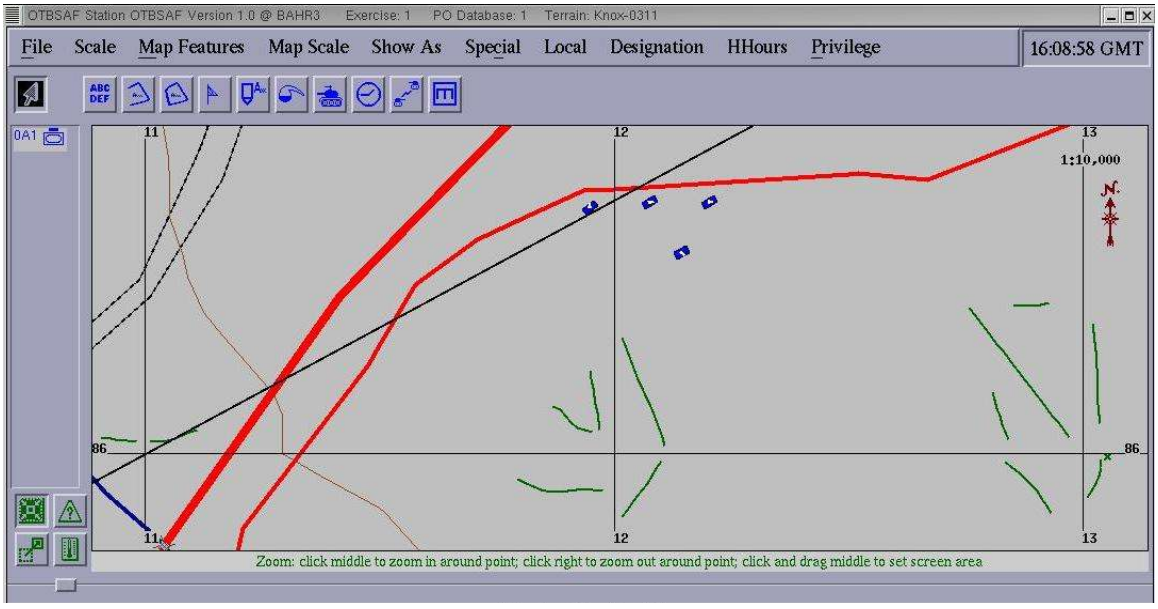



Figure 4. Scenario: Reestablishing Relative Positions Before Approaching Destination Location

Figures 5, 6, 7, and 8 display four different communications scenarios in the case of limited bandwidth. These figures are based on the previous exercise scenario, however the plan view display has been cropped and expanded to show more detail about the total scenario and the entity state reports verses the time they are received over the duration of

the exercise. The total exercise lasts about 12 minutes. It starts at the locations indicated by the vehicle icons, and the individual vehicles follow the paths indicated by the separate colors. Figure 5 displays a condition where communications channel's bandwidth is only 50% of that required to transmit all the data in real-time.

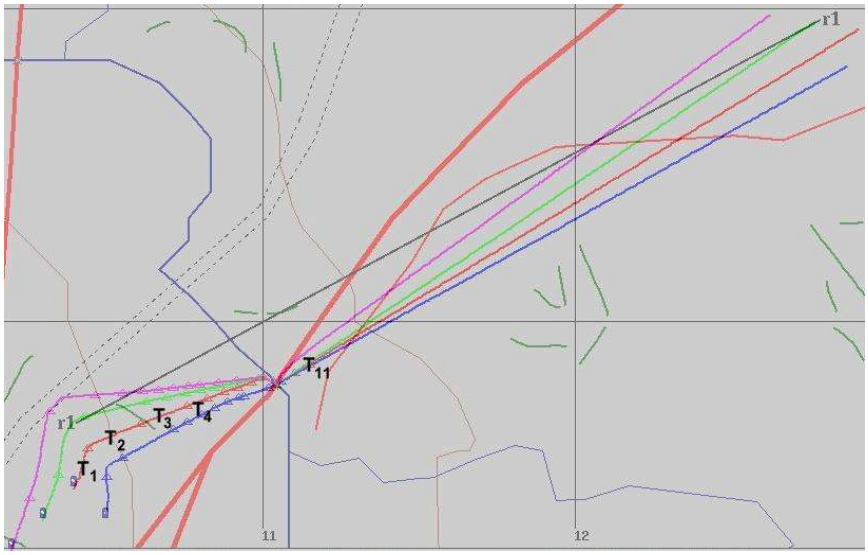


Figure 5. Communications Congestion at Time Intervals $\mathbf{T_1}$ through $\mathbf{T_{11}}$

In the case where all the messages are queued and transmitted as soon as possible, then *congestion* results. Congestion impacts the time that the message is received at a distant observer. The difference between time that the event occurs and the time that the event is observed is called o*bservational latency* which we will designate as $t_{observe}$. The component of $t_{observe}$ that is due to the waiting time to put the message on the communications link is designated as $t_{wait}$. In this set of four figures, we are assuming that all other components of latency are negligible for this discussion. As such the other three figures show the observed time as equal to the transmitted time. The observed time **T** is depicted by $\mathbf{T_1}$ through $\mathbf{T_{11}}$ on all the figures corresponding to 1 through 11 minutes

after the start of the exercise. The left terminus of each line is the position of the vehicle at the start of the exercise, and the right terminus is at the end of the exercise 11 minutes and 58 seconds (0:11:58) later. A small triangle is placed on each path corresponding to the observed position at each minute throughout the scenario. Due to potential confusing overlap not all positions are labeled, but each minute is indicated by a small triangle on the corresponding path. Note that since all the vehicles cross the same bridge, all paths overlap at the bridge. In Figure 5, the observed positions are further and further away from the actual positions as indicated by the discrepancy in the location of the entities. Since we assumed that the available bandwidth is 50% of that required $\boldsymbol{BW_a = M_a/t = 0.5*BW = 0.5* M/t}$, at the end of the exercise only 50% of the data $\boldsymbol{M_a = M/2}$ has been received. Where $\boldsymbol{M}$ is the number of messages and $\boldsymbol{t}$ is the time to deliver the messages and $\boldsymbol{BW_a}$ is the available bandwidth. To deliver the remaining messages $\boldsymbol{M/2}$ at the same rate an equal amount of time $\boldsymbol{t}$ will be required. Thus the final position is delayed a time equal to the exercise duration. The $\boldsymbol{t_{wait}}$ for this message would be equal to the length of the exercise $\boldsymbol{t_{ex}}$ = 0:11:58. Assuming an uniform transmission rate the average waiting time $\boldsymbol{t_{ave}} = \frac{\sum_{i=1,N} t_i}{N} = \boldsymbol{t_{ex}/2}$ = 0:5:59, where $\boldsymbol{t_i} = \boldsymbol{t_{wait}}$ for each individual message. While this may be acceptable for post exercise analysis, it is not acceptable for any real-time interactive applications such as those proposed for embedded simulation. Real time interaction requires feedback within the attention span of the observer, Based on my observations over 20 years of instrumenting live simulations this is nominally within 2-3 seconds, and for many situations even shorter latencies are required.

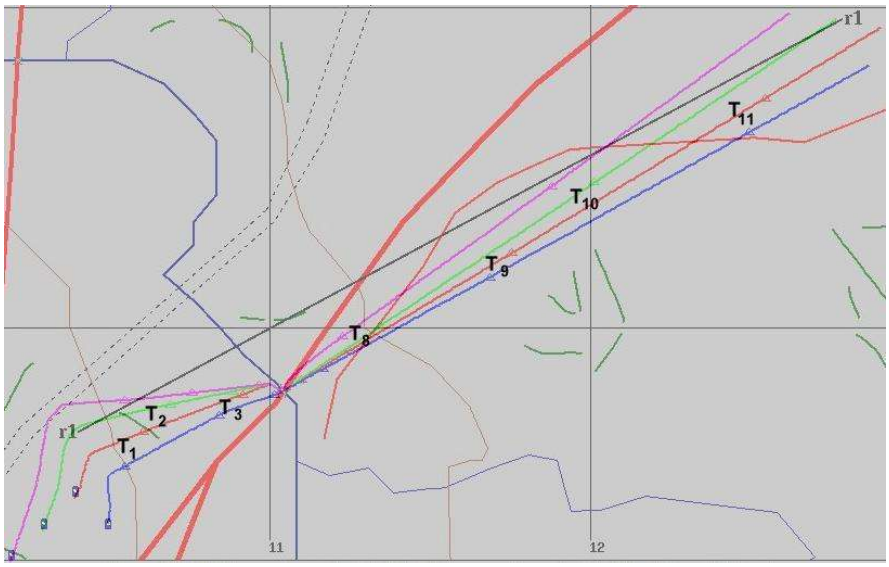



Figure 6. Communications Reduction Improving Estimate of Vehicle Location

An alternative to message backup is to reduce the number of messages to correspond to the available bandwidth by changing the rate of the information that is exchanged. An example of this approach is portrayed in Figure 6. In this case, messages are staggered using time division multiplexing such that the odd numbered vehicles transmitted at odd times and the even numbered vehicles transmitted at even times. Here there are only two triangles representing the reported locations for each labeled time. The position update interval between each report is twice as long as would be otherwise if more bandwidth was available. This still leads to undesirable consequences due to inadequate bandwidth, however, it may be preferable than allowing backup during real-time interaction. Another technical challenge facing communication in ES is common in wireless links. This is the problem of message dropouts. Dropouts are the condition where no communication is possible due to interference, or unavailable due to inadequate signal strength. Figure 7 depicts a message dropout between time $\mathbf{T_2}$ and $\mathbf{T_{10}}$. The result is that

no position reports are received over this period which is undesirable for both post exercise analysis and real-time interaction.

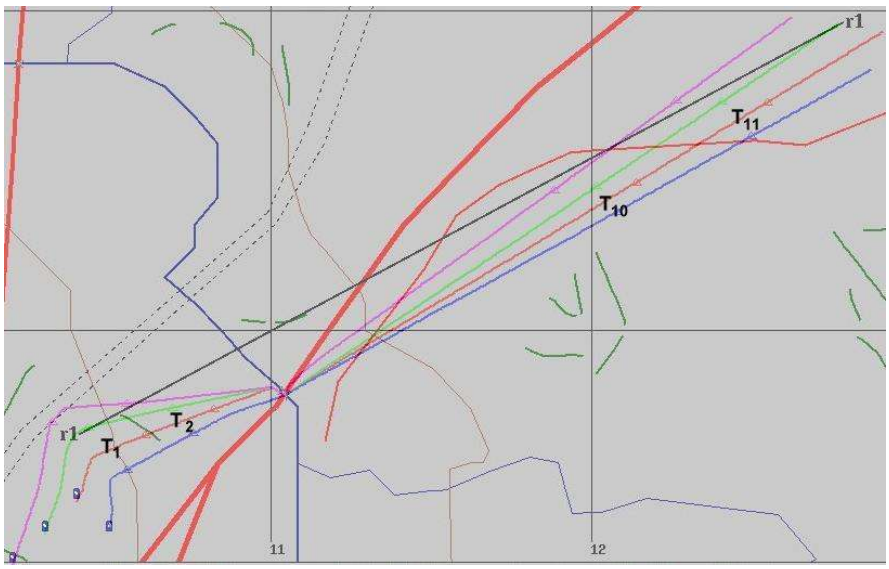


Figure 7. Communications Message Dropouts

As a solution to these problems, Figure 8 represents the results for each of these scenarios under the Concurrent Model Approach proposed in this dissertation. As illustrated each report that is used by the observer is behaviorally correct for the time of the report and there is also no loss of information, because bandwidth requirements have been reduced. This proposed solution utilizes the communications channel to transmit more effective information based on entity behavior rather than entity position. Thus, it can compensate for longer latencies. This approach was initiated to compensate for the differences in the communications resources used for virtual simulations using the DIS protocol and live simulations using highly customized special purpose protocols. Let ***N*** denote the number of bytes required to transmit entity state information for a typical DIS scenario, and ***I***

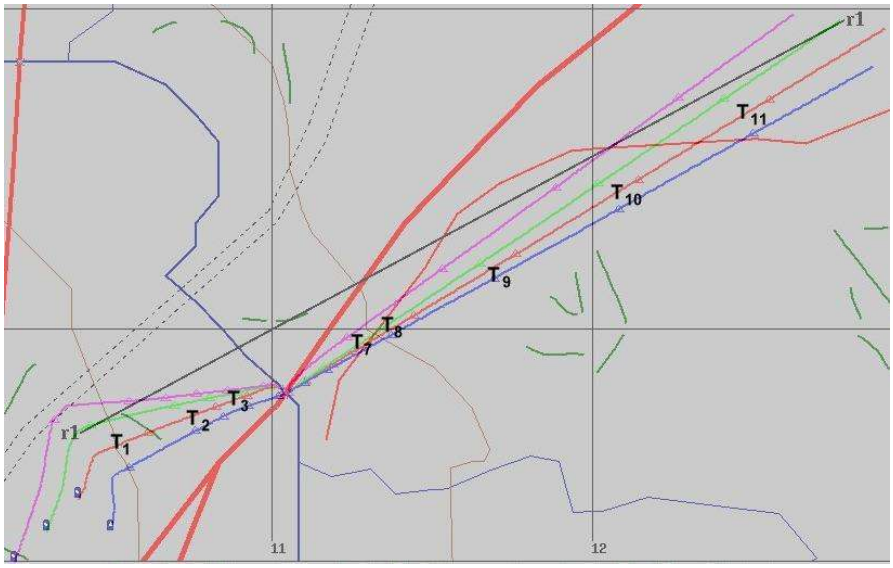



Figure 8. Concurrent Model Reports

denote the number of bytes required to transmit the entity state information for an equivalent live scenario. The ratio $B_R = N/I$ which indicates the increase in traffic under simulation environment can exceed $B_R > 100$ [Goblick 1996] [Valle 1997] [Bahr 1994A]. Using $B_R=100$ as a requirement for ES bandwidth ratio then a goal for the amount of data transferred under the Concurrent Model Approach can be set to $N/100$, rather than $N$. In addition, the packets that are transmitted under the Concurrent Model Approach are not merely position reports, but scheduled changes to model parameters that indicate properties to compensate for latency and message dropouts. These additional properties are among those that are developed in the dissertation to achieve *temporal congruence* and *behavioral congruence* between the originating entity state stream and the observed entity state stream, as will be defined in Chapter 5.

## 1.4 Contribution of Dissertation

This dissertation develops and validates the Concurrent Model Approach as a method of managing communication requirements and hiding the latency in ES. This establishes the need for causality, repeatability, and synchronization in order to be feasible. It uses discrete event simulation, normally reserved for non-real-time models, to provide repeatable simulations in a real-time environment. It establishes the validity of operating an environmental visualization in soft real-time as opposed to hard real-time, thereby allowing more efficient algorithms to maximize model size for the given computational environment. It introduces a novel data structure to maintain the most important events in priority order during the simulation. Finally it establishes, rationale for de-coupling the display update rate from element position location calculations update rate, thereby allowing a higher peak to average ratio for useful computations per clock cycle. These are achieved and demonstrated by providing a testbed that can continue to support exploration and validation of the Concurrent Model Approach for additional applications.

Latency hiding is accomplished by performing time-critical computations on each local platform, rather than computing them at one location and broadcasting the results to others. This change has the effect of reducing the characteristics of the communications link impact on ES performance. Discrete event simulation is used to allow computations to be scheduled on an as needed basis as opposed to a cyclic basis prevalent in real-time simulations. However, this introduces the need for an efficient event list implementation to keep the scheduling algorithm from dominating the execution demands of the

simulation. The use of discrete event simulation ensures that all computations are performed in the same order, thereby enabling causality, and repeatability.

Using the testbed developed, the above concepts and rationale are then tested by applying them to the application of *En route Mission Rehearsal* [Lawlor 2002]. The scenario is that during the multi-hour flight en route to a destination, the unit would rehearse its plan as an ES exercise. This is an ideal application for the concurrent model approach as it requires wireless communications over long distances. This particular application is for the concurrent model approach is to allow the remote operators to interact with the deployed unit without requiring the real-time transmission of typical DIS traffic between the participants.

The event list capabilities are demonstrated by testing with various event insertion distributions, and comparing results for the same set of distributions against most known implementations in previous studies. [Ahn 1999][Ronngren 1997][Tan 2000]

## 1.5 Overview of Dissertation

Chapter 2 covers previous work relevant to this dissertation. It discusses previous attempts at reducing the amount of information that is transmitted between participants of a DIS exercise. It then develops methodologies applied to causality and time management, and characteristics of real-time, simulation-time and scheduling techniques. Finally, Chapter 2 covers the background literature on event list management.

Chapter 3 develops the problem description by introducing Embedded Simulation, the required technologies, and its current status. It then examines performance constraints and computation versus resource tradeoffs for the three potential applications of force-on-force training, en route mission rehearsal, and situational awareness. It then examines the use of OTBSAF as a testbed for these concepts, and finally looks at the limitations of current priority queue structures for application for the proposed Concurrent Model approach.

Chapter 4 presents the Concurrent Model approach by comparing it to dead-reckoning. It then explains the functions of each of the CM components. It then addresses the impact on discrepancies between modeled and measured results. Finally it introduces the design criteria critical to the various stages of investigation of the concept.

Chapter 5 presents the analytical relationships by providing the definitions, theorems and proofs. It initially defines the correctness characteristics of *Behavioral and Temporal Congruence*. It ends by discussing the performance benefit characteristics of bandwidth reduction, latency immunity and outage immunity.

Chapter 6 presents the constructs and mechanisms of the CM approach. It addresses the changes that need to be made to OTBSAF to explore the CM approach and details the relationship between simulation-time and real-time and the characteristics of soft-real-time used by the CM approach. Finally it addresses the instrumentation of the Smart Priority Queue and its integration into OTBSAF for the CM approach.

Chapter 7 presents the experimental comparison of the alternatives investigated in this dissertation. It initially describes the experimental configurations of the Concurrent SAF, the Remote SAF Operator, the Smart Priority Queue and statistical distributions used for evaluation of the queue. It then describes the scenario used to drive the experiments for the Concurrent SAF, and the Remote SAF Operator. Finally it presents the experimental results of each experiments.

Chapter 8 summarizes the contribution of study, the major results and future work. It then presents the major results for bandwidth reduction and priority queues. Finally it introduces topics for future work with suggested approaches.

# CHAPTER 2 PREVIOUS WORK

## 2.1 Overview

Simulations whether live, virtual or constructive have become valuable tools for training, mission rehearsal, and course of action analysis. Initially they were developed, and used for specific purposes. Live simulation has been used for small team real-time training, virtual simulation for individual and crew training on the use of the equipment, and constructive simulation for command and control training or war planning. As the simulations have improved, and the physical size of the computational resources continue to shrink, the trend has been made to incorporate simulations with the actual operational equipment. The U.S. Army's Inter Vehicle Simulation Technology (INVEST) Science and Technology Objective (STO) investigated the incorporation of previous developed simulations with combat vehicles and demonstrated feasibility of providing a virtual training environment in the vehicle [Bahr 1998] [Klingensmith 1998]. One outcome of the INVEST-STO research was the need to address the communication requirements of ES. It became apparent that the current DIS paradigm of using only one generating source for each entity could exceed the capability of Wireless Wide Area Networks (WWANs) [CBO 2003] [Tiernan 1995] [Valle 1997]. The concurrent model approach

changes this paradigm to multiple local generating sources for each entity to take advantage of the technologies proven to work for local area networks.

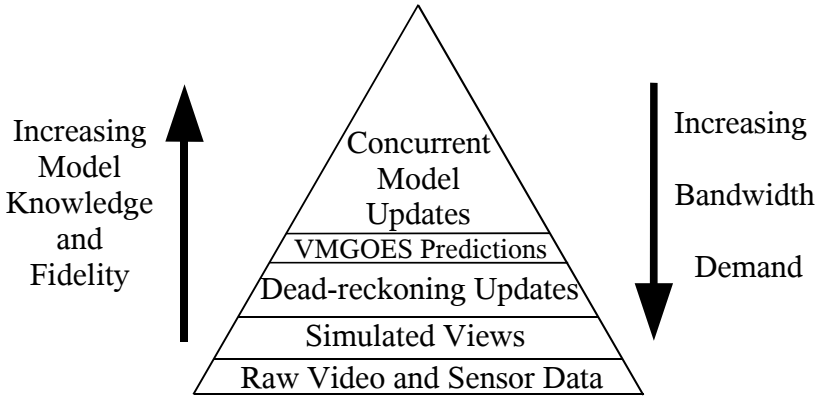



Figure 9. WWLAN Data Reduction Pyramid

Figure 9 provides a visual representation of the communications bandwidth hierarchy of various strategies previously implemented to address the sharing of state data between the different entities in a simulation. The height of the pyramid is proportional to the added complexity, while the width represents the required bandwidth. The maximum bandwidth is required when transmitting the data as it is generated as indicated by the raw data level of the pyramid. One can avoid transmitting some raw data by instead generating simulated views as depicted by the second layer of the pyramid. The *Dead-reckoning* approach can reduce the simulation bandwidth by an order of magnitude [Bassiouni 1997] and is commonly used on LANs. *VMGOES* pushes this reduction still further by applying advanced modeling techniques using neural networks [Gerber 2001] [Henninger 2000B]. The proposed *Concurrent Model approach* takes this to the limit by eliminating the transmission of entity state data, but replacing it with local generation and the communication of congruence information to maintain synchronization between all

data generators. Each of these previous works are reviewed in detail in subsections later in this chapter.

To establish the critical issues for the proposed approach, the following areas are investigated in this dissertation. First, the current technologies in simulation communications reduction strategies are reviewed. Second, the problems of synchronizing multiple entity generators is assessed. This investigation centered around an in depth look at the area of causality and simulation time management, which pushed for further exploration into the issues of real-time versus simulation-time scheduling strategies. As the scheduling strategy became of clearer importance, the priority queue realization was found to be critical. This motivated an improved data structure that could improve the scalability and throughput of the task scheduler used in ES.

## 2.2 Communication Reduction Strategies

These first two approaches address taking the periodically generated entity state messages, and eliminating those that do not add significant new information to the entities current state. This adds moderate complexity to both ends of the transmission stream, but reduces the amount of data that needs to be transferred from the source entity to the observer. The first of these is implemented in OTBSAF and is part of the baseline value of ***N*** in the DIS stream.

### 2.2.1 DIS Dead-Reckoning

*Dead-reckoning* [Fullford 96], a term first coined for navigation, is an abstraction of the physics of motion. It is based on Newtonian Physics in that a moving object does not change direction unless some force is applied. In DIS, this concept is used to maintain the communications of state information between various participants. Each vehicle in motion keeps detailed state information about its position and movement. This is used to populate a dead-reckoning model of itself and every other participant with which it may interact. Each time a vehicle needs to update the position of other participant vehicles in the simulation, it updates the position estimate by extrapolating a new position based on a linear distance model $\boldsymbol{x = v\Delta t + x_0}$ where the elapsed time since the creation of the latest information is $\boldsymbol{\Delta t}$ and $\boldsymbol{v}$ denotes the velocity and $\boldsymbol{x_0}$ denotes previous position. In the simplest case, the dead-reckoning model is based on position, time, and the velocity vector transmitted by the other participant. To make sure other participants in the exercise have a valid dead-reckoning model, each participant is constantly comparing its own position with its own dead-reckoned position. As long as the displacement between the two positions does not vary more than a predetermined threshold then the dead-reckoning model is left unchanged. Once this displacement exceeds the predetermined threshold this participant broadcasts a message to all other participants that provides an updated time, position, and velocity vector of motion. In some cases, higher order models, using acceleration in addition to velocity, are used to try to reduce the number of updates that are required [Gerber 2001]. While dead-reckoning reduces update traffic,

more packets are required to be transmitted than can be supported in an ES environment [Bahr 1996].

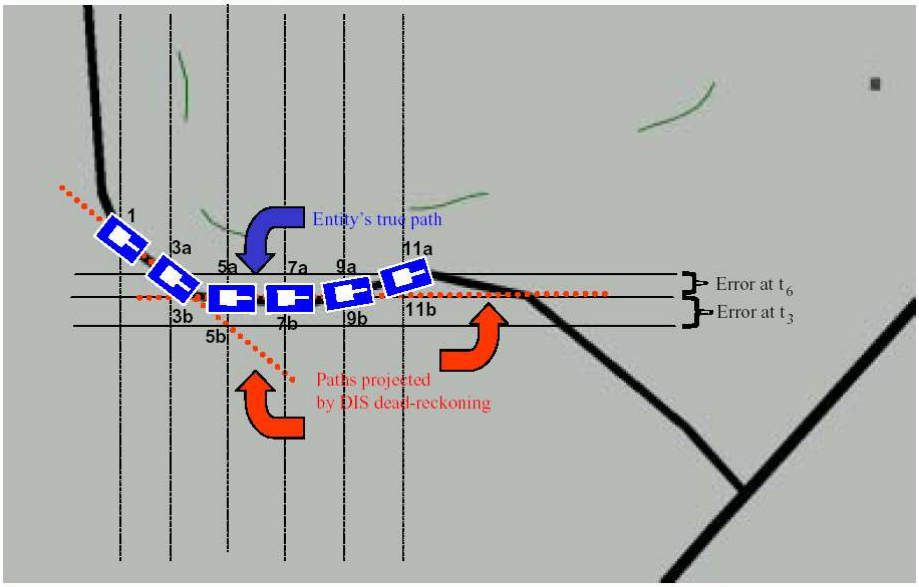



Figure 10. Dead-Reckoning Example Path Display

Figure 10 from [Henninger 2000B] provides an example of dead-reckoning. In this figure the true path is designated $\mathbf{I_a}$ and the dead-reckoned path is designated by $\mathbf{I_b}$ where **I** denotes the integers 1.. 11 which represents different times. At time $\mathbf{t_1}$ both the true path and the dead-reckoned path coincide. At time $\mathbf{t_3}$ the difference between the paths exceeds the dead-reckoning threshold so entity state PDU is generated and the remote user receives an updated position and velocity. The path it generates corresponds the the dead-reckoned path $\mathbf{I_b}$. Again at $\mathbf{t_6}$ the threshold is exceeded and again a new entity state PDU is generated to update the remote paths. The official exercise position is the paths

generated by dead-reckoning as that is the only path transmitted on the network. The only simulation that sees the actual path is the local host.

Bassiouni [Bassiouni 1997] shows an example of dead-reckoning that reports a data reduction of 76 but states that generally the data reduction is greater than one order of magnitude but varies dependent on the activities of the host and other exercise variables. In the scenario shown in Chapter 1 we observed a reduction of between 10 to 15. We will cover this in additional detail during our comparison to the Concurrent Model approach in Chapter 4.

### 2.2.2 Vehicle Model Generation and Optimization for Embedded Simulation

The Vehicle Model Generation and Optimization for Embedded Simulation (VMGOES) project [Gonzalez 1998] [Henninger 1998] [Henninger 2000] concentrates on reducing the communications bandwidth by providing higher fidelity models of manned vehicles. Henninger [Henninger 2000B] reports that the primary objective was to develop a neural network based movement model that could be used in lieu of a Newtonian-based DIS dead-reckoning model to support the synchronization of the entities states in an embedded training exercise. In support of this objective, a coherent framework for learning and testing such models was developed, and two important issues were addressed. The first issue was how to effectively model a near-term movement skill model from real-time data and how to measure the performance of this model. The second issue was whether this approach would generalize to human driving. The initial models were developed in a simulated testbed environment. First, the models were tested

on movement methods that were not used in training, but similar to those used in training. The best performing models in the first case gave an average ESPDU reduction of approximately 28% over current distributed simulation dead-reckoning methods. They generalized approach did not yield any reduction. When this models were developed using Subject Matter Expert (SME) generated data for testing that was not used in the training of the neural network-based near-term movement models. The best performing models, in this instance, resulted in an estimated ESPDU reduction of 67%. Also evaluated was the potential for generalization across drivers by applying the movement models developed from one SME to a second individual. The model results, in this instance, did not reduce ESPDUs. From the CM approach viewpoint this yields two important observations. First that better models can be achieved over the current OTBSAF implementations, and second trained SMEs tend to be better subjects than the OTBSAF models.

Here, a system that extends the dead-reckoning concept to the behavioral level is used to anticipate position changes, this provides benefits beyond just maintaining current velocity due to anticipating the operators behavior based on the context of the current activities. A methodology used called *Context-Based Reasoning*, establishes that each observable behavior has its own associated actions that can represented by an active *context*. For example, a tank, operated by a crew conducting a tactical road march, would be exhibiting behavior associated with following a road, such as being near the road and proceeding in roughly the same direction as the road [Gerber 2001]. Since modeling the system based on each of the contexts, and the human reactions based on

these contexts involves significant detail, the VMGOES researchers developed the models using a machine learning technique known as *Learning-by-Observation* which was shown to be effective.

Gerber [Gerber 2001] reported that the efficient synchronization of human-controlled ground vehicle with a VMGOES behavioral model was demonstrated in a ModSAF testbed. This demonstrated improvement over dead-reckoning. He also reported development of a Learning-by-Observation methodology to infer low-level actions of a ground vehicle operator in real-time was successful as well as the establishment of a software testbed to implement and evaluate the concept. The approach demonstrated appears to provides a mechanism to link various networks together to provide a more complete model. One drawback to these approaches is they assumed no prior knowledge of the subjects intent. As shown in the next section some prior knowledge is available and can help in the mechanization of CM approach.

## 2.3 Synchronized Player Models

Synchronized Player Models (SPM) research addressed an important element related to the proposed Concurrent Model approach: providing repeatable models. Repeatability was obtained by modifying the predecessor of OTBSAF, called the Modular Semi-Automated Forces (ModSAF), to provide a simulation that produced identical simulation behavior on repeated runs.

The SPM project [McHale 1998] addressed the problem of providing deterministic Computer Generated Forces (CGFs) as part of the solution to reduce the need for communications. Assuming that all other aspects of a simulation environment are synchronized, synchronization of the parallel models used by individual platforms can be achieved by using a repeatable implementation of the CGF. A CGF implementation is said to be *repeatable* if the simulation events (e.g., vehicle location events, firing events, damage events) occur at the same simulation time in each run of the same scenario. The SPM researchers chose to modify ModSAF to provide a repeatable mode of execution. They observed that repeatability can be achieved by modifying the scheduling of simulation events, the generation of stochastic events, and the control of distributed events as follows.

First, they modified ModSAF 3.0 by changing the event scheduling by essentially severing the simulation to real-time clock dependency, altering the simulation queue to be event driven, and leaving all non simulation functions on the system clock, while the simulation functions advanced by the event driven clock. They modified ModSAF to use a repeatable random number sequence as long as the same seed is used for each run. Variability is still available simply by changing the seed at the beginning of the run. They chose to avoid the potential variations from network events by simply not running in the network mode. These modifications have been incorporated into the ModSAF baseline with ModSAF 4.0 and beyond. The impact of this effort on the Concurrent Model will be discussed later in Section 3.5.1 and Section 4.3.2.

These initial tests of repeatable ModSAF were conducted by separate executions were ran for each scenario on the the same computer. The data was collected to confirm that identical location update, fire and damage events at exactly the same simulation time in successive runs. These were also compared to runs of the same scenario using standard ModSAF. While the standard ModSAF runs exhibited large variations, repeatable ModSAF reported exactly the same information at the same time. These results would provide a important starting point for our Concurrent Model approach experiments, while their warnings about problems in the networked environment invoked some concerns. Several of their changes to standard ModSAF were important to the CM approach. This included segregation of behavior and physical model update functions from the user interface and network functions. The behavior and physical model update functions were placed on the simulation-time queue and the network and physical model functions were left on the real-time queue.

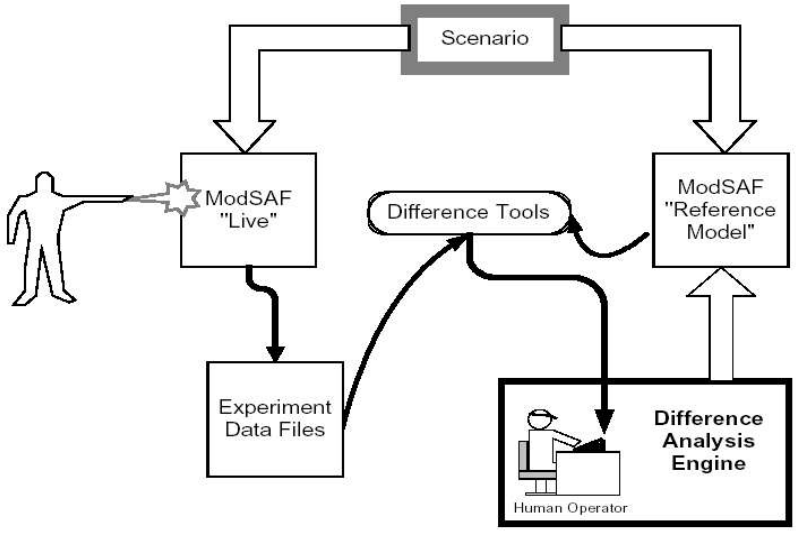



Figure 11. SPM Human Surrogate DAE

Ourston [Ourston 1998] reported on experiments to analyze the feasibility of a knowledge based Difference Analysis Engine (DAE). For this purpose they substituted a human controller as indicated in Figure 11 for the CM approach DAE. The function of the DAE is discussed in Section 4.1.2. In this role the Human operator performed the analysis and determined the corrections to the reference model to maintain congruence. Their purpose was to determine the following:

- Whether or not is was possible to maintain a reasonable coherence between the live vehicle and its reference model,
- What effect the complexity of the mission scenario had on the ability of control, and
- What effect the frequency of control updates (directly related to bandwidth) had on the accuracy of control.

They were able to identify the desired level for control of the SPM reference model and clones. The level of control that we selected was the behavioral control parameter level. This level provided a sufficient degree of control to maintain synchronization between the live and reference vehicles. They also found that simply allowing frequent reference model corrections did not necessarily result in better synchronization. They discovered that there needs to be a match between the accuracy of the reference model control and the frequency of update. Fewer but more accurate corrections provide improved synchronization when compared to more frequent but less accurate corrections. This is an encouraging result in terms of the CM approach network bandwidth reduction objective, as it implies that we may be able to reduce network bandwidth through the use of accurate corrections to the live vehicle clones.

## 2.4 Causality and Time Management

In this section, we identify the various factors that contribute to *congruence* of simulated models. Congruence deals with those constraints that need to be satisfied to en sure correctness in the resulting entity states generated by the remote generator. We also establish Discrete Event Simulation as a viable technique to implement the remote generators, as well as identify some of its potential limitations.

*Causality* is the concept that no event should appear to observers before the event that causes it. For example, as shown in Figure 12, given the three entities **A, B,** and **C** where entity **A** fires at **B** and sends a message indicating this event to both **B** and **C** at time $\mathbf{t_1}$.

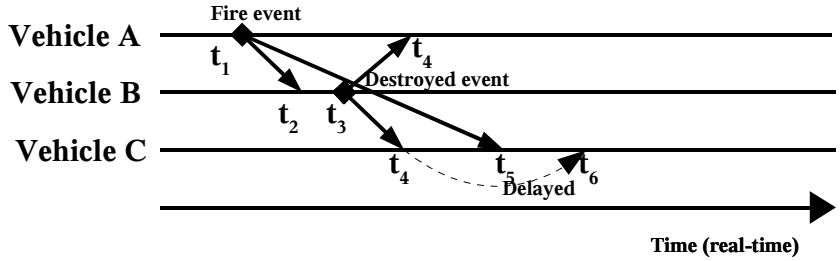



Figure 12. Scenario demonstrating causal event ordering [Fujimoto 1998].

The numbers indicate the time of the event, and the arrowheads indicate the arrival of the message at the other entities. The message arrives from **A** and **B** determines that the fire event caused it to be destroyed and creates the destroyed event, which it then transmits to **A** and **C**. **A** receives the message from **B** in the appropriate causal order, however, because in variances in the communications network, **C** receives the message about the destroyed event prior to receiving the message about the fire event. This last occurrence is an instance of causal misorder which causes ramifications as a simulation progresses.

The successor to DIS, called High Level Architecture (HLA) [HLA 1996], [Carothers 1997], [HLA 1998] provides various mechanisms to avoid causality problems. If causal event ordering was selected for the interface, the message from **B** to **C** would be delayed as indicated by the dashed line in Figure 12, until after the delivery of the fire event message.

Lamport [Lamport 1978] shows the the concept of "happening before" defines an invariant partial ordering of events in distributed systems that can be extended to somewhat arbitrary total order. He points out the problem with the "happened before relationship," when a system is unaware of all external relationships. One solution is to construct a system of strong clocks which satisfies the following conditions. Let $\rightarrow$ denote the happening before relationship for members of the set $\zeta$. For any events $a, b \in \zeta \; a \rightarrow b$ then $\Pi(a) < \Pi(b)$. Where *a* and *b* are discrete events and the function $\Pi$ *(x)* returns the timestamp for the event *x*. Fujimoto [Fujimoto 1996] explains that HLA provides four message ordering mechanisms in order of increased functionality they are *receive, priority, casual*, and *timestamp* order. Only the last two satisfy the relationship illustrated in Figure 12, and only the timestamp order satisfies Lamport's strong clock. One problem with the mechanisms provided by HLA is they don't adequately address real-time, as the real-time clock advances independent of any message traffic [Fujimota 1996].

Discrete event simulation is a technique where simulation time is advanced based on a logical clock rather than a physical clock. This simulation technique is one in which the

state of the modeled system is broken into a sequence of discrete, but possibly random set of simulated time points [Schriber 2001]. These time points are associated with events, and at each event the entities are updated to reflect their state at that event time. Since these are unique points in time, once all states are updated, the clock can be advanced to the next time that an entity has an event that would cause a change to its state. The data structure or queue that is used to keep track of the next time, is often referred to as the *event list*. The event list operation will be presented in Section 2.5.

Parallel implementations of Discrete Event Simulations [Fujimoto 1990] are available to speed up the execution of the simulation. However, causality and the need to en sure that all the data required to properly calculate the new state of each entity can counteract the gains of parallelism. As a result there has been much work on resolving these conflicts, they are usually classed as *conservative* such as the Chandy, Misra and Bryant approach, or *optimistic* such as the Time Warp Mechanism as studied in [Lin 1991A], or alternatives such as discussed in [Lin 1991B]. There is no clear consensus concerning whether optimistic or conservative synchronization perform better; indeed, the optimal approach usually depends on the application [Fujimoto 1999]. Time management in distributed simulation has two major challenges, that of time anomalies, and repeatability [Baker 1999]. Another problem that occurs with time management and discrete event simulation is management of *simultaneity* [Wieland 1999]. Wieland defines it as simultaneous events will be very narrowly defined to mean two or more events of identical type with identical time stamps, to be executed by the same logical process. This causes a question for causality in how to break the tie, or how to resolve the correct

solution when two entities use the others state to calculate their current state. Currently, OTBSAF assumes the input data to occur from a previous state and does not allow current calculations to propagate so all simultaneous events use only data from earlier timestamps. Repeatability, as pointed out in Section 2.3 remains as a primary consideration for causality and time management. To address these issues we propose to use strong clocks and synchronization. Furthermore we plan on using rigid First in First out (FIFO) ordering for simultaneous events, as processing time may move initially simultaneous chain of events to different clock tics, thus endangering repeatability, if ordering is allowed to vary.

## 2.5 Real-time, simulation-time, Scheduling, and Synchronization

In the Section 2.4 we introduced Discrete Event Simulation as a simulation technique that might be used to satisfy our need for a remote entity state generator. However, it is primarily used with a logical clock rather than real-time simulation. In this section, we will explore techniques to use this type of simulator in real-time and the additional congruence factors required for correct operation in this domain. We further subdivide congruence into two categories called *temporal congruence* and *behavioral congruence*. Temporal congruence addresses the factors that impact the timeliness of the simulation output, and behavioral congruence addresses the factors that impact the correct interpretation of the output. So they span both causality and simultaneity concerns.

Real-time processing defies having an unique definition [Wise 1971], [Mellichamp 1983], and [Laplante 1997], but relies on each author to develop a more precise definition

for a specific application area. Real-time processing does refer to systems which depend on several underlying principles. Time is a very important parameter in all simulation systems, its significance varies from synchronization to other processes that occur in nature, manufacturing, or communication; to being a key competitive component of products that depend on human interaction in terms of responsiveness. Real-time processing for this dissertation is defined as constraining presentation of simulated cues and signatures within a human discernible period consistent with physical reality. This is akin to a *soft real-time* definition whereby acceptable constraints can not be precisely quantified but only judged by their effects.

Three possible categories of real-time simulation, *live, virtual* and *constructive* each have different real-time constraints. Live simulation is training by simulation with the actual equipment or equipment surrogates, in a training area similar to the expected performance locale. In this case, the effects are simulated by various cues and signatures to create the sounds and visual effects. In this case, the simulation must generate these cues in the same time frame as the actual devices would in use. The second category, virtual simulation primarily refers to “manned simulators,” that is a training system modeled after the actual operational equipment as far as the man machine interface but not physically accomplishing the mission. In this case the response of the system to the man's manipulation of the system controls are simulated by changing “out the window” displays and other sensual stimuli. In this case all the stimuli are generated locally on the simulator, but when used in the collective environment (more than one simulator used to train a team working together) the actions initiated by one platform may also impact the

"scene" on another platform. Similar to the live simulation when activities on one platform impact additional platforms some method must be provided to initiate the generation of the appropriate changes on the remote platforms. The real-time constraints on scene generation depend on how fast the scene can be generated with the desired detail. This then drives the frequency of scene update, called the frame rate. Generally the faster this frame rate, the smoother and more realistic the simulation appears. The other constraint is related to the physical system being simulated as it dictates the amount of the scene that need to be updated. The physical constraints are those related to both linear and angular speed as they govern how much of the "out the window" scene must be changed for a given time quantum. The third category of simulation, constructive simulation only requires time synchronization between entities not real-time, as it not used for real-time operator interaction.

Simulation-time on the other hand, is a logical time maintained by clocks in a simulation, it can be faster or slower that real-time. simulation-time primarily serves two functions, one to establish the proper order to execute the events in a simulation, and the second is to provide a relationship to how long in real-time the execution of events would take. In most cases simulation-time is completely independent of real-time, however in some training environments it needs to be synchronized with real-time.

Scheduling in real-time is a much studied area, primarily addressing the need to have solutions available by a certain hard deadline. If the deadline is not met in these systems a catastrophic event may happen. As indicated in the following references, these studies

provide some useful insight to the problems addressed by this dissertation, but only indirectly. Ballerina [Balarin 1997] has proposed a schedule validation scheme for reactive embedded systems. If it shows a schedule to be valid, it is guaranteed, however the converse is not true, it may be labeled invalid but still be valid. While this may be useful during development of individual models it is not appropriate for dynamic application. Abbott [Abbott 1992] discusses performance evaluation of real-time database. This has some properties similar to requirements of a real-time discrete event simulation system in that it is transaction oriented, and for coherency purposes once a transaction is started it must be completed. Chou and Borriello [Chou 1994] present a scheduling algorithm for embedded reactive systems and introduce the idea of *safe exit points* to use *watchdog constraints.* These principles may apply to the *dynamic adjustment of computation rate* as introduced in Section 2.6, see Table 2. Sun [Sun 1997] provides a set of algorithms for bounding the completion time with arbitrary release times, the best algorithm although suitable for off-line analysis another algorithm is proposed for online control. This is another potential approach to the dynamic adjustment of computation rate. George and Minet [George 1997] show that FIFO ordering based on release times for real-time distributed systems has the advantage of preserving consistency. This is very important for causality. Lehoczky [Lehoczky 1996] proposes an alternate approach to real-time system scheduling which promises the potential of predictability for systems characterized by substantial stochastic behavior while providing a much higher level of utilization than worst case analysis. This may be beneficial in the sizing of concurrent generator if performance becomes critical. Sun and Liu [Sun 1996] propose and compare three synchronization protocols for distributed real-

time systems, these are compared for average and worst case end to end response time. This analysis assumes that the distributed processors are being allocated to satisfy a workload that can be modeled as a set of periodic tasks, each of which consists of a chain of sub-tasks executing on different processors. In general, the review of these various techniques confirmed the selection of the soft real-time approach.

Synchronization primarily applies to the clocks whether physical or logical, although it can be generalized to the coordination of any activities. By synchronizing the clocks among distributed objects the coordination of activities can then be referenced to coordination with the local clock. Hardware synchronization of clocks has become significantly easier in recent years, with both the advent of the Internet, and the Global Positioning System (GPS). Internet gives ready access to various standards organizations and established standards for maintaining computer clocks to within few milliseconds. GPS depends on knowing the time within a few microseconds and has very stable atomic clocks in each satellite, and a complex ground network that keeps them in long term synchronization. Since, the application area of this study is for embedded simulation it is assumed that all of the platforms will have a GPS unit for navigation purposes that could readily provide a synchronizing signal for the simulation clock. Under all synchronization strategies, some method to manage the list of events is required.

## 2.6 Queuing Strategies

Discrete event simulation promises a strong potential for repeatable results given that the events can be kept in timestamp order. In [Bahr 1994B] it was found that in some

situations the performance of the priority queue data structure can have a major impact on the overall simulation execution time. Since the concurrent model approach requires the simulation at many more locations than DIS, it is important that the scalability of the simulation remain practical. Thus, it is important to understand some of the potential limits of the simulation process.

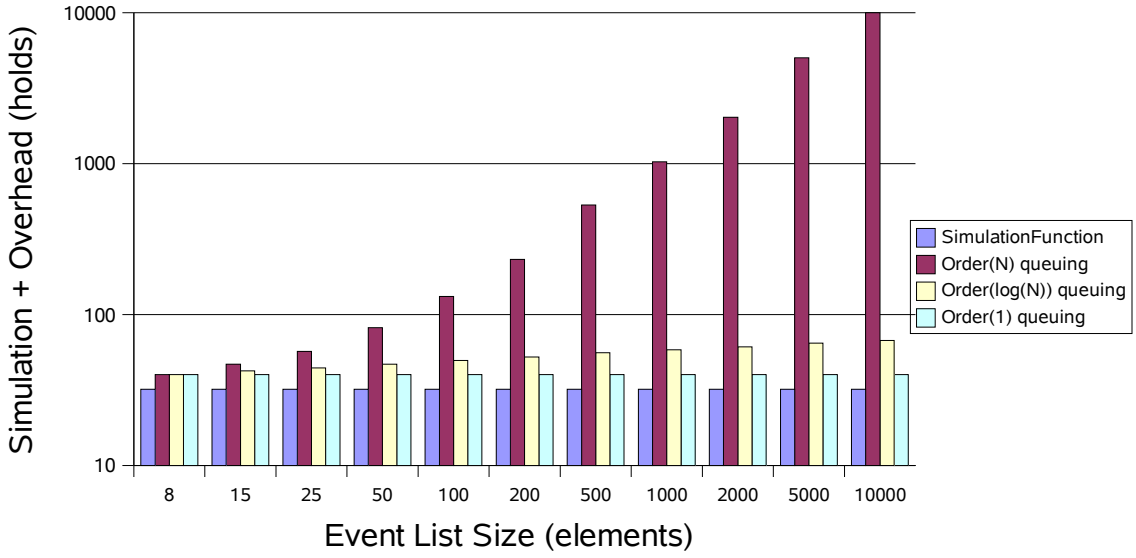



Figure 13. Increase in Computational Load Versus Events Queued

Figure 13 illustrates the type of impact that different priority queue structures can have on the overall efficiency of the simulation, especially if we increase the size of the simulation. Figure 13 shows the situation where the nominal size of the most frequently executed function of the simulation takes four times the time as the baseline priority queue hold, later in this section we will explain the hold and why the length of the hold varies depending on the architecture. We assumed that all queues had the same

performance for a queue size of 8 elements. To simulate these 8 elements it takes the equivalent of 32 baseline holds. The first bar of each size cluster shows this time. The other three bars in each cluster represent the total time to simulate 8 elements plus the overhead time of 8 holds. Thus, at this common point the three different classes of queues showed a total of 32+8 = 40 or an overhead of 8/40 or 20% for this queue size. The hold time for each element of priority queue can vary due to the number of elements in a queue. The amount of this variance is dependent on the architecture of the queue. As the number of elements in the queue increases the time of each of the 8 overhead holds increase, while the time to simulate the elements remains constant. Thus the overhead takes a greater proportion of the time. The three remaining bars in each grouping show the effects on performance depending on the class of the priority queue algorithms. The simplest to implement class exhibit O(N) performance, where ***N*** is number of simulation events in the queue. The next class is the O(Log(N)) which show reasonable performance for moderately sized queues, and the most complex to implement class of queues (O(1)) which exhibit near constant performance through some very large queue sizes, as will be shown in the detailed discussions to follow. Note that the O(N) queue drops to less than 50% efficiency above queue sizes of 25 elements and O(log(N)) above queue sizes of 5000, while Order(1) queues maintain a constant efficiency. Most ES applications involve instantaneous queue sizes of several thousand entries.

Table 1 presents the numerical values from Figure 13 that will be used in the example calculations. The following examples will illustrate the impact that this characteristic could have on ES. Table 2 provides a classification of entities based on distance or other

factors and an execution profile to keep the observer aware of the entities yet keep the amount of computation required within the capabilities of the target system.

Table 1. Queue Percent of Computation for Given Queue Sizes

| N | Order(N) | | Order(log(N)) | | Order(1) | |
|---|---|---|---|---|---|---|
| | Overhead | % of Base Value | Overhead | % of Base Value | Overhead | % of Base Value |
| 8 | 8 | 100% | 8 | 100% | 8 | 100% |
| 15 | 15 | 118% | 10.42 | 106% | 8 | 100% |
| 25 | 25 | 143% | 12.38 | 111% | 8 | 100% |
| 50 | 50 | 205% | 15.05 | 118% | 8 | 100% |
| 100 | 100 | 330% | 17.72 | 124% | 8 | 100% |
| 200 | 200 | 580% | 20.38 | 131% | 8 | 100% |
| 500 | 500 | 1330% | 23.91 | 140% | 8 | 100% |
| 1000 | 1000 | 2580% | 26.58 | 146% | 8 | 100% |
| 2000 | 2000 | 5080% | 29.24 | 153% | 8 | 100% |
| 5000 | 5000 | 12580% | 32.77 | 162% | 8 | 100% |
| 10000 | 10000 | 25080% | 35.43 | 169% | 8 | 100% |

This classification is based on an assumption that the observer is generally more likely to be impacted by closer entities than distant entities. While each entity in general would control a fixed area, the area available for occupation increases as a square of the distance from the observer. In addition if each entity can move at the same velocity, the time it takes to reach the observer is proportional to the distance. Thus, there is the potential for more entities to exist at greater distances, but since it will take them longer to impact the observer, the observer does not need to have the distant entities information updated as often. There are actually many factors that could be included in a prioritization scheme, but this example is used to illustrate a method of allocating the computation resource

based on a priority. This approach would allow the monitoring of many more entities than a constant priority.

Table 2. Prioritized Execution

| | Immediate | Near | Distant | Outside | Other | Total |
|---|---|---|---|---|---|---|
| Number of Entities | 50 | 100 | 250 | 600 | 4000 | 5000 |
| Frequency of Comp | 15 | 3 | 1 | 0.25 | .0125 | |
| Computations/sec. | 750 | 300 | 250 | 150 | 50 | 1500 |

Every scheduled operation on each entity has the overhead of a priority queue hold. If we take the simplest case of Table 2 of only computing the entity state for the *Immediate* category of entities and keeping the assumption that computation time for each entity was 4 times the baseline hold, we can calculate the equivalent computations per second for each queue. Let us label the different queues from left to right of Table 1 $Q_{O(N)}$, $Q_{O(logN)}$, and $Q_{O(1)}$. The number of equivalent operations for $Q_{O(N)} = 2.05 * 750 = 1537$, $Q_{O(logN)} = 1.176 * 750 = 882$, $Q_{O(1)} = 750$. By just using a more effective queuing strategy, it is possible to allow the use of a higher resolution scenario with all the entities of $Q_{O(1)} = 1500$, or a lower resolution scenario including through the distant category with $Q_{O(logN)} = 1.35 * 1300 = 1755$ at the same computational load of the simplest queue structure. Note that to improve the computational performance a simulation, it is imperative to reduce the computation time of the most frequently encountered operations. Thus, the basic operation of the event queue gains significant importance, because it represents a significant portion of the inner loop of any simulation. The remainder of this section will

concentrate on lineage of the priority queues that lead to the one developed in this dissertation.

As previously suggested by Gonnet [Gonnet 1976], some specific distributions of non-uniform events can occur during a simulation that can impact the performance of the event list storage strategy. Likewise, Brown observed that queue statistics should be continually monitored to determine which storage structure will minimize overhead for the particular distribution of events encountered [Brown 1988]. Although the management of time-flow in the simulation can be handled either synchronously or asynchronously [Emshoff 1970], the asynchronous scheduling is often preferred [Evans 1967]. In this case, a prioritized list of future events must be maintained so that individual tasks can be scheduled for execution at the appropriate time. Thus efficient implementations of the priority queue can be key to an efficient mechanism for maintaining causality and repeatability.

A standard metric for comparison of the relative performance of an event list management algorithm is the time required for a HOLD operation [Vaucher 1975]. A HOLD operation consists of first retrieving the event at the head of the scheduling queue, then adding the delay for the new event to the current events time, and inserting the new event into the appropriately prioritized location. The HOLD operation is representative of the total of the queueing operation for each event. Measuring the time spent to execute HOLD operations provides a measure of event list overhead during simulation. As typically defined, a HOLD operation consists of the combined time for insertion and

removal processes for a pending event. Another fundamental task used to assess performance is the DELETE operation that removes a superseded event which is no longer pending. DELETE operations can cause more severe performance degradation for some priority queue strategies such as heaps.

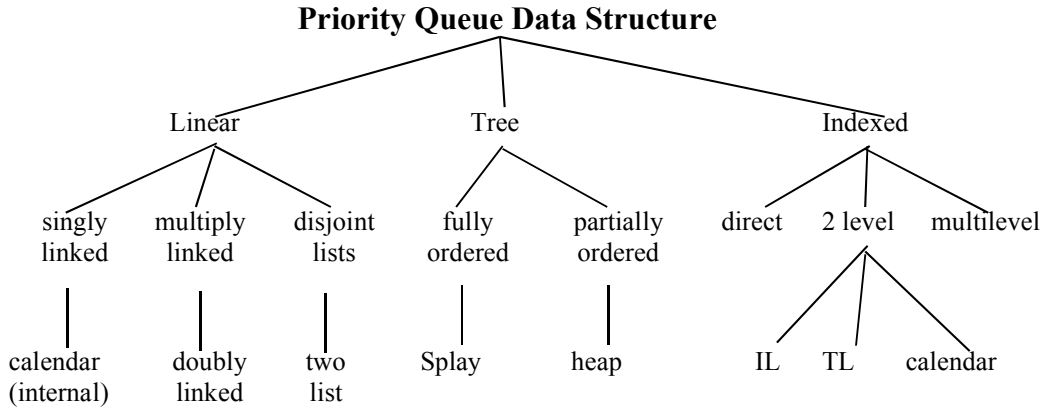



Figure 14. Priority Queue Taxonomy

In a priority queue, the ordered structure of the elements must be continuously maintained according to their relative time for scheduled execution during the simulation. For elements of equal priority, an accepted practice is to store them in First-In First-Out order. Figure 14 illustrates the various structures that have been explored for implementing priority queues, including several linear, tree, and indexed data structures. We will focus the discussion on linear and indexed structures as they form the basis for the SPQ techniques.

### 2.6.1 Linear List Priority Queues

Linear lists can be divided into subcategories of singly, multiply, and disjoint lists [Jones 1986]. The linear queue is formed by maintaining pointers to the head and tail of a singly-linked list. During an enqueuing operation, the events are stored in their prioritized location by repeatedly comparing the simulation time of the new event to the previously stored values. A new entry has its priority compared first with the priority of the head element and if it is less than the head then it is immediately inserted as the new head as shown in Figure 3. Otherwise, it is compared to the tail. If it is greater than or equal to the tail then it is appended as the new tail of the list. If it is neither a new head nor a tail, the priority is compared in succession with each next item in the list until it is less than the succeeding entry. Upon locating the correct point, the new entry is inserted in the list by updating the corresponding pointers. The primary advantage of this queue structure is that head of the list is always available without searching, thereby simplifying the removal portion of the hold operation. Insertion and removal operations generally require two pointer updates. The disadvantage is that successive entries must be examined until the correct entry is found for both en queue and arbitrary element delete operations.

### 2.6.2 Indexed Lists

Indexed lists attempt to mitigate the drawbacks of linear queues by reducing the amount of searching required to locate or insert an item. Voucher and Duval [Vaucher 1975] introduced a time mapping algorithm called the indexed list algorithm. This algorithm

consisted of a linear list for event storage and an array of pointers to a set of dummy markers inserted into the list with the last pointing to the overflow area. The markers are a fixed-time interval apart and as the current simulation time passes them they are moved forward into the overflow area and inserted into the list at the appropriate time-ordered point. In this implementation, time is represented by an integer variable, and no finer subdivisions are allowed. The time at which an event is scheduled can be used as an index to select from the list into which the notice must be placed. With a FIFO priority system, the new notice is placed at the end of the selected list, and no scan is necessary. Although this algorithm in its basic state must be generalized in order to be widely applicable, the basic idea of grouping is useful to reduce the scan-time. However, the implementation did not include a feedback mechanism for adjusting the spacing between markers. The last pointer in the array delimits the overflow portion of the list.

### 2.6.3 The Calendar Queue

Brown [Brown 1988] introduced the Calendar Queue, which derives its name from its structural similarities to a desk calendar. The basic concept is that two arrays hold pointers to the head and tail elements of singly-linked lists of events. Each element of the list stores the priority of the event and a pointer to the next element. As shown in Figure 15, the length of the array is equivalent to the number of days in a calendar year. The figure depicts the queue with additional elements to illustrate storage in days 0 through 7. The index of each array is equivalent to the count of number of days since the beginning of the year, i.e. entry 0 is the start of the year and entry $n-1$ is the last cell in the year where $n$ is the number of days in the year. Overflow of events on any day is taken care

of by placing the elements outside the current year, called *outyear* elements, in the appropriate day of the calendar in time sequence. The index is calculated as *(priority/day_size) modulo year_size*, and converting to an integer value. If year sizes are kept as power of two, the modulo operation becomes a binary AND operation with a mask value of $n-1$ where $n$ is the year size. Interior to each day, the individual events are kept in priority order by scanning the list and inserting the elements in the appropriate location.

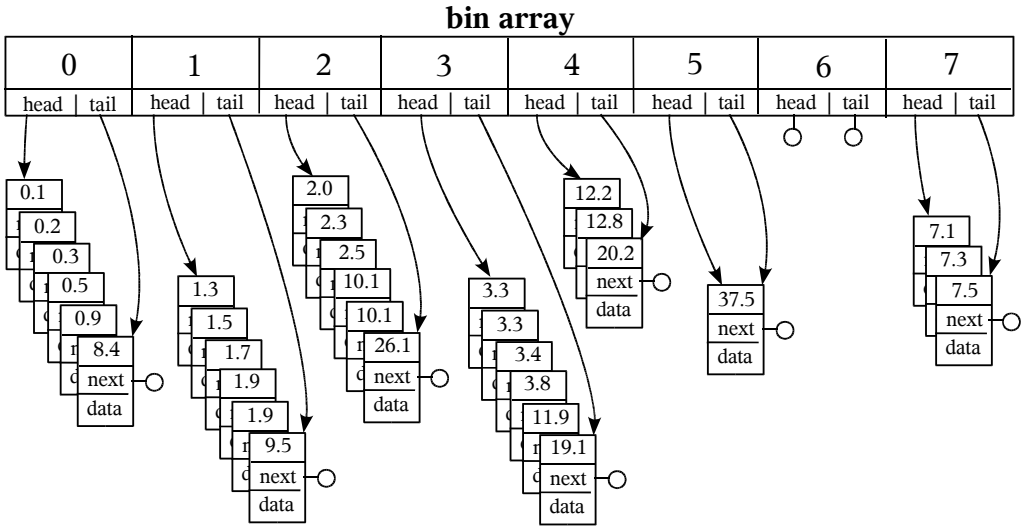


Figure 15. Calendar Queue

The key mechanism for calendar queue's improved performance over the linear queue is that it reduces the average sequential search length to half number of elements in a bin as compared to half the number of elements in the total list. However, there is a price to pay for this capability, and it consists of several factors. The first is the basic bin selection process or indexing, this requires conceptually a floating point multiply, a floating point

to integer conversion and then an integer modulo operation in addition to the compare and step, whereas, the linear search requires only comparison and advancing pointers. Another is the queue resizing cost, which requires sampling data in the queue, to calculate distance between events and moving the data to the appropriately spaced bins. The third is keeping track of the head element of the bin. In the linear queue, the head is always first in the list. In the calendar queue, there are as many list heads as there are bins. In a model with smooth evenly spaced events, the next head is always near to the current bin. In a worst case, it can lead to searching for the earliest event over the entire set of bins. Since the Calendar Queue has become popular several modifications have been proposed to improve its performance with specific event insertion distributions by Ahn [Ahn 1999], Bahr [Bahr 1994B], Ronngren [Ronngren 1993],and Tan [Tan 2000].

### 2.6.4 Relative Performance

Table 3 presents the results of a previous study by Jones which compared average and worst-case performance of 11 different algorithms this showed performance advantages for splay trees over a wide range of conditions and linked lists (linear queue) for short lists [Jones 1986]. Brown compared his calendar queue experimentally to the linear queue, and a queue implemented with a splay tree.

Table 3. Results of Jones Study

| Priority-queue implementation | Code size[a] | Performance | | Relative speed[b] | Comments |
|---|---|---|---|---|---|
| | | Average | Worst | | |
| Linked List | 47 | $O(n)$ | $O(n)$ | 11 | Best for $n < 10$ |
| Implicit heap | 72 | $O(\log n)$ | $O(\log n)$ | 8 | |
| Leftist tree | 79 | $O(\log n)$ | $O(\log n)$ | 9-10 | |
| Two list | 104 | $O(n^{0.5})$ | $O(n)$ | 9-10 | Good for n < 200 |
| Henriksen's | 68 | $O(n^{0.5})$ | $O(n^{0.5})$[c] | 1-7 | Stable |
| Binomial queue | 188 | $O(\log n)$ | $O(\log n)$ | 1-7 | |
| Pagoda | 110 | $O(\log n)$ | $O(n)$ | 4-8 | Delete in $O(\log n)$ |
| Skew heap, top down | 56 | $O(\log n)$ | $O(\log n)$[c] | 5-7 | |
| Skew heap, bottom up | 103 | $O(\log n)$ | $O(\log n)$[c] | 4-6 | Delete in $O(\log n)$ |
| Splay tree | 119 | $O(\log n)$ | $O(\log n)$[c] | 1-3 | Stable |
| Pairing heap | 84 | $O(\log n)$ | $O(\log n)$[c] | 3-6 | Promote in $O(1)$ |

[a] The total lines of Pascal code for initqueue, emptyqueue, enqueue, and dequeue.

[b] 1 is fastest; 11 is slowest:

[c] An amortized bound; single operations may take $O(n)$ time.

The calendar queue consistently outperformed the splay tree for the scenarios tested. In fact, the calendar queue exhibited nearly constant-time performance for many queue sizes, while splay tree execution time increased O(log n) or in worst case linearly with queue size. Ronnegren [Ronngren 1997] provided a comparative study which expanded Jones' study by introducing additional queue implementations and additional event set insertion distributions to highlight the differences of commonly used priority queues. Figures 16, and 17 provides the results for the Calendar Queue which forms the basis of comparison in Chapter 7.

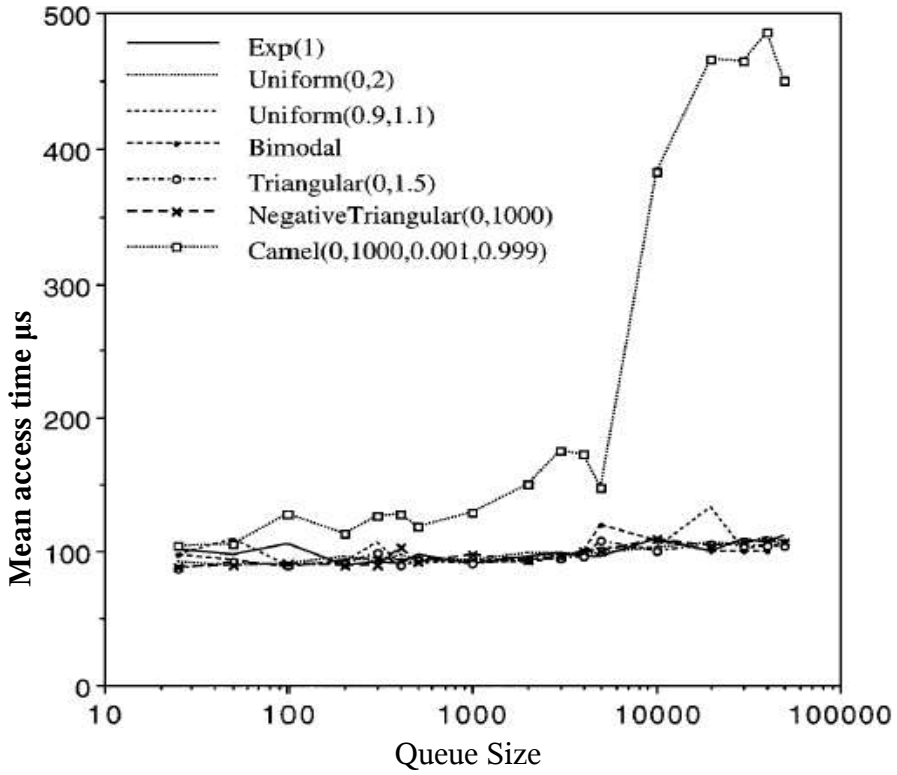



Figure 16. Mean access time for Calendar Queue and Classic Hold Experiments

In many simulation applications, event activity is not uniformly distributed throughout the range of simulation time. Frequently, the majority of activity occurs over a small interval at any time, for example near the beginning of the queue. This characteristic of the event scheduling distribution can be used to optimize queue storage structure. For example, consider a case study [Bahr 1994A] involving simulation of a distributed communications system using YACSIM [Jump 1993]. In this case study, the Range Data Measurement System (RDMS) for the U.S. Army was simulated where asynchronous communication between 2,000 source entities takes place over shared data channels to a central processing site. Several processors received the data in parallel and prepared it

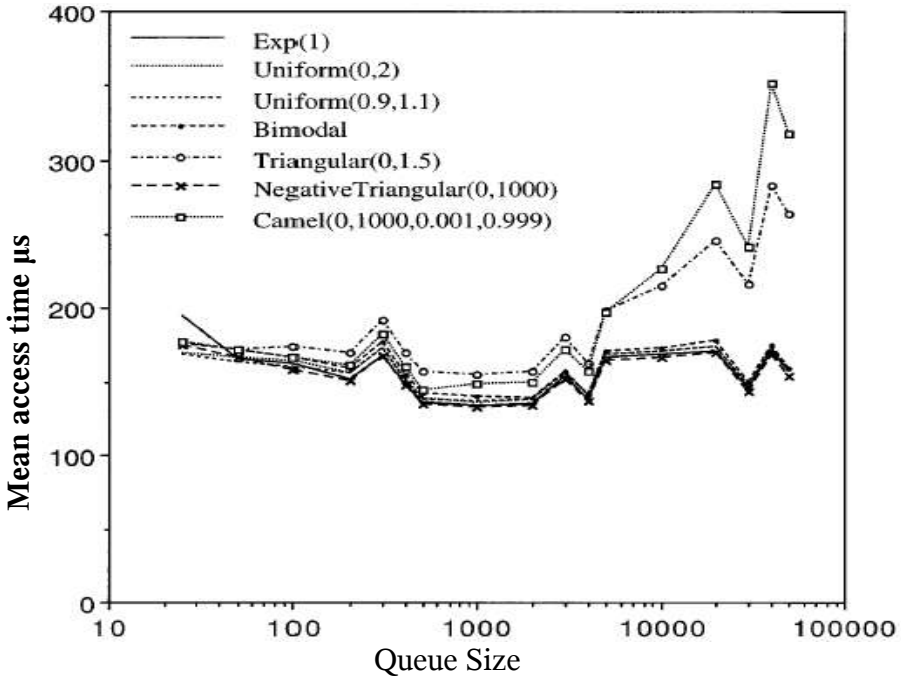



Figure 17. Mean access time for Calendar Queue Up/Down experiments

for retransmission in variously formatted data streams to other sites. To gain insight into placement and removal of activities on the event queue, several scenarios were simulated and statistics were gathered [Bahr 1994B].

The choice of data structure used to maintain the event list can significantly impact the efficiency of a discrete event simulation. In the case study of RDMS, the calendar queue exhibited a cyclic nature and degradation under high head-end activity. A representative metric for the overhead required to support the distribution of event list activity is the number of data comparison operations for each event insertion. There were approximately 7.5 million insertions for one simulation run. Figure 18, shows the distribution of the baseline linear queue which had 4.5 million insertions at the head of

the queue. The maximum search length was 2,301 and the majority of the events required more than 10 comparisons, however that the majority of events occurred near the head.

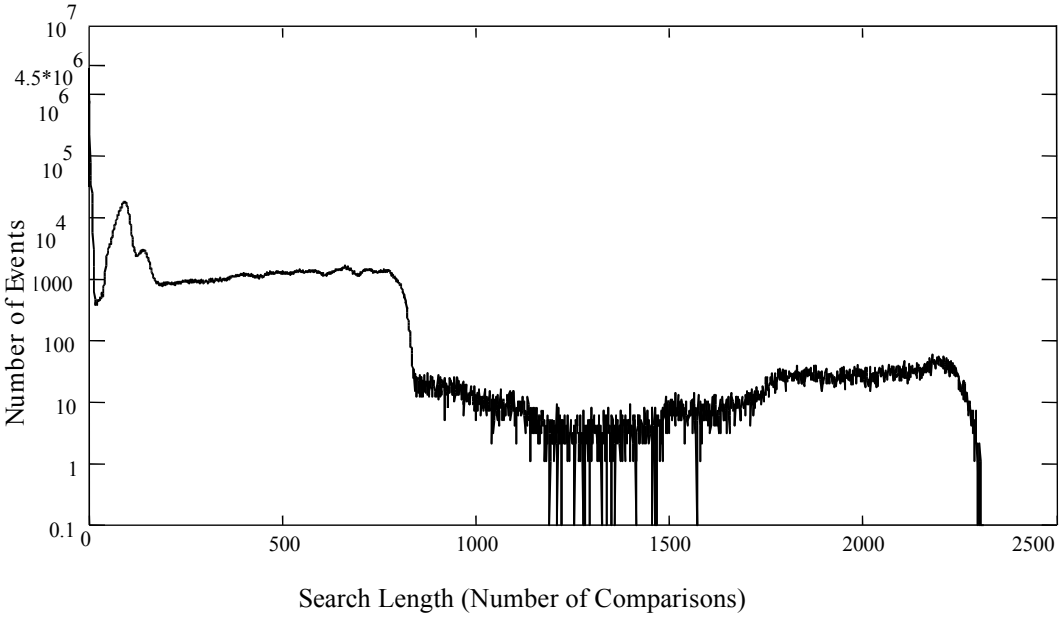



Figure 18. Distribution of search Length in DDC using a Linear Queue

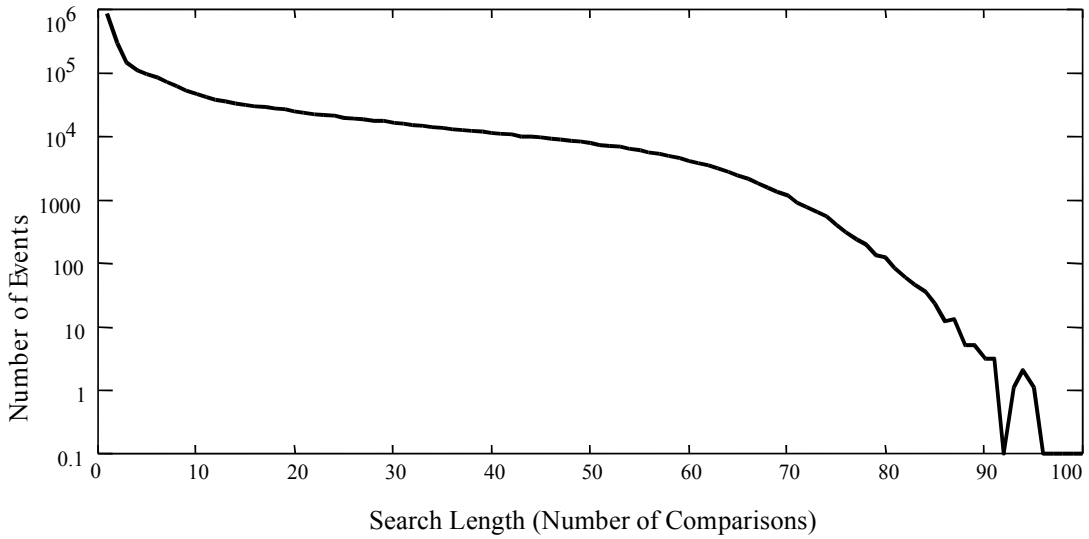



Figure 19. Search Length Distribution in a Calendar Queue

Figure 19 shows the improvement offered by the baseline calendar queue, with a maximum search length of 95 comparisons. Thus, calendar queue effectively limits the

maximum amount of time for any given operation, however the majority of the events are close to the head. The above data confirms that the majority of event list activity occurs near the head end of the queue. In [Bahr 1994B] analysis and experimental measurements demonstrate scenarios with a high degree of head-end activity can occur after the queue size has reached steady state. Since a linear list has lower traversal overhead for near events, a simple linear queue can outperform the calendar queue in simulation with a heavy head-end activity In addition Tan [Tan 2000] and Ahn [Ahn 1999] also proposed different adjustments to the calendar removing some additional weaknesses, however none of these three alternatives addressed a low overhead cost, highly stable solution for a large range of event insertion and deletion distributions.

# CHAPTER 3 TECHNICAL PROBLEM DESCRIPTION

The primary challenge facing Embedded Simulation is same as that of Distributed Interactive Simulation and situational awareness. That is maintaining a coherent view of the virtual environment between all participants. By taking a data-centric view then the entire environment corresponds to an ensemble of data streams. We can further divide this it sub-streams based on the volatility of the data. The most dynamic of these sub-streams is typically the entity state data, so we will refer to this as an *Entity State Stream.* This decomposition will allow us address the coherence problem as one of determining which data to transfer from its storage location to the viewing location within the constraints of the operational systems at hand. In this dissertation, we propose a novel approach to allowing this data stream to also be distributed and/or duplicated when needed to optimize the trade-offs between computation resources and communication bandwidth.

## 3.1 Background

Embedded simulation has been shown to be both cost effective [McDonald 1998A] [McDonald 1998B] [McDonald 2000] as well providing the training where and when needed [McDonald 1998A]. While much progress has been made in providing embedded training in a static (vehicles parked and connected by wires) mode, it still hasn't reached

the augmentative capabilities envisions by Bahr and Abate [Bahr 1997A] [Bahr 1997C] [Bahr 1998] [Abate 1999]. These features all entailed un-tethered or dynamic operation, along with the need to maintain realism and synchronization between distributed simulators.

In addition, operational systems keep getting more sophisticated, with longer ranges, and that extend beyond the line of sight. These new capabilities not only add to the training burden, but also add to the required sophistication to represent those capabilities in the virtual world and thus in the embedded simulation. Appendix 1 discusses the envisioned technologies required to implement the "total unit training" and Appendix 2 reports on the current status of these technologies.

## 3.2 Operational Constraints

The performance constraints on an Embedded Simulation system are generally the same as those present in a similar dedicated DIS system, but intensified by the operating environment of the embedded host. Thus they have similar requirements for communications and responsiveness, but with the mobility requirements including moving vehicles operating in a severe environment. While the static systems use wired LAN and ATM, technology the mobile training system has been limited to dedicated instrumentation systems as described in [Bahr 1994A] and [Goblick 1996]. Three representative dynamic application domains of embedded simulation are f*orce-on-force training*, *en route mission rehearsal*, and *situation awareness*. They all have the constraint of wireless communications and an ideal solution would be compatible with

each. The demand for bandwidth in wireless communications systems in ES systems is greater than the current and future anticipated capability. In cases when compression techniques can be used to reduce the quantity of the data the choice is frequently made to increase the amount of data provided rather than decrease the bandwidth [CBO 2003].

### 3.2.1 Force on Force Training Simulation.

One of the primary difficulties in providing the interaction between Live and Virtual objects is conveying the large amount of data between the respective entities for real-time interaction [Goblick 96]. These exercises are primarily conducted at fixed installations so an additional physical infrastructure could be provided to supplement the wireless system. The largest current installation covers an area of 2,400 square Kilometers. The interaction is between maneuvering ES equipped vehicles as well as being supplemented by virtual entities. Actual environment, sight, sound, and weather further impact the latency and bandwidth constraints in ES beyond baseline values.

### 3.2.2 En route Mission Rehearsal

This application is similar to the static training, however, it is conducted in or near the ES equipped vehicles as they are being transported over long distances. The opposing force in the ES would normally be played by Semi Automated Force operators at a distant location so the communication occurs between the home base and the various transport craft. Here the interaction is entirely virtual, making it the primary candidate for initial study described in this dissertation.

### 3.2.3 Situational Awareness

The primary goal in this application is to keep the operators of the vehicles aware of circumstances that could impact the successful completion of their mission, and for them to keep their chain of command aware of their situation. Again a key constraint is communications. However, at the same time as they and their commanders are sharing information, they want their opponents unaware. These two goals can be conflicting, and tend to limit the bandwidth at their disposal, degrading the communication capacity. A second limitation is, even if the information is available, it may not be in a format that is readily assimilated by the crew, and may not be recognized as pertinent to their current situation, or that it will be pertinent in the future. This imposes the most restrictive conditions with both live and virtual interaction. This application provides a high potential payoff in the enhancement of mission performance.

## 3.3 Computational and Communication Resource Tradeoffs

Initial experiments involving live simulation and virtual simulation domains, was basically seen in requirements documents for Embedded Simulation systems where update rates for ground vehicles in live simulation were on the order of once every 5 seconds, while the software simulators for the virtual domain they were on the order of 50 times per second. The specifications for the live simulations were driven by the restrictions of available technologies within the available radio frequency spectrum allocations. The virtual simulation requirements were driven by the need to present information at realistic frame rates within the less restrictive communications domain of

local area networks. Thus, these two domains specified a difference in communications requirements on the order of 100-fold. Latter studies have reiterated this ratio. Historically demand has soared to exceed capacity so even with the improvements in communications technologies this 100-fold ratio has remained and is therefore interpreted as a bandwidth objective . Let ***N*** denote the number of bytes required to transmit entity state information for a typical DIS scenario, and ***I*** denote the number of bytes required to transmit the entity state information for an equivalent live scenario. The ratio $B_R = N/I$ which indicates the increase in traffic under simulation environment can exceed $B_R > 100$ [Goblick 1996] [Valle 1997] [Bahr 1994A]. Using $B_R=100$ as a requirement for ES bandwidth ratio then a goal for the amount of data transferred under the Concurrent Model Approach can be set to ***N/100***, rather than ***N.***

A second parameter that needs to be constrained is the event communications latency. This is the time between the occurrence of an event and the report of that event to the observers. For embedded simulation, we generally classify these events as visual and aural events, that is where the real event would be transmitted at near speed of light, or those that are transmitted near speed of sound. These can be relaxed as long as nominal speed of a human reaction, and the nominal speed of a human logical decision process are taken into account. For example at a range of 1 kilometer a visual event would be seen in about 3 microseconds, while the sound of the firing event would not heard for 3 seconds. On the other hand the arrival of a fired projectile could vary somewhere between 1-5 seconds depending on type. An observer of the muzzle flash could react within about 0.5 seconds and possibly evade the fired projectile, whereas those sensing the sound of the

firing could only prepare for a future event as the longer thought process required to identify the location of the firing and its potential path as well as the much shorter interval left between the arrival of the projectile and the sound of its firing would prevent evasion of the current round. In the virtual world, all transmission of events occurs electronically. In this case, the latency depends on the processing of the required code to create and sense the event, the transmission speed and capacity of the links between the sender of the event and the observer, as well as the queuing status or wait time for the event to be transmitted over the link. We will analyze these elements in detail in the following sections.

## 3.4 Situation-related Communication

An immense amount of information is needed to portray the current situation to the participants of either a training situation or operational situation. In the live situation, most of the information is available to the observer through his five senses. It can also be enhanced with electronic devices such as radar, thermal viewers, and chemical detectors. To provide this same information in the virtual world we must create and communicate stimuli for all sensors. Table 4 provides a description of various sub-databases that might be used to organize the information required to completely convey the current state of the environment. Column 1 provides the classes of data, column 2 indicates the relative frequency of updates, and column 3 provides a description of the data elements and how they are used. The three classifications used in column 2 are *S* for static, *M* for may be modified during an exercise, and *H* for highly dynamic information that changes on an

individual participant basis. We will refer to this highly dynamic data as *entity state data*, and it is currently conveyed as an entity state stream.

Table 4. Situation Database

| ***Data Set*** | **Update Type** | **Description** | |
|---|---|---|---|
| *Terrain-database* | **S** | A detailed terrain and features data-base that allows models to exercise the procedures appropriate to the environment. Ground vehicles for instance are blocked by impassable areas, may be masked by terrain features or dust. Can sink in dry lakes, etc. This data-base replaces the human observation of the terrain. | |
| *Threat-platform models, sub-models* | **M** | The set of adaptive constructive models that are clones of the reference models on the respective simulators/weapons platforms. Parameters for these models can vary from identifiers of functions, numerical values, to identifiers of pre-tabulated characteristics. | |
| *Pre-defined orders* | **S** | Standing orders, or orders issued prior to the start of the exercise. | |
| *Vulnerability data* | **S** | Susceptibility of the local platform to the various threats, as well as the susceptibility of the threat to the local platform. | |
| *Weather-data* | **M** | Any weather related information that can impact the results of the exercise. | |
| *Mine-field, obstacle data* | **M** | Contains locations and types of mines and other obstacles. Includes visibility data. | |
| *Current state data* | **H** | Status of combat team members: | Location of team members and their combat status. |
| | **M** | Current intelligence information: | Contains information about foe above and beyond what sensors can provide. |
| | **H** | Status of each threat platform: | Current status of each of the targets within field of view. |
| | **M** | Current orders: | The set of orders that govern platform's objectives and techniques for achieving those objectives. |
| *Predetermined parameters of all potential inter-actors* | **S** | A data-base of all players identified as potential participants in the exercise. This allows the initialization of the clones based on an identifier rather than by detailed transferred parameters. | |
| *Learned reaction of local operator and object against each inter-actor* | **M** | A historical data-base used by the DAE subsystem to initialize the reference model. | |

The concurrent model approach proposed in this dissertation modifies the DIS approach of broadcasting a single entity state stream between all hosts to generating a congruent stream at several hosts as indicated in Figure 20.

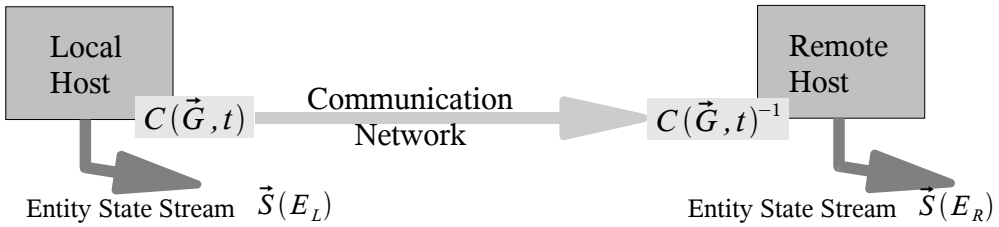



Figure 20. Congruence Transfer Function

In this Figure, we show two hosts each generating an entity state stream. The local host is also generating a set of congruence messages $C(\vec{G},t)$ that explicitly state the setup of the generator and the time that this configuration becomes effective. The remote host then schedules the setup of the remote generator to coincide with the local generator at the same time as the time. ***C*** is the congruence function for each entity for which the state is to be generated. It is dependent on $\vec{G}$ which is the vector of all model parameters that control a given entity, and ***t*** the time those parameters will start applying. These values are of Category M in the situation database. Congruence functions also include all messages being conveyed by the operators of the exercise. If they are not scheduled by the operator the local host will assign a time of execution $t_e = t_c + \delta$ based on the current time $t_c$ plus a $\delta$ time based on an estimate of the worst case latency between the local host and the remote host to en sure coincidence of the application of the messages.

In general then the concurrent model approach depends on all three categories of the situation database. Referring to Table 4 the static **S** dataset that is preloaded along with the simulation prior to the start of the exercise. The **M** dataset that is transferred as Congruence functions, and the **H** stream that is generated at each host. The **S** data provides the basic information that provides the foundation to the rest of the information. The **M** dataset provides the rules for generating the **H** stream, which in turn is used by the DIS applications to provide the views for the observer. In subsequent sections, it will be shown how this can be realized in OTBSAF to dvelop the concurrent model approach. This will then be used to quantify the impact of each of these categories on the communications transmission by using the current OTBSAF / DIS packet sizes and frequencies as the baseline for comparison.

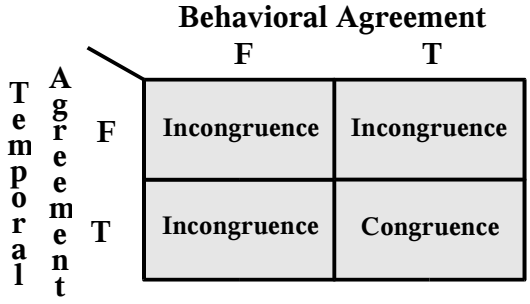



Figure 21. Congruence Karnaugh Map

Congruence depends on both temporal and behavioral factors being maintained. To illustrate the factors, if all the factors are collapsed into two Boolean variables, one indicating Temporal Congruence and one indicating Behavioral Congruence. Expressing these in a Karnaugh map as shown in Figure 21, showing an **"and"** relationship between

these two sets of factors. If neither Temporal or Behavioral agreement is maintained then congruence is not expected to hold true, but if the reactions are correct then why is it not necessary for temporal agreement to hold true. If reactions occur out of order they are no longer reactions per se. Thus, timeliness is necessary as well. Likewise even if reactions occur at the right time if they are not correct then congruence fails to hold true. Thus congruence, doesn't hold true except when both Temporal and Behavioral factors are true.

The function for determining Temporal Congruence is denoted $\Psi_T(t, E_R)$ and the function for Behavioral Congruence $\Psi_B(E_L, E_R)$. The Temporal Congruence function $\Psi_T(t, E_R)$ is dependent on the timing $t$, and the output of the remote generator $E_R$, while the Behavioral Congruence function $\Psi_B(E_L, E_R)$ is dependent on the relationship between the output of the local generator $E_L$ and remote generator $E_R$. The truth function for each of these relationships $\Gamma_T$ and $\Gamma_B$ are dependent on these functions remaining within limits. So $\Gamma_T$ is true if $\Psi_{TO}-\delta \leq \Psi_T \leq \Psi_{TO}+\delta$ otherwise it is false, and likewise $\Gamma_B$ is true if $\Psi_{BO}-\delta \leq \Psi_B \leq \Psi_{BO}+\delta$ otherwise it is false. The subscript $O$ indicates the desired value.

Figure 22 provides some examples of the types of graphical displays that might be used to depict remote environments. The two labeled VSAM processing, which stands for Video Surveillance and Monitoring, are actual sensor and camera views which would lie at the base of the communications pyramid shown in Figure 9. The one labeled ModSAF is a plan view or map based display with the entities displayed as stick figures currently these are transmitted by the Dead-reckoning block of the pyramid.

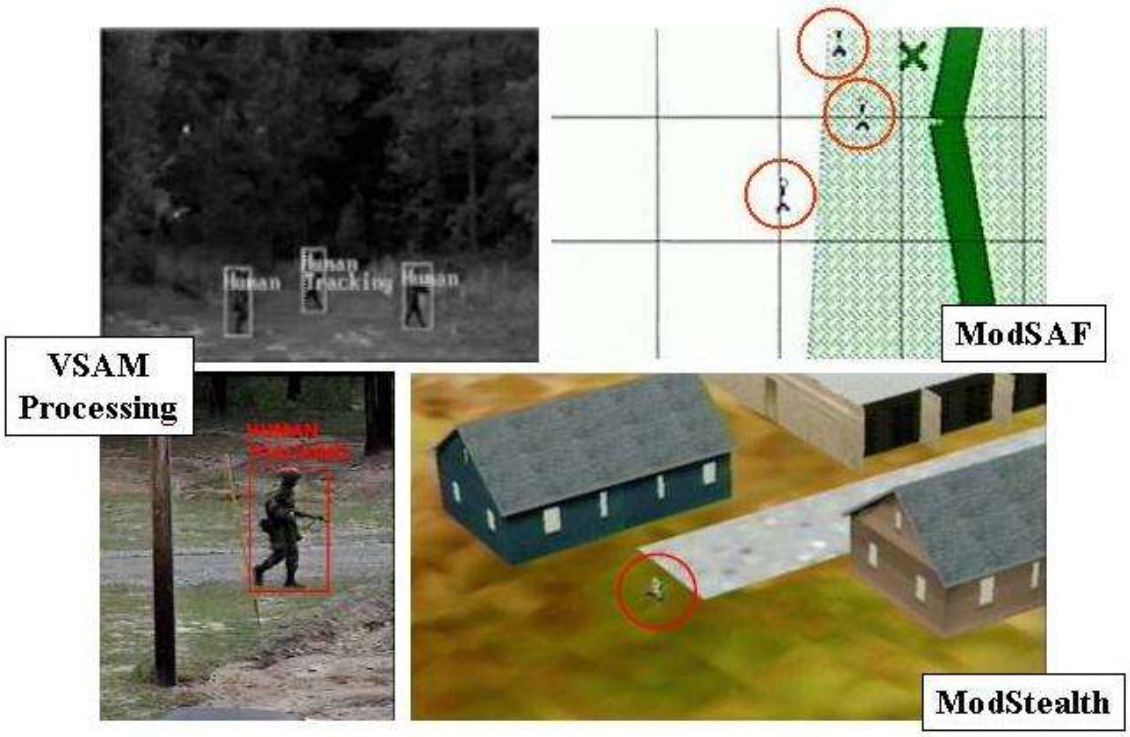



Figure 22. Example Graphics Illustrating Tolerances

The view labeled ModStealth is a three dimensional representation of the view from an arbitrary viewpoint [Collins 2000].

As as example of computing $C(\vec{G},t)^{-1}$, applications already exist to render the ModSAF and ModStealth views based on terrain databases and visual models of the objects and an entity state stream based on the DIS protocol. As can be seen from these examples, though the video or sensor views can provide increased detail, the value of this detail is somewhat questionable. In most cases the depiction of individual blades of grass are irrelevant to the viewer. Thus, the generated views provide the necessary information at a much lower bandwidth. This motivates the selection of the $\boldsymbol{\delta}$ tolerance parameter.

To take advantage of the current rendering technology, the entity state data must be present as a DIS protocol stream at the observer. All of this is present locally for standalone embedded training, and available on a LAN for static embedded training. Category S data can be provided in advance of the exercise, Category M data is not currently the limiting factor in latency and bandwidth considerations for the communications link. So if category H data could be generated locally, this would make the communications of category M data the requirements driver for the WAN. Given that it has already been found practical to include the simulation components for embedded training on each remote platform, this dissertation focuses on the implications of trying to synchronize multiple simulations in order to present a consistent view of the simulation-space to all observers.

## 3.5 OTBSAF as a Prototyping Testbed

OTBSAF is a large scale constructive simulation system developed to portray elements down to the individual platform or entity level. Although a constructive simulation, it provides both logical and real-time clocks so it can be used for real-time interactive simulation to portray additional elements in an exercise beyond those represented by manned simulators. For communicating with the manned simulators, it uses the DIS protocol. Each entity is simulated by instantiating the appropriate model for that entity. Initially, each entity assumes the default values for each parameter that can be modified by the SAF operator. Below we will describe OTBSAF distribution, and then explain its application as a testbed.

### 3.5.1 OTBSAF Simulation Framework

OTBSAF version 1.0 [SAIC 2001A] is distributed with over 5,000 pages of documentation describing 636 libraries and 5 applications. The libraries have over 1 million lines of executable code with 80% written in the “C” language and the majority of the balance in Finite State Machine (FSMs). The FSM [SAIC 2001E] code generator is an AWK script that translates the FSMs source files into C-language constructs.

Semi Automated Forces (SAF) a simulation system that can provide operator controlled semi-automated entities that can maneuver on the simulated battlefield similar to a manned simulator. The goal of OTBSAF is to replicate the outward behavior of simulated units and their component vehicle and weapon systems to a level of realism sufficient for training and combat development [SAIC 2001C]. Utilization of OTBSAF takes advantage of a large range of domain knowledge to model all simulated systems, implementation knowledge including networking, and user interfaces, available from the various organizations that use and adapt the SAF to their needs.

In order to modify a system with OTBSAF's complexity it is necessary to grasp the design methodology. OTBSAF was designed for the Replacement of Individual Subsystems, Hardware Independence, Programming Language Independence, Distribution of Subsystems, and Time Management for scheduling of execution. For Replacement of Individual Subsystems, the methodology chose layering the architecture, support for object-based programming, and definition of rigorous interfaces. To enable the distribution of subsystems, OTBSAF uses two protocols. The DIS protocol is used

for sharing of entity appearance data and the PO protocol to ensure persistence of objects despite hardware failure. Time management was implemented to allow allocation of computation resources to all the services and subclasses of the layered system [SAIC 2001C].

The layered characteristic of the OTBSAF led to the implementation of services. Distribution of information between the calling module and the service was implemented in two ways. The first was by polling the higher layer periodically for information about a state change. The other was the establishment of a callback mechanism for the handling of a simulation event. In this case, the higher layer module provides the lower layer module a function to call, if the specified event occurs [SAIC 2001C].

Data Driven Execution in OTBSAF refers to the system's use of data files (referred to as reader [.rdr] files) to provide detailed descriptions and parameters of objects. This allows the characteristics and behavior of the objects to be changed without recompilation. Not only does this allow definition of the objects at creation time, in some cases it allows reconfiguration of objects during run-time. Together these features were used to facilitate the adaptation of OTBSAF to the Concurrent Model Approach of this dissertation.

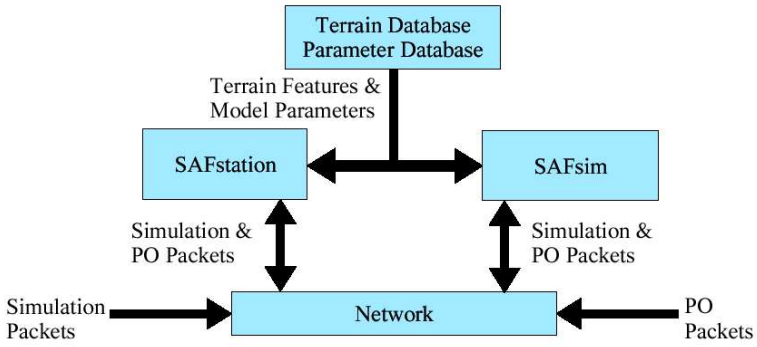


Figure 23. SAFstation and SAFsim Components

### 3.5.2 Application

OTBSAF is distributed between *SAFstation* and *SAFsim* components as indicated in Figure 23 derived from the User's Manual [SAIC 2001B]. SAFstations provide the run-time user interface to the simulation, as many workstations as necessary can be used as long as they reference the same DIS exercise number. SAFsims provide the simulation of entities. Here as many computers as necessary to simulate the required number of entities can be used on the same exercise. SAFstations share control of entities among all components that use the same PO database identifiers. Different forces can be privately controlled by using different PO database numbers identifiers while participating in the same exercise number. Thus, this enables different teams to interact independently within the same exercise. More than one exercise can share the same network as long as they use different exercise identifiers.

Selections of static values for the total execution time of the simulation occur at invocation of OTBSAF. This includes such items as exercise and PO-database identifiers, the seed for the random number generator, and the configuration of OTBSAF

as a *station*, a *simulator*, or combined as a *pocket SAF*. In addition, more than one instance of OTBSAF can be invoked on the same computer/workstation at the same time, as long as memory and processor power can support the demand.

The static (**S)** databases in Table 4 are provided by Terrain and Parameter Databases block of Figure 23. The moderate update (**M)** databases are provided by PO databases, and the highly dynamic (**H)** databases are provided by the exercise database. When using OTBSAF the changes to the **M** and **H** databases of Table 4 are realized accordingly using the PO and Simulation packets of Figure 23.

Table 5 provides a summary of the packets generated by OTBSAF in the scenario introduced in Chapter 1. All the packets with "po_" as part of their name modify the PO databases, the rest of the packets modify the exercise database and are shared with other DIS applications. The second column of Table 5 provides the number of each packet that was transmitted over the course of the exercise of 12 minutes duration. The third column presents the size of each message in bytes. Those with fractional sizes were variable length packets so the size represents the average length of all the packets of the given type. It was developed by taking the sum of the individual packet lengths divided by the total number of $s_{ave} = \frac{\sum_{i=1}^{N} s_i}{N}$ packets , where $s_i$ is the size of the individual packets. The fourth column, which was determined experimentally and is covered in later chapters, indicates whether this packet is used in maintaining congruence. The relative frequency in number of packets/second can be found by taking the entry in column 2 and dividing

by 720 which is the number of seconds in the exercise. The average bandwidth (BW) required for the exercise can be calculated by summing the number of packets for each entry times that entry's packets size in bytes times 8 the number of bits in a byte and dividing by the exercise duration of 720 as follows:

$$BW = \frac{\sum N_O \cdot s_O \cdot 8}{Duration} = \frac{\sum N_O \cdot s_O \cdot 8}{720} = \frac{4{,}966{,}280 * 8}{720} = 55{,}180\, bits/sec$$

Table 5. Current Communications Packets

| OTBSaf Packet Type(*O*) | Transmittals(*N* | Size(*s*) | Congruence |
|---|---|---|---|
| acknowledge | 24 | 32 | |
| aggregate_state | 26 | 160 | |
| entity_state | 4,059 | 176 | |
| po_delete_objects | 1 | 52 | Y |
| po_line | 48 | 116 | |
| po_link | 26 | 1,120 | Y |
| po_objects_present | 143 | 954 | |
| po_overlay | 102 | 72 | Y |
| po_parametric_input | 116 | 90 | |
| po_parametric_input_holder | 51 | 64 | |
| po_point | 78 | 100 | Y |
| po_simulator_present | 141 | 100 | |
| po_task | 1,150 | 122 | Y |
| po_task_authorization | 15 | 72 | |
| po_task_frame | 147 | 247 | Y |
| po_task_state | 1,486 | 289 | Y |
| po_unit | 202 | 648 | Y |
| po_variable | 9,028 | 356 | |
| start_resume | 6 | 44 | Y |
| stop_freeze | 12 | 40 | Y |
| transmitter | 780 | 104 | |

Note that the two most frequent entries in the table, *entity_state,* and *po_variable* are not required to maintain congruence and as will be seen in chapter 7, not all of transmittals of

even the types of packets needed for congruence will be required . Also note that this is for a very small scenario nowhere near the size indicated in Table 2 that could be of interest for the situational awareness application. Also, Table 5 does not include any of the packets transmitted to initialize the simulators as this would be part of the parameter database. Other items that are not transmitted are the Terrain Databases, the one for this exercise is 6 MBytes compressed or 60 MBytes expanded for use during the simulation. Other parameter entries of about 43 MBytes. High resolution terrain/feature databases used to generate the three dimensional Mod-Stealth views of Figure 22, could approach 1 Gigabyte after being extracted from an 80 Gigabyte source database. Video data to generate images such as the VSAM views of Figure 22 for the full length of the exercise of 12 minutes at 525-line television resolution and using MPEG compression would be about 400 Mbytes for each view.

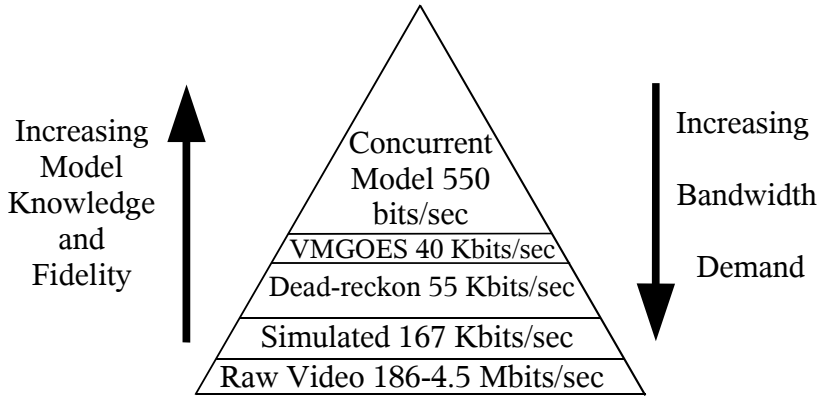



Figure 24. WWLAN Data Reduction Pyramid with Data Rates

So in review of Figure 24, if we start with a single viewpoint the base of the pyramid would start at 186Megabits/sec using MPEG compression it would drop to about 4.5 megabits/second. Using simulation with pre-stored terrain and parameter databases we

could drop this to 167Kbits/sec and using dead-reckoning to 55,180 bits/sec and we are proposing to drop it to below 550 bits per/sec. Only the base section of the pyramid is restricted to a single viewpoint. Once the simulation domain is entered it is possible to choose any arbitrary viewpoint and display multiple viewpoints at the same time.

## 3.6 OTBSAF Scalability and Priority Queue Performance

In the previous sections, we have reviewed the benefits of in bandwidth reduction by using simulation as a motivation for the concurrent model approach. With the concurrent model approach, we introduced the need for maintaining congruence. This section addresses the needs of the physical implementation associated with the concurrent model approach. One of the problems inherent to the concurrent model approach is the need to generate all the entities at each separate site. This raises the question as to its feasibility. How much processing power is required? This question has existed since the introduction of OTBSAF and its predecessors, one such study for MODSAF 3.0 presented the results shown in Table 6 as reported by Roberts et al [Roberts 1998].

Table 6. ModSAF 3.0 Benchmark Results [Roberts 1998]

| **Number of SAFsims (Processors)** | **Total Entities Modeled by all SAFsims** | **Average Entities Per Processor** |
|---|---|---|
| 1 | 160 | 160 |
| 2 | 264 | 132 |
| 4 | 384 | 96 |
| 8 | 640 | 80 |
| 15 | 720 | 48 |

This study was based on a 16-node Silicon Graphics Origin 2000 computer where separate SAFsims were invoked on each processor for the number of processors shown from 1 to a maximum of 15 processors. These results were obtained from use of the standard ModSAF benchmark. All the processors in this study were identical. Roberts attributed the major source of this nonlinear scaling to the situation that each processor had to dedicate more and more processing time to the increased number of Protocol Data Units (PDUs) exchanged between SAFsims.

To update these results to personal computers, a similar test using the OTBSAF "-benchmark" option was used. This option allows you to specify the number of platoons to use in the benchmark, which thus increments the number of vehicle entities by four for each platoon. A heterogeneous system with three computers was used. Host "bahrd" was a Dell Inspiron 8000 laptop using a Pentium III processor running at 1 Ghz. It had 0.5 Gbytes memory and was running the Linux 2.4.20 kernel. Host "bahr2" was a generic desktop using an Athlon 2400+ processor with 0.75 Gbytes memory running the Linux 2.4.22 kernel. Host "bahr3" was another generic desktop using an Athlon 2600+ processor with 2.0 Gbytes memory running the Linux 2.6.7 kernel.

Table 7. OTBSAF Benchmark Results

| | Number of Processors | | |
|---|---|---|---|
| Host Machine | 1 | 2 | 3 |
| Bahrd | 180 | 0 | 100 |
| Bahr2 | 256 | 188 | 168 |
| Bahr3 | 280 | 200 | 176 |
| Total vehicles | 280 | 388 | 444 |
| Incremental | | 108 | 56 |

Table 7 gives the result of these benchmarks. The test was conducted by iteratively changing the values until the threshold between pass and fail was discovered. The entries in Table 7 reflect these results, where the first column identifies the computers by their host-name, the second column shows the maximum number of vehicles when each computer was operated separately. The "Total Vehicles" row reflects the maximum number of vehicles that could be generated by the respective number of computers. The third column reflects the results of taking two computers at a time, and likewise the fourth column represents taking all three computers at the same time. The "Incremental" row reflects the number of vehicles that were added to the exercise with the addition of another computer. These results again reflect the nonlinear scaling reported in the earlier report.

The final report of the Synthetic Theater of War experiment conducted in 1994 [Tiernan 1995], identified similar problems with scalability, reporting the maximum number of entities on a common network occurred with 10 computers. They overcame the scaling problems by isolating portions of the network and only sharing the necessary PDUs between the different segments. Other reported solutions reported making changes to the OTBSAF/ModSAF architecture using for example only one copy of the PO database in a shared memory common to all processors.

One item to note from Table 7 is that the faster processor continued to support more entities, implying that increased computation efficiency can contribute to the total number of entities that can be simulated. Another item discovered in the experiments is

that the speed of the LAN between the computers did not change the results, as both a 1 Gbit and a 100Mhz LAN were evaluated using “Bahr2” and “Bahr3”. The laptop was restricted to a 54 Mbit wireless link. At least at this level of entity counts current LAN technology is not the limiting factor.

To update the current state of the art even further, we had the opportunity to test a cluster of gaming machines. Two of these machines were based on AMD 64 Processor, an XP3000+ model, and a Pentium 4 Processor. All three of these machines had a 1 Gigabyte main memory, and ran at a 2 Ghz clock speed. Table 8 provides the results of this benchmarking effort. Definitely the two AMD 64 machines identified as Bob and Rednight were faster than the Pentium, however the Pentium fell midway between the 2 Athlon Processors although their model numbers indicated they would be faster in a standard office benchmark. The two AMD 64 machines actually did perform more than 1.5 times faster as proposed by their 3000+ model number. Again note the non-linear scaling reported in the other tests which motivate the following section.

Table 8. OTBSAF Benchmark Results - Part II

| | Number of Processors | | |
|---|---|---|---|
| Host Machine | 1 | 2 | 3 |
| Redscull | 268 | 0 | 148 |
| Bob | 412 | 276 | 244 |
| Rednight | 412 | 280 | 256 |
| Total Vehicles | 412 | 556 | 648 |
| Incremental | | 144 | 92 |

Specifically, in Chapter 2 we introduced causality, synchronization, and DES as important tools to maintain the congruence necessary to make the concurrent model approach feasible. We also showed how the event queue's priority queue data structure could impact the execution efficiency of a simulation. We showed that Order(1) priority queue had distinct advantages over Order(N) and Order(log N) implementations. Currently OTBSAF uses a heap which is Order(log N) implementation. We introduce here an Order(1) implementation extending the author's previous work [Bahr 1994B], to include head end optimizations and a high stability solution for a large range of event insertion and deletion distributions. We will call this queue a *Smart Priority Queue* (SPQ) because it uses bounded heuristics to dynamically adjust its structure to adapt to the event distribution. This approach improves the efficiency of the simulation and minimizes the negative impact of some additional techniques of scaling such as prioritized frequency of execution as indicated in Table 2 in Chapter 2 where additional events would be added to priority queue but not require as many model executions and PDU generations. The number of events in a priority queue for a given number of entities ***N*** in an exercise is ***K*N*** where ***1<K<U***. ***K*** is a multiplier characteristic of a simulation to represent the spawning of additional parallel events for each entity other than just a routine update. An example would be a routine status check at a much lower rate than the normal update. Assume that ***U*** representing an upper limit will be 10 or less depending on how many parallel activities are scheduled during the processing of the $N^{th}$ entity. Observations during debugging of the SPQ implementation for OTBSAF captured a peak of about 10 scheduled events per entity, and routinely between 3 to 4 events per entity. Together, these results will be used to develop the Concurrent Model Approach in chapter 4.

# CHAPTER 4 CONCURRENT MODEL APPROACH TO EMBEDED SIMULATION

One approach used to reduce the communications traffic required during Distributed Interactive Simulation (DIS) is the use of *dead-reckoning* algorithms [Dahman 1996]. Dead-reckoning takes advantage of knowledge of the physical behavior of entities which dictates that moving bodies can only change speed or direction in certain predictable ways. As long as the moving body does not deviate from this predicted route, there is no need to send additional information from the source monitoring the movement to the receiver using the motion information to determine the current location of the entity.

The DIS concept provides large number of entities by adding additional Semi-Automated Forces stations (SAFstations). As shown in the previous section, each SAFstation can generate from 100 to 300 entities. The current state of each entity is updated periodically by the generating SAFstation. The problem with this approach is the periodic update traffic. Although the *Concurrent Model* still uses periodic updates, they are at a much longer period between each update.

In the *Concurrent Model* approach, the principles underlying dead-reckoning are extended. In dead-reckoning, at both the local source and remote receiver, an algorithm

is executed based on the positioning information provided by the source to the receiver as shown in Figure 25.

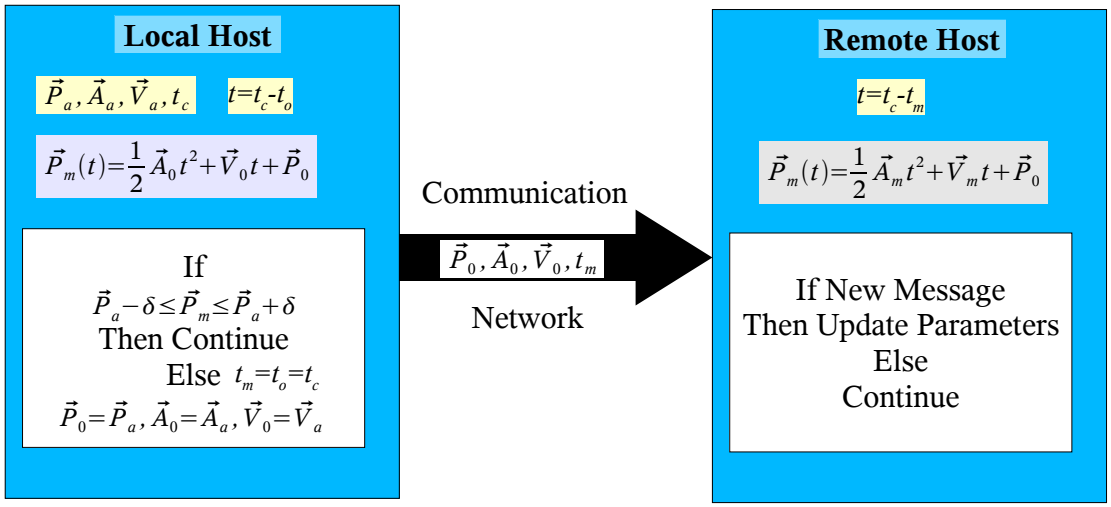



Figure 25. Dead-Reckoning

The local source continues to compute the current position as indicated by the vector $\vec{P}_m(t)=\frac{1}{2}\vec{A}_0 t^2+\vec{V}_0 t+\vec{P}_0$ and compare that to the measured position indicated by $\vec{P}_a, \vec{A}_a, \vec{V}_a, t_c$. As long as the calculated position is within certain error bounds $\vec{P}_a-\delta \le \vec{P}_m \le \vec{P}_a+\delta$ then no updates are provided to the receiver. Meanwhile, the remote receiver calculates the predicted position $\vec{P}_m(t)=\frac{1}{2}\vec{A}_m t^2+\vec{V}_m t+\vec{P}_0$ and uses it as the current position of the moving body, as it has confidence that, in the absence of correcting information, this position is accurate within the error bounds. In this case, the dead reckoning algorithm represents a model of moving-body positioning. Thus, positioning information is the parameter that is exchanged between two copies of the model. In the

above equations $\vec{P}(t)$ is the position vector at time ***t***. $\vec{A}$ is the acceleration vector, and $\vec{V}$ is the velocity vector. Thus given the $\vec{P}_0$ the position at the start of the period and ***t*** the change of time since the position was observed the new $\vec{P}(t)$ can be determined.

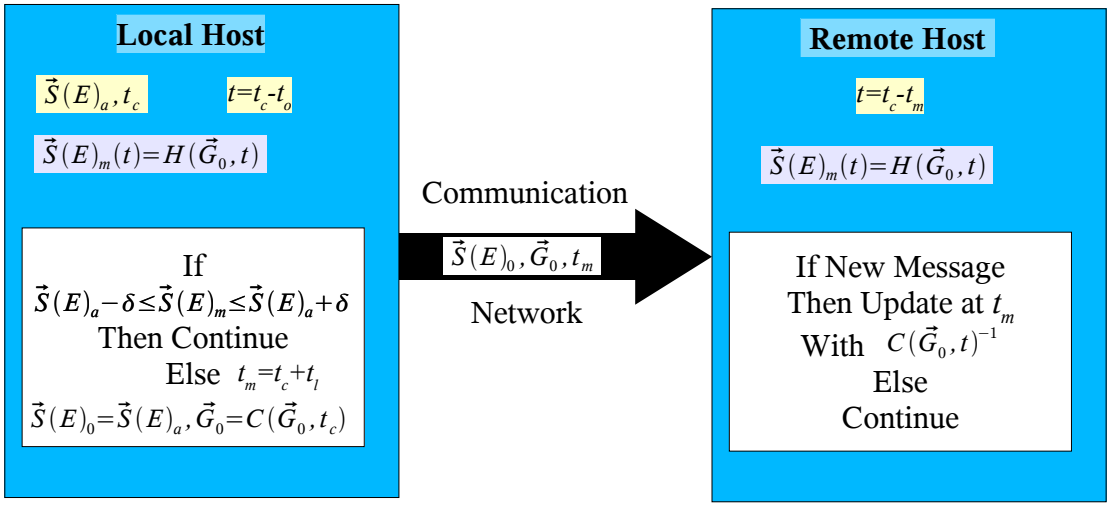


Figure 26. Concurrent Model

In the *Concurrent Model,* dead reckoning is extended to predict the behavior interaction between players. The approach uses pairs of full-platform models, rather than only sub-element models. The difference between this and the dead-reckoning approach is this employs two high-fidelity models $\vec{S}(E)_m(t)=H(\vec{G}_0,t)$ as indicated in Figure 26. Ideally, the required correction data $\vec{S}(E)_0, \vec{G}_0, t_m$ is non-existent if the behavior model is sufficiently accurate. At the source the observed data $\vec{S}(E)_a, t_c$ where is $\vec{S}(E)_a$ the state vector for each entity ***E*** as actually observed as indicated by the subscript ***a*** where subscript ***m*** designates the model. The initial time $t_0$ is subtracted from the current time $t_c$ to determine the change in time ***t*** which is used by the model to determine the models current state vector. As long as the current modeled state vector $\vec{S}(E)_m$ stays within the

acceptable range $\boldsymbol{\delta}$ of observed vector $\vec{S}(E)_a$ there is no need to update the model. If it exceeds the acceptable range then the correction function $\vec{G}_0 = C(\vec{G}_0, t_c)$ must be applied to determine a new congruence vector $\vec{G}_0$ which will be applied at time $\boldsymbol{t_m}$ which is a future time computed by adding the latency correction time $\boldsymbol{t_l}$ to the current time. This allows both the local source model and remote receiving model $\vec{S}(E)_0, \vec{G}_0, t_m$ to be adjusted concurrently. Only the revised initial state vector, Congruence vector, and designated time of invocation are transmitted to the receiver. This minimizes the data transfer between the remote interactive parties, and yet maximizes responsiveness, while allowing detailed manipulation of articulated components at the local level. An interactive situation requires pairs of models for each participant. Essentially, the respective platform is cloned on the target platform. An exact clone would respond identically as the simulated platform and crew, since it is collocated with the target there would be no measurable delays, thus resulting in the highest fidelity simulation. In actuality, cloning the crew and platform is impossible, but cloning a model is routine. Thus, the proposal is to place a high fidelity model of the simulated crew and platform on that platform, and in a closed-loop environment tune that model to match the capabilities of the platform. Concurrently, place a clone of the model on interacting objects and, in an open loop environment, apply the same corrections made to the reference model to its clone, thereby keeping it a clone of the reference model.

The early research on the Concurrent Model Approach was split between multiple research teams. SAIC corporation explored the adaptation of Modular Semi-Automated Forces (ModSAF) to the concurrent model approach [McHale 1998] [Ourston 1998].

Gonzalez, DeMara, and Geogiopolos developed smarter models [Gonzalez 1998]. It is assumed that high-fidelity models/simulations will be available, differences between the player platform can be detected, models can be adjusted to minimize the errors of prediction and technology will provide the necessary, cost effective, computational resources [Petrasko 1993]. Even with these assumptions the Concurrent Model Approach is still highly dependent on maintaining coherency between the reference models and their clones.

The focus of this dissertation is then on:

(1) How scheduling can be synchronized to maintain concurrency.

(2) How concurrency can be maintained in a multithreaded environment to allow scalability.

(3) What technological capabilities are required, in terms of data storage, computational capability, and communication bandwidth.

(4) Generalization of the approach for application to other uses. The goal is to allow each platform maximum independence to adapt to its unique demands while maintaining the coherency between the reference models and their clones.

By using concurrent models clones at both ends of the communications link, we can also compensate for delays in the feedback loop. The error detection and correction loop would exist only at the source end and the receiving end would operate-open ended by just utilizing the correction parameters sent from the source end. There always exists the possibility that the correction parameters sent to the receiving end could be lost or have

errors so some method of verification will have to be employed. The goal is to tune the model to where it accurately emulates the crew and platform being modeled. This is much different from just changing the output to agree with reference system. It means analyzing the error to determine what model manipulations are required so the error would not have occurred. In view of the situation where we have humans in the loop, we will never have a perfect predictive model of behavior so in this case the goal would be to minimize the average error.

As introduced earlier in this paper, even if we had perfect models and the error correction data was non existent then, there would remain a requirement for communications of other information between the field platforms and the rest of the participants. What the Concurrent Model concept achieves is a reduction to the minimum data while removing transmission delays from apparent reaction times. The data that is required to be transmitted is primarily shared data. There is initially information that could be pre-stored at each entity, such as, terrain database, expected weather conditions, the set of generic models, and force composition. During the exercise, this information would have to be supplemented by new orders, intelligence information, and changes of status of any player in the field of view.

At times we adopt a one-on-one trainee interaction whereas for each platform it is in reality a team-on-team environment. To cope with the team interaction issue, the requirement for models executing on each platform must be extended to include one for

each player in the field of view. However, this does not increase the number of required reference models, as all clones of player would be subject to the same correction data.

## 4.1 Player Units (PU)

The Player Unit (PU) could be a person, a manned platform, an unmanned vehicle or a simulation of such an object. The key characteristics are that it is independent, capable of making decisions, and interacts with other units. Communications between units is wireless as any hardwired system is part of a larger unit. Interaction can be either threatening or supportive.

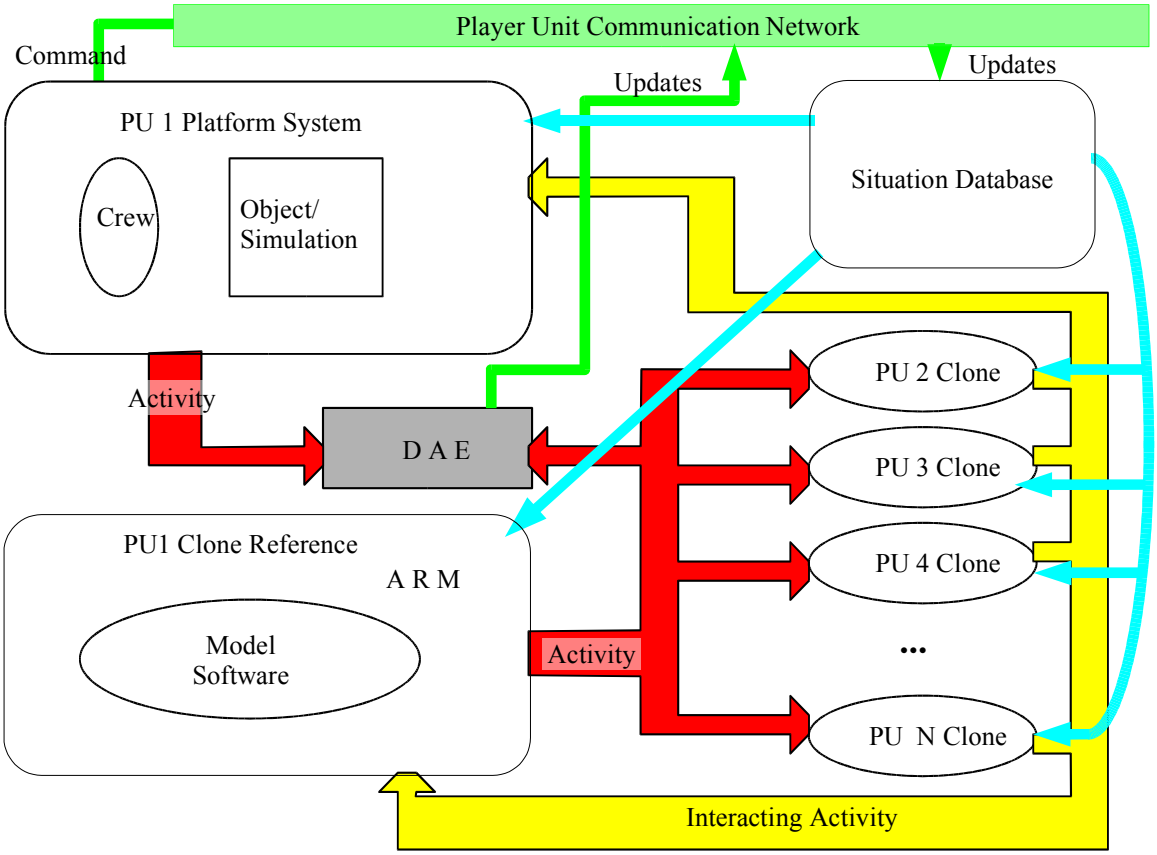



Figure 27: CRM

### 4.1.1 Concurrent Remote Model (CRM).

The *Concurrent Remote Model* (CRM) shifts the transfer of data away from the interaction parameters to be primarily the typical Interaction Situation Information, with model tuning parameters as required. The interaction information as depicted in Figure 27 still exists and is available in greater quantity, higher precision, and with less delay than by either of the previous methods. The difference is, all this information is generated locally at each platform. The CRM platform consists of the major elements identified in Figure 27. They are the object simulator or object system and its requisite instrumentation. The Difference Analysis Engine (DAE) that replaces the comparator in the dead-reckoning approach. The Adaptive Reference Model (ARM) that serves as either the reference model or a clone of the reference model and the situation database. Each of these blocks is described in more detail in the following sections. As illustrated in Figure 27, the play of the simulator or the manned platform is only noted locally, the play of the reference model is the "official" view of the interaction. This allows a consistent view across the exercise while still allowing individual evaluation to take place at the platform level. Since the reference model and all of clones are changed synchronously, they play the same for a given situation regardless of location. The maintenance of the situation database thereby becomes the primary purpose of the PU communications network. Since this database should be the same on any platform participating in the same conflict location, the information on this link can be broadcast to all platforms.

In comparison to the normal DIS network, all the communications labeled *activity* and *updates* in Figure 27 would be transferred over the wireless PU communications network instead of just the updates. This is in addition to the normal operational traffic of the players. The goal is to reduce the updates to a fraction of the normal traffic, whereas DIS traffic would tend to be an order of magnitude larger as discussed later.

### 4.1.2 Difference Analysis Engine (DAE)

The DAE is the element that compares the performance of the simulator with the reference model and develops the parameters that are passed to the reference model and its remote entity clones. It develops the parameters that are used to adapt the ARM. This is the primary place where the states as defined by the simulation or object system are used. The status reported to the rest of the interacting elements is the output of the reference model. However, in this subsystem, the results from the platform system are treated as absolutely correct, the results of the reference model are considered as flawed if differences occur. The parameters generated from its analysis will be used at some time in the future. This delta between current time and future time is design dependent but can be large enough to ensure correct transfer to all clones. It is assumed that changes will be made to the clones synchronously with the reference model. The synchronous time base will probably be based on a global time-base such as GPS time. Next to a object/simulation, this is probably the most highly customized portion of the concept. This subsystem depends on internal knowledge of how the object/simulator generates results, how the reference model generates results, and what the prescribed solution is. It also takes advantage of the operators' history to improve its predictions.

This is the subsystem that uses Artificial Intelligence techniques to determine why the parameters need to be changed and what changes to make. This subsystem learns how a specific manned object performs, converts that knowledge into a set of parameters that it transfers to models of the object, and expects those models to perform as if they were clones of the manned object.

### 4.1.3 Adaptive Reference Model (ARM)

The *Adaptive Reference Model* (ARM) is the element that will be cloned to serve as the reference model and the remote entity models. It is anticipated that this model is constructed from a set of generic modules. Along with this would be parameters that would differentiate this particular object system from the others in its class. In addition to the object system capabilities model, operator model is included. This gets into modeling things such as reaction time, target recognition, driving tendencies, and impulsiveness. This would be an unbounded task except that characteristics of the physical platform and training narrow the range. Other modeling required is for those characteristics that tend to vary over the course of the interaction, or due to changes in capability during the simulation. A key characteristic of these models is that the performance of the model can be adjusted in real-time during use. The model must continuously generate as an output state, all outputs that determine the location and status of the object system and its operators. All parameter changes to this model are applied synchronously to the reference model and all clones. That is, parameter changes are received with the time that they are to be applied. Then when the prescribed time is reached the changes are made. The reference model directly interacts with clones of the

target systems, while the clones interact with the reference models of the target systems. The state as generated by the reference model is taken as the state of the platform used.

### 4.1.4 Instrumentation for Player Units

The instrumentation for PUs is the set of sensors that are used to determine the state of the Objects Platform and its operators. It must provide the location and time measurement, stores status, and articulated components status of the platform. The stores would include for example fuel for any vehicle based ES. While this information is readily available on simulators, instrumentation will probably have to be added to most live platforms. Future research is proposed to determine the required accuracy and resolution of these sensors.

### 4.1.5 Situation Database

The Situation Database is data that is stored on each platform required for concurrent simulation to work. Assuming that a model can always adapt more precise information to the level of detail that it requires, the level of detail required for each element is that required by the most discerning live platform or the reference model. Component data-bases would include such items as depicted in Table 4. Items such as Terrain-database, Threat-platform models, Pre-defined orders, Vulnerability data, and Predetermined parameters are all quasi-static. That is they must be identical for all Player Units, but the changes are outside the scope of this discussion. Items such as weather data, and obstacle

data, and learned reactions would be considered low-dynamic elements that would have minimal impact on network capacity.

The goal is to separate situation database information from dynamic components which can be generated locally by models. The OTBSAF elements that would make up the Situation Database would be the DIS database, the PO database, the Terrain database, and the Parameter database. The first two are dynamic and the last two are provided/selected at simulation initiation. The DIS database provides the dynamic state of each simulated object, while the PO database provides command and control information, such as unit makeup, orders, and other parameters.

## 4.2 Processing of Discrepant Results

The primary results used for battle assessment is the states generated by the clone reference model. These results should be identical at all sites as they are synchronously updated for all copies and they participate in the same simulation. They only interact with other reference clones. The only place other results are observed are at the individual players and their local DAE. These results may be used for evaluation of the model, but will primarily be used early in the development cycle. The DAE uses the results to adjust the reference model and its clones, so the disagreement is noted and will influence future results. For this reason, it would be beneficial if the model parameters could be transferred with the trainees from exercise to exercise. The comparison of the trainees to clones responses can be made available for individual assessment independent of the exercise results. No matter how perfect the clone, there exists the possibility that

results achieved by the simulator or live platform differs from the reference model. Some key items that were kept in mind while deciding how important their congruence would be and determining the congruence truth thresholds are:

- This is a simulation whether live or virtual trainees are involved.
- The degree of agreement needed between both views of the interaction.
- If the ES objective evaluates actual interaction or a response to training.

If corrective actions are desired, they should be inserted as realistically as possible within the normal response times of human interaction to avoid negative training. All of these factors provide room for engineering tradeoffs, while still meeting the end users needs.

A basic premise of the concurrent model approach is that the clone models executing in other platforms will perform identically with their reference model. Even if the models give repeatable results for the same set of inputs, it does not guarantee that at any given time the results are the same. Two other constraints are required. The same data must be presented to the clones in the same sequence as the reference model and the clones must be at the same point of execution at that time. The most direct approach at guaranteeing these constraints are met, is to have each clone processor be identical to the reference processor and having them execute from the same clock from the same data stream. However, this would defeat the purpose of using the clones. The purpose of using the clones is to allow the generation of the same output stream at physically separated locations at the same time. This physical separation in some cases could be thousands of kilometers away. A second function of using the clones is to allow interaction with other elements of the simulation within the natural reaction times of the human operators. A

third characteristic of these clones is that they can be dynamically changed to give a different result to the same data source to realize adaptive behavior. Let the set of output states be denoted as $O$, input states as $I$, reference models as $r$ and clones as $c$ with the present state as $n$. For each change of output state from $O^r_n$ of the reference model to $O^r_{n+1}$ the states $O^r_n = O^c_n$ and $O^r_{n+1} = O^c_{n+1}$ must hold true for all clones of the given reference model. Furthermore, there must be no human perceptible difference in the time of occurrence of the state changes between any of the models including all clones and the reference. Since model $c$ is a clone of model $r$ then $I^r_n = I^c_n$ and $I^r_{n+1} = I^c_{n+1}$ must also hold true. For the purposes of training these states do not have to be exact, but rather within an envelope dictated by the ES application. This envelope is dependent on the required fidelity to achieve objectives. For this reason these equations can be restated as $O^r_n \pm \delta = O^c_n$ , $O^r_{n+1} \pm \delta = O^c_{n+1}$ , $I^r_n \pm \delta = I^c_n$ and $I^r_{n+1} \pm \delta = I^c_{n+1}$. This allowable error term must be specified before exact performance expressions can be determined. However, a set of expressions can be postulated, incorporated into a simulation, exercised and presented to Subject Matter Experts (SME) to gain insight on the magnitude sensitivity. Upon completion of this experimental cycle the equations can be refined to a set of design criteria. The first step of this experimental design process is to postulate a design, then examine this design for error sources and develop these equations. The next step would be model this set of error equations and examine their cross dependencies. After this initial design refinement, the postulated changes could be incorporated into OTBSAF and sample runs presented to SMEs for further assessment of relative importance.

## 4.3 Concurrent Model Approach Design Criteria

In the Concurrent Model approach, each platform has a set of models that interact during the training exercise. Figure 28 shows multiple CRM platforms that are participating in a collective training exercise.

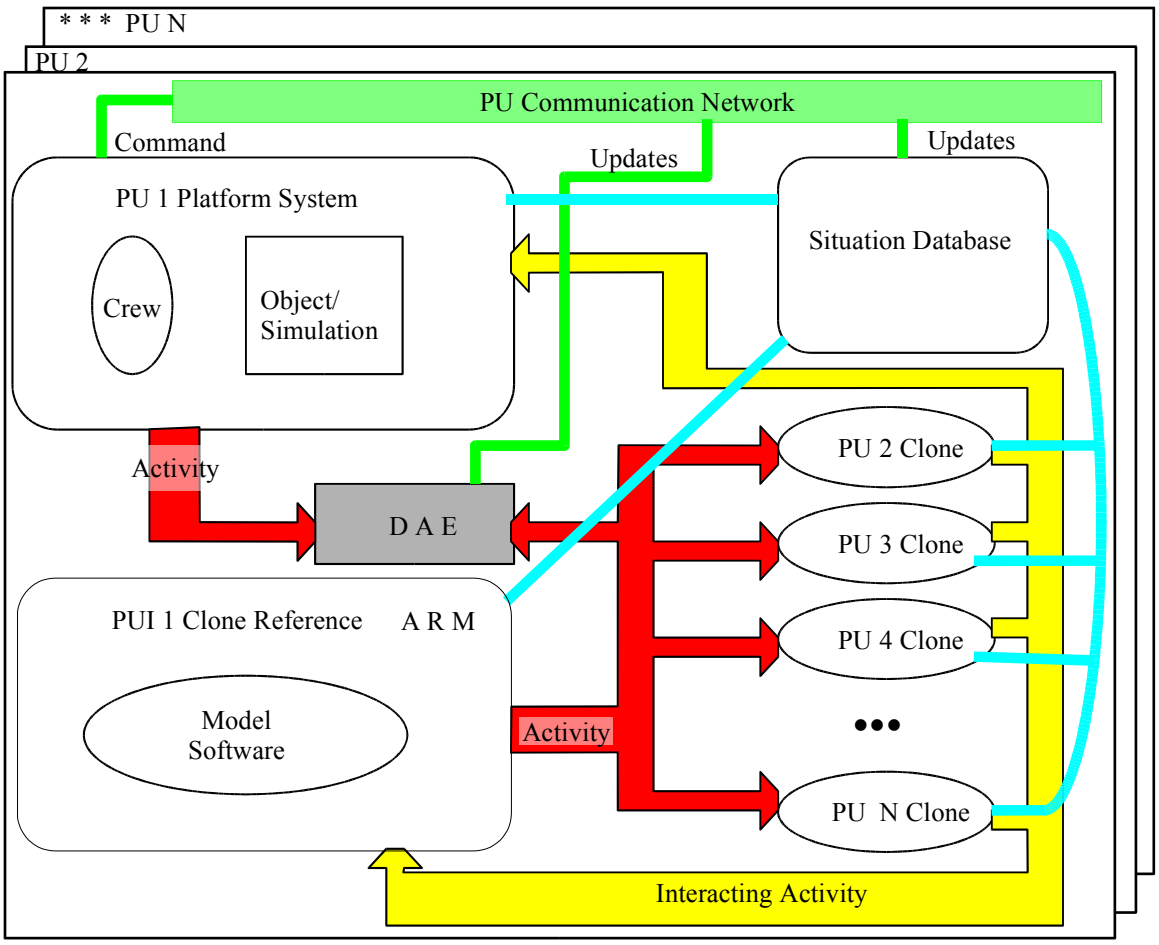



Figure 28: Collective Interaction under Concurrent Model approach

Each one with his own set of models and his own copy of the situation database. The challenge here is to maintain the coherency between the platforms within error $\boldsymbol{\delta}$. In chapter 3 we introduced the congruence functions $\boldsymbol{\Psi_T}$ and $\boldsymbol{\Psi_B}$ and their corresponding

truth functions $\boldsymbol{\Gamma_T}$ and $\boldsymbol{\Gamma_B}$. The is for the local host to dynamically adjust the congruence vector $\vec{G}_0$ to keep $\boldsymbol{\Gamma_T}$ and $\boldsymbol{\Gamma_B}$ TRUE.

### 4.3.1 General Criteria

The proposed coherency strategy is as follows:

1. Each reference model will broadcast an entity state message that includes both a time tagged model parameter set and a separately tagged model status set on a periodic basis at least three orders of magnitude less frequent than the local updates.
2. On a event-driven basis, an entity state message will be broadcast to correct both the status and the model parameters based on DAE discrepancy sensing.
3. Each system will have a real-time clock locked to GPS time.
4. Each player platform will be responsible to update data to the current time based on model parameters and the difference between tagged time and current time.
5. Local Model release times will be based on the Real-time clock.
6. Local Model tick rates will be adjusted on a integer multiple of a base period basis, to best meet the demands of the local system.
7. tick rates are to be adjusted on a dynamic basis to ensure that local calculations stay at near real-time.
8. All pseudo random number generators are advanced on the base period. All calculations will be adjusted to minimize the impact of dynamic scheduling. Ideally all random numbers will be replaced by fixed values.

The rational behind each Concurrent Model strategy element is as follows:

1. The periodic broadcast provides a known base state to correct for any clone drift, and also provides the information necessary to add a new entity into a vehicles area of concern. Since this change may generate a ripple effect on calculations, it needs to be at a frequency that only increases total processing load by a small percentage. For example if, the additional processing required per periodic update is 10 times greater that of normal tick processing and its rate is $10^{-3}$ of the normal tick, the average processing load has only increased by 1%. Separate time tags allow for freedom on order and processing priority.

2. Event driven processing is used near real-time corrections and keeps errors within a pre-defined delta.

3. Each system needs GPS for position determination already so this provides a low cost approach of providing time stability at a level where its error contribution is relatively insignificant. Most parameters are rate based so time differences are always a term in each state update.

4. This allows each platform to update its state based on the data currently available to it. Since it has a highly stable time source, the update becomes independent of transmission and processing delays.

5. It is assumed that processing will be based on a priority queue with time as a priority. Information should be processed as soon after it is scheduled as possible.

6. Some model factors are based on counts or increments in order to simplify corrections required to compensate for dynamic loading. A similar simplified scheme is used to account for time base differences.

7. Normally, models are processed on a periodic basis, but due to external events the amount of processing required can change. To get the best precision, the number of updates needs to be adjusted based on external activity.

8. Pseudo-random number generators are used to provide variety in training scenarios. Yet, for coherency purposes each vehicle needs to be able replicate the activities of the reference models independent of the local vehicles tick rate. This is a tradeoff issue that must be carefully clarified with the end user.

Once a suitable scheduling system has been developed, this system is used for experiments in multithreading on a local network to establish the validity of local processor scalability. ModSAF has already been demonstrated to be scaleable from a distributed computer viewpoint, but its current single threaded architecture with scheduling at the application level does not allow it to take full advantage of modern Symmetric Multiprocessing computers which use multiple processors to share the computation load.

### 4.3.2 Remote SAF Operator

The Remote SAF Operator is an application that demonstrates the capability of meeting several key criteria of the concurrent model approach by modifying OTBSAF. The

remote SAF operator requires that an operator at one location controls semi-automated forces that are interacting with other players conducting Mission Rehearsal en route to a trouble spot. In this case, there is a clone pair of simulations at the operators location and with the deployed unit. Communications between the two model sets is via a multi-hop wireless system to aircraft en-route to the trouble spot. Thus, it takes concurrent simulations, one to provide the SAF entities for the Mission Rehearsal, and the other to provide feedback to the operators. Latency must be hid to avoid having the operators wait for the commands to reach the aircraft, initiate the action of the simulated entities, and send entity state packets back to be displayed on their screens. Instead, the local simulation provides that feedback. Communications bandwidth can be reduced because the entity state packets do not need to be sent back to the operators. real-time synchronization is required to keep the clocks in sync at both sites, and repeatable performance for both sets of models must be maintained.

The Remote SAF Application as implemented for this dissertation does not demonstrate the complete Concurrent Model approach. It's primary focus is on demonstrating the communications reduction potential of this approach. A GPS based clock was not available, and it would require a more extensive modification to OTBSAF to be incorporated. This was not critical at this stage of investigation. In addition, the use of the DAE and feedback have had some initial exploration by VMGOES [Henninger 1998] [Gerber 2001]and SAIC [Ourston 1998], and further work in this area should be explored in the future. Furthermore, although some preliminary investigation was conducted on the criteria and implementation it was only partially implemented at this time.

The primary focus of of this investigation is the tradeoffs between bandwidth reduction and the impact of congruence. OTBSAF provides a vehicle for investigating these capabilities, but its extensive use of random numbers needs further investigation beyond the tested alternatives. SAIC used a non-network approach for *Repeatable SAF*, and the Concurrent SAF experiment used a single master random number generator. Both showed success in repeatability, but were lacking in other aspects of the complete Concurrent Model approach.

# CHAPTER 5 ANALYTICAL RELATIONSHIPS IN COMMUNICATION MECHANISMS

To demonstrate properties of the the Concurrent Model approach, we first require temporal congruence and behavioral congruence between two entity state streams at physically distinct locations using a communications channel, ***C***, with a finite bandwidth of ***B*** bits per second and exhibiting characteristic latency of $t_l$ and behavioral generators $G_L$ and $G_r$. The initial congruence parameter vector $\vec{G}_0$ is updated to correct these generators as needed to maintain congruence.

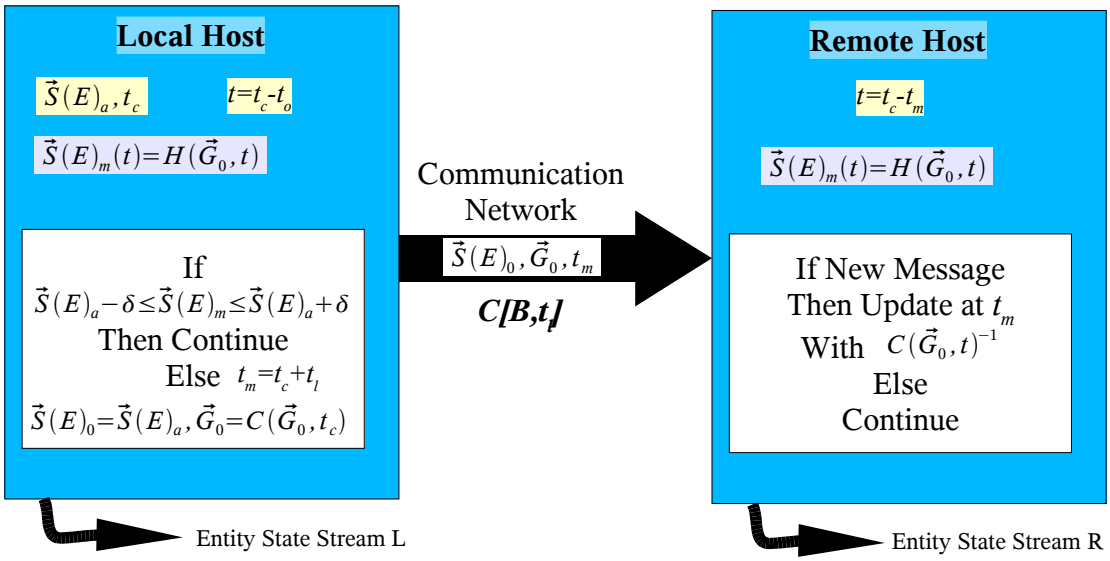



Figure 29. Concurrent Model Analysis

Figure 29 added the details of Figure 26 to Figure 20 to illustrate how the earlier introduced concepts relate. In this case, it is given that the **Local Host** generating entity state stream $\vec{S}(E_L)$ information is separated by some physical distance ***d*** from the remote location where the **Remote Host** is generating the entity state stream $\vec{S}(E_R)$ information. For the **Remote Host** there is both a $H(\vec{G}_0, t)$ generator component and a $C(\vec{G}_0, t)^{-1}$ control component. The subscript ***a*** denotes measured or actual values, whereas the subscript ***m*** denotes model generated values. Subscript ***0*** denotes initial values. The control signals are sent over the communications link ***C,*** which is limited by its characteristics ***B,*** and $t_l$, from the **Local Host** to the **Remote Host**. We provide theorems governing the following classes of characteristics:

- Correctness characteristics: the factors that determine congruence, and
- Performance benefit characteristics: bandwidth and latency assessments.

## 5.1 Correctness Characteristics

*Definition 5.1.1: Congruence.*

Congruence is achieved between two entity state streams $\vec{S}(E_L)$, and $\vec{S}(E_R)$ when the views generated from those streams allow the independent *observer* to react to those views in a correct and timely manner. The standard for correct and timely are based on the *observer's* reaction to the same views if they were generated by a single stream in a DIS environment. Congruence is subdivided into Behavioral Congruence and Temporal Congruence. Thus $\Gamma = \Gamma_B \wedge \Gamma_T$ where $\Gamma$ is the *Congruence Truth* function and ^ denotes

conjunction. $\Gamma_B$ is the Behavioral Congruence Truth function and $\Gamma_T$ is the Temporal Congruence Truth function

*Definition 5.1.1.a: Behavioral Congruence.*

Behavioral Congruence is achieved between two entity state streams $\vec{S}(E_L)$ and $\vec{S}(E_R)$ when the view generated by remote receiver matches the view generated by the local source within an acceptable tolerance $\delta$. Let $\Psi_B(E_L, E_R)$ denote the behavioral congruence function, then $\Gamma_B$ is said to be TRUE if $\Psi_{B0}-\delta \leq \Psi_B \leq \Psi_{B0}+\delta$ otherwise it is FALSE. $\Gamma_B$ is the truth function for behavioral congruence, $\delta$ is one half of the acceptable range, and $\Psi_{B0}$ is the desired value.

*Definition 5.1.1.b: Temporal Congruence.*

Temporal Congruence is achieved between two entity state streams $\vec{S}(E_L)$ and $\vec{S}(E_R)$ when the view generated by remote receiver occurs within the same timeframe as the view generated by the local source. When $\Psi_T(t, E_R)$ is the temporal congruence function, then $\Gamma_T$ evaluates to TRUE if $\Psi_{T0}-\delta \leq \Psi_T \leq \Psi_{T0}+\delta$ otherwise it is FALSE. $\Gamma_T$ is the truth function for temporal congruence, $\delta$ is one half of the acceptable range, and $\Psi_{T0}$ is the desired value.

*Definition 5.1.2: Simultaneity.*

Simultaneity is defined as the scheduling of two or more events at the same simulation time.

*Definition 5.1.3: Causality.*

Causality is the property that no event should appear to the *observer* prior to any event that caused it. No simultaneous event can exhibit causality for another event scheduled at the same time.

*Definition 5.1.4: Strong clocks.*

Strong clocks satisfy the following relationship introduced by [Lamport 1978]. Let $\rightarrow$ denote the happening before relationship for members of the set $\zeta$. For any events $a, b \in \zeta \; a \rightarrow b$ then $\Pi(a) < \Pi(b)$. Where $a$ and $b$ are discrete events and the function $\Pi(x)$ returns the timestamp for the event $x$.

*Definition 5.1.5: Repeatability.*

Repeatability is the property that states for every instance of a model $\vec{S}(E)_m(t) = H(\vec{G}_0, t)$ given the same set of parameters and state, it must generate the same output, irrespective of physical location or clock time. Here the time parameter for the model is the change in time since the previous update, not the wall clock time.

*Definition 5.1.6: Soft Real-time Scheduling.*

Soft Real-time Scheduling is defined to be a process scheduling methodology where the process is not initiated until the real-time clock reaches the scheduled time, however, it all processes are executed that are scheduled at that time in some sequential order until they are all completed. This means that the processes are not guaranteed to be executed at the scheduled time, but are guaranteed not to be executed before that time. Soft Real-

time Scheduling also guarantees that all processes scheduled for an earlier time are completed before any subsequent process is executed. This methodology will be elaborated later using Figure 30 and its related discussion.

*Definition 5.1.7. Simulation time.*

Simulation time is the logical time maintained by a discrete event simulation, and refers to the scheduled time of the last event selected for execution. It remains constant until the next event is scheduled for execution. A simulation implementing this strategy satisfies the the requirements for provision of a strong clock.

The properties of Concurrent Models are analyzed under the following assumptions:

(1) The priority queue provides strict First-In First-Out (FIFO) ordering for all equal priority events.

(2) The real-time clock maintained at both source and receiver are synchronized to GPS time.

*Theorem 5.1.1. Necessary and Sufficient Conditions for Behavioral Congruency.*

$\vec{S}(E_L)$ is behaviorally congruent to $\vec{S}(E_R)$ if the models are repeatable and they are given the same inputs in the same order.

*Proof.* Given that the models are repeatable, this implies they will generate the same outputs given the same inputs. By requiring the priority queue to preserve FIFO ordering in the presence of simultaneity, this maintains ordering even when events with the same

priority are processed sequentially. Thus, they will remain in the same order in the remote execution as the local execution even if the next scheduled iteration occurs at different clocks as long as the time step increment ***i*** is the same. With this strong ordering causality is also maintained because $\boldsymbol{a} \rightarrow \boldsymbol{b}$, then $(\boldsymbol{a}+\boldsymbol{i}) \rightarrow (\boldsymbol{b}+\boldsymbol{i})$ for all event pairs (a,b). Where → denotes the *happening before relationship* of the strong clock.

Theorem 5.1.2. Temporal Congruency

$\vec{S}(E_L)$ is temporally congruent to $\vec{S}(E_R)$ if the simulations are soft real-time synchronized to global time such as GPS time, and all changes are received ***s*** seconds before scheduled execution time, and are processed in the same order as transmitted. Where ***s*** is congruence setup time.

*Proof.* Given that the simulations are synchronized to GPS time, which has higher resolution than the OTBSAF one millisecond clock, then individual simulation times can be advanced an identical real-time rate. As long as $\vec{S}(E)_0, \vec{G}_0, t_m$ are received ***s*** seconds before scheduled execution time, soft real-time synchronization guarantees all processes will be executed after the scheduled time and in clock order. As long the changes are tagged sequentially, the receiving system can properly order them within the same clock period. Since this is again a strong clock ordering causality is still maintained. The processing time required to compute $C(\vec{G}_0, t)^{-1}$ must be no more than ***s*** seconds.

## 5.2 Performance Benefit Characteristics

Performance benefits of the concurrent model approach address the primary characteristics of a communications system. Those are bandwidth and latency. The concurrent model approach is postulated to address the limitations of a communications system used for mobile systems operating. The focus is on either a satellite-based or a multi-hop wireless network. These systems tend to have restricted bandwidths for simulation traffic and long latencies. As such the objective is an approach that has a reduced bandwidth demand, can operate with extended latencies, and mitigates the impact of communications outages.

*Definition 5.2.1: Reduced Bandwidth Ratio.*

Reduced Bandwidth Ratio denoted by $\boldsymbol{B_R}$ is the ratio of the bandwidth used by current the DIS approach over the bandwidth required for the postulated Concurrent Model approach. In this case, the number of bits $\boldsymbol{N}$ transmitted on the network for local traffic divided by the number of bits $\boldsymbol{I}$ transmitted between the local and remote generators of the concurrent model approach yielding $\boldsymbol{B_R = N/I}$.

*Definition 5.2.2: Latency Hiding.*

Latency Hiding is the combination of providing low latency solutions for highly dynamic state changes, and latency compensation techniques for other changes. Latency hiding techniques compensate for when the messages are present in the network but delayed due to transmission characteristics.

*Definition 5.2.3: Outage immunity.*

Outage immunity is the situation where the output of the remote site continues with minimal degradation during periods of communication outage. Outage immunity techniques compensate for situations when messages are lost or not transmitted due to a loss of transmission capability.

The concurrent model approach provides data reduction by transmitting only those packets necessary to update the models. It does not send any entity state packets, and only a subset of the persistent object packets. From Table 5 we see that only 10 of the 21 categories of messages are transmitted. Furthermore, in Chapter 7 we will see that only a small percentage of packets in the transmitted categories are needed on a regular basis.

*Theorem 5.2.1: The concurrent model approach provides reduced bandwidth demand.*

*Proof.* ***N*** is the sum of all local packets transmitted. ***I*** is the sum of the packets required to maintain congruence. From Table 5, it is clear that ***I*** is a subset of ***N***, therefore $B_R = N/I$ is greater than unity so $B_R > 1$.

Simulations latency is the length of time it takes for a message about an event to travel from one simulator to a remote simulator. It includes various communications delays such as protocol formatting, amount of other traffic on the link, number of links/hops between simulations and transmission time. Other factors include reliability and routing.

Reliability influences the average latency as a retransmission may be required before the message is received by the remote simulation. Routing delays are prevalent in wireless systems where the route is subject to change as the systems move as given by the total time:

$$T_{tot} = t_{prot} + t_{wait} + N_{hop}(t_{hop} + t_{dist}) + t_{rel}$$

where:

$t_{prot}$ = time required to format the message according to the protocol,

$t_{wait}$ = time caused by waiting for link due to contention,

$N_{hop}$ = average number of hops times the

$t_{hop}$ = average delay per hop,

$t_{dist}$ = per hop fly time due to physical length of links,

$t_{rel}$ = average time to correct an error in a message times the expected number of errors per message.

Theorem 5.2.2. *The concurrent model approach provides a latency hiding ratio of* $\frac{t_{prot} + t_{wait}}{T_{tot}}$.

*Proof*: The concurrent model approach generates high dynamic states at the remote location thereby reducing latency to that of a local network. The concurrent model approach compensates for the latency of other changes by scheduling them far enough in the future so they can be executed in synchronism. Thereby meeting both requirements of the definition.

Theorem 5.2.3. *The concurrent model approach provides outage immunity ratio of up to* $\boldsymbol{B_R}$.

*Proof:* The concurrent model approach generates high dynamic states at the remote site. Once this generation is started it continues until changed by new commands. As opposed to the current system which generates the high dynamic states at the local site and transmits them to the remote site. If communications is dropped the state information is lost until communications is reestablished. Thus the concurrent model approach provides outage immunity as it continues to provide high dynamic state information even during communications drop outs. As long as the communications system at least provides the capacity for the Congruency data the system will continue to operate.

# CHAPTER 6 CONCURRENT MODEL CONSTRUCTS AND MECHANISMS

## 6.1 Background

Concurrent Model Constructs and Mechanisms were initially developed around an initial premise that the Remote SAF Operator (RSAFO) is capable of isolating the commands generated during the simulation. Thus, only those packets that initiated the commands would be transmitted from one simulation to the other. As discussed in Chapter 3, the architecture did not readily yield a definitive answer on how this could be integrated into the extensive OTBSAF code. This led to the necessity to develop techniques to track down what action initiated which packet generation and identify all the other modules that activated the generation of similar packets, so to determine where to isolate them.

The first approach was to assume that the command packets would be unique, and only that subset of packets could be filtered from the total packet stream. This proved to be problematic because only some could be isolated, and not to a significant level. The second approach was to consider the user interface. For each button pushed it is noted in which routines ultimately generate a command packet. This proved to be difficult because of they layered approach with a mixture of polling and callbacks separating the

services and the originating functions, and the reuse of libraries to accomplish similar functions. This investigation through public and private interfaces certainly introduced some of the challenges imposed by an object-oriented programming solution adopted by OTBSAF.

The evolved investigation system consisted of a significant use of `greps` nested inside of `for` loops traversing the many libraries, coupled with the setting of judicious breakpoints inside of dynamically-debugged instantiations of multiple communicating processes. Reverting to the command line interface of the operating system inside a window, coupled with the Dynamic Debugging Tool called `ddd` provided the necessary environment. A further aid was to take advantage of the documentation that was created in "texinfo" format to ultimately combine all the documentation into one large hyperlinked file of 5,200+ pages. Having this documentation allowed the ready transversal from top level discussion down to the Programers Reference Manual [SAIC 2001E] details and back. It also allowed the use of the internal `find` command to locate other possible using modules.

## 6.2 Integrated Model Execution

OTBSAF uses one main program depending on command line options to function in one of three modes. The first SAFstation or `-gui` mode provides the user/operator interface. This is the key operational mode for collecting the external commands at runtime. The second SAFsim or `-sim` mode operates in the simulation-only mode, which generates all the entities and provides the update packets that are eventually graphically displayed

on the SAFstation. The third or default mode is a pocket SAF that has both functions combined in one executable. The use of the same executable to operate in all three modes made it necessary in some cases to identify the mode of operation in order to properly isolate the modifications. In addition, the same libraries are used for other applications. Where possible, changes were made to OTBSAF that only required a command line parameter to differentiate between concurrent model operation and the distribution version. Those changes are discussed here. Any other specialized functions that were created will be discussed in Section 6.3.

Some of the modifications for the SPM [McHale 1998], were required for the RSAFO. These include isolation of display update functions based on fixed real-time increments. They are based on an update rate and not a sequence of events. They avoid referencing event generating functions which could adversely affect causality using the simulation-time clock. Using the same seed for multiple invocations of the simulation to generate the same sequence of random numbers was retained. Isolation from the network was abandoned for RSAFO by necessity, as RSAFO needed to generate entity state streams. Scheduling was changed to enable discrete event simulation that would advance based on a fixed relationship to the real-time clock. Simulation-time will not advance before this relationship is met. We provide scaled to real-time in this mode, but not independent from real-time. The real-time clock is used to advance the discrete event scheduler from a release time viewpoint. Figure 30 provides a graphical representation of the provided capabilities.

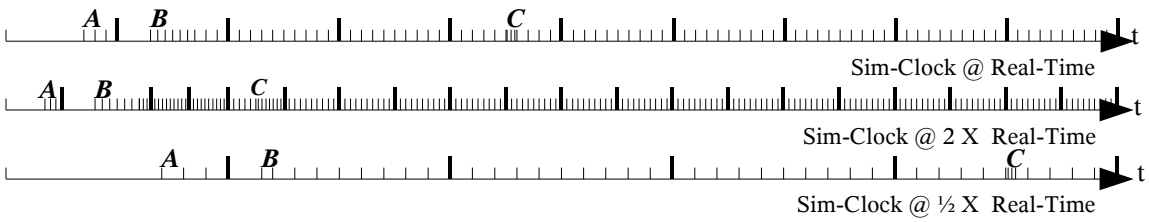



Figure 30. Real-Time and simulation-time Relationship

The top line represents a real-time tick of once every 10 milliseconds. The second line is scaled to a 2 times real-time rate. The third line is scaled to 1/2 times the real-time rate. Each timeline is annotated with simulation-time events occurring at the indicated tick on the lines. Point ***A*** represents the first event as occurring at time 0.07 sec., In line one scaled to real-time this will occur at 0.07 seconds in real-time. On the second line scaled at twice rate, while it occurs as 0.07 seconds simulation-time, it is executed at 0.035 seconds in real-time. On the third line, scaled at half rate, it occurs at 0.14 seconds in real-time.

At point ***B*** in Figure 30 there is a situation where the event at simulation-time 0.10 requires a physical time to execute longer than the next clock tick. Since this is a physical execution period, it takes the same time interval in real-time independent of the scaling, although the start of event at simulation-time 0.10 occurs at the scheduled simulation-time, the next event scheduled at 0.11 occurs as quickly afterwards as it can. This has a ripple effect on each successive event until total execution period of all successive events matches the passage of the event stream. On line, one this occurs until simulation-time/real-time reaches 0.17 sec., on line two this occurs at simulation-time

0.40 and real-time 0.20, while on line three the only event delayed is the event at time 0.11 seconds.

Point ***C*** on the timelines depicts the situation when more than one event is scheduled at simulation-time 0.45 seconds In this case, the events are executed as fast as possible. In the real-time and half rate case they do not delay the next scheduled event. However in the twice rate case they again cause a simulation-time ripple out until the 0.50 second event. This example also indicates to obtain a similar effect as "as-fast-as-you-can" simulation. Thus, the Concurrent Model uses a scaled clock that advances faster than the discrete event scheduler can advance to provide an equivalent mode of operation to the OTBSAF "-fast_time" option.

Next, a separate communications channel was established to allow the SAF operator's SAFstation to communicate with the remote SAFstation as shown in Figure 36. For the purposes of this study, it was implemented as a separate named pipe for each receive and send function. The name is relative to the invoking directory so multiple pairs of named pipes can be established. Connection between the pipes is established external to the simulation. This enabled using a `tee` connector that allowed the tapping of the information being sent between the separate simulations for recording the traffic. It also allows various filters to be used as necessary to match the chosen communications medium and protocol. For this study, the direct connection with a `tee` was used. To allow asynchronous reception, the receive function was scheduled as a periodic task that checked for the presence of data in a buffer, and if present, empties the buffer prior to

returning control to the scheduler. If a partial message is received then it performs an OS `sched_yield` to allow another task to run. The repetition rate of receive function needs to be high enough not cause the remote send to block. The send function executes as needed until the buffer is full or all data is transferred. If the buffer is full, it blocks until it can send the message.

To maintain hardware configuration independence at the respective sites, it is imperative to allow the PO databases to be independent. While the vehicle identifiers from the operator 's viewpoint must be consistent, the assignment of vehicle to simulator, and etc. should be independent. Thus, the receive function is given the responsibility to perform the mapping of the Vehicle identifiers to the PO identifiers and simulator identifiers. Since, the various PO PDUs use nested fields, the receive functions parse these fields to map all of their entries.

## 6.3 Isolated Model Functions

In observing the operation of the SAFstation, it was found that changes were introduced to the simulations by modifying the PO database either by creating a new database object or changing an existing object. These changes are then transmitted to all systems using the same PO database. For the concurrent model approach, only the initiated actions need to be shared between simulations. Furthermore, repetitive transmission of changes is not desired as it would interfere with the simulations of the second concurrent system. Modifications were necessary to further isolate the execution paths of routines to only those directly initiated by the human operator, not those initiated by the simulation or

recursive execution of the same code. In addition, the originating SAFstation needed to complete local operations as well as transmit the changes to the remote SAFstation.

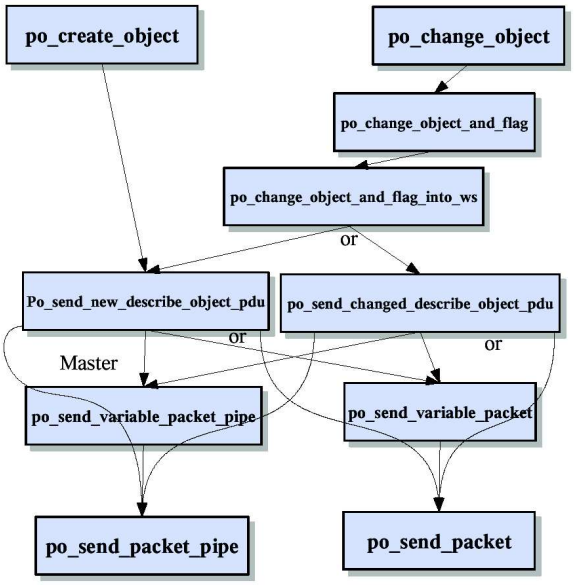



Figure 31: PO object Modification

The remote SAFstation needs to receive to changes from the originating SAFstation and merge those into the local database. To allow independent startup and configuration of systems at both locations, it was necessary to provide an object/simulator mapping methodology. This involved parsing the appropriate fields of each object and mapping the values from the originating SAFstation environment to the remote SAFstation environment. This mapping was primarily initiated upon the receipt of the `create_new_describe_object` entry. In some cases entries that were created by a SIMstation, would need to be modified. In this case they were assumed to be hashed into entries by the same displacement on both the originating and remote SAFstations, and mappings for the missing entry and all intervening entries were created in the mapping

table by retaining that relationship. To-date, that has proved sufficient to handle the mapping requirement.

For robustness purposes, OTBSAF transmits all PO PDUs 10 times at one second intervals after each change is made. This conflicts with the objectives and assumptions of the Concurrent Model approach and could be eliminated from the information being transmitted between the concurrent SAFstations. Therefore, similar functions were implemented as indicated in Figure 31, to separate those PDUs that were sent to both local and remote networks, and those only sent locally. In some cases, the only location to make such a distinction was at a higher level than in the PO library. In this case, a new but similar function was created so that the appropriate action could be selected at the higher level. All functions that call `po_send_new` or `changed_describe_object_pdu` are appropriate for establishing the SAFstation PO database, those functions are listed in files under `libpo`. Three new functions were created to isolate SAFstation operations to use a pipe between master and clone as identified with the extension of `_pipe` to their name.

## 6.4 Smart Priority Queue Data Structure

In order to accommodate non-uniform distribution of event list activity, we developed and evaluated several different structures based on modifications of the calendar queue. Once the benefits were fully understood, a distribution-adaptive data structure was developed to provide stable performance across a wide range of distributions [Bahr 2004].

The characteristics of the RDMS model emphasized that an optimal queue structure for event list management requires low overhead for the accesses that occur most frequently, for example, head insertion and deletion. Furthermore, the structure should incur the minimum number of comparisons for insertions. In addition, the list structure must be capable of accommodating rapid arbitrary deletes. The usage distribution can be difficult to describe analytically and its characteristics can change throughout a simulation run. However, a user of discrete event simulator or library of simulator tools need not be burdened with the selection of the appropriate priority queue structure for a particular simulation. Ideally, the priority queue used for event list management should be close to optimum for all expected distributions. To satisfy these objectives, a dynamically-adaptable queue structure is required to consistently meet the following performance goals:

- minimize the total number of operations required to access the most frequently scheduled events,
- reduce the overhead cost of sample taking, resizing, and finding the new head, by reducing their frequency of invocation, and
- perform threshold testing only when beneficial.

Nonetheless, adaptive mechanisms create the potential for oscillation. Although Brown identified the potential for oscillation, models used in his experiments did not excite that condition. In an adaptive algorithm, the cost of employing feedback during discrete event simulation directly increases simulation overhead, i.e. the same processor is employed for both the direct actions and the feedback operations. So a realistic

assessment of performance of an adaptive strategy needs to take the additional overhead from feedback into account.

Notably, the data structure with the least absolute operation cost is the linear singly-linked list. However, the linked list loses its performance advantage if its length exceeds about 10 entries [Jones 1986], so an indexed structure such as a calendar queue can limit the list length, yet still accommodate the entries that overflow this length. Calendar queue performance can be adjusted by altering the width and number of bins. The primary disadvantages of adaptive structures are the three components of overhead: sensing, evaluation, and adjustment. This penalty can be reduced by minimizing the number of operations executed in the primary execution path and then amortizing high-cost routines such as adjustment over a large number of HOLD operations. The working of the SPQ Figure 32 en queue operation can be explained using the flowchart in Figure 33.

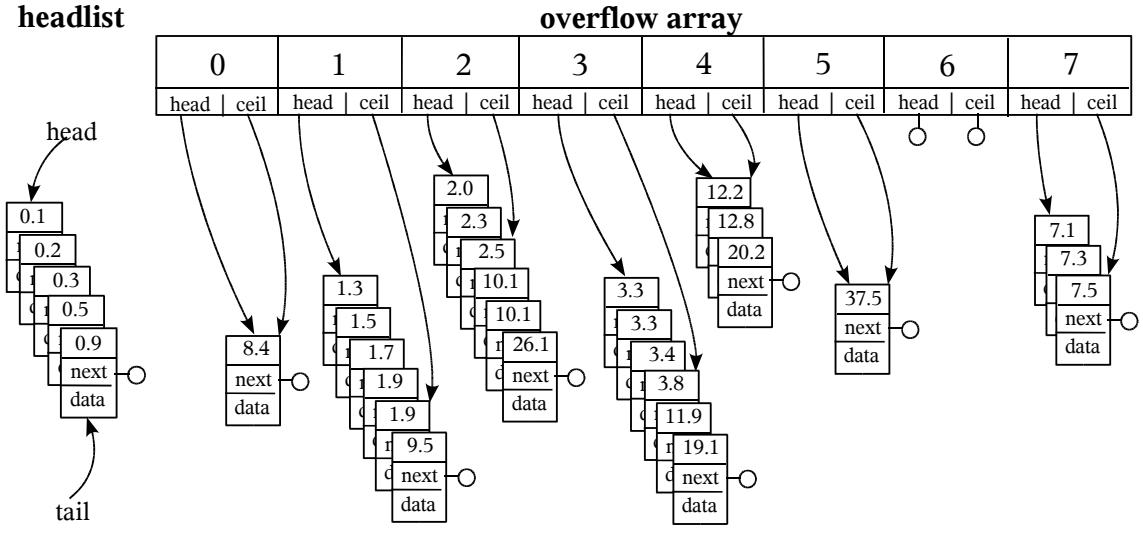



Figure 32. Smart Priority Queue (SPQ)

The SPQ structure of Figure 30 differs from the Calendar Queue of Figure 15 in that it is a linear queue with an overflow structure of a modified Calendar Queue. The linear queue is identified as `headlist` and calendar queue as `overflow array` in Figure 32. The difference in the `overflow array` from the normal Calendar Queue is that the second pointer to each bin points to the last entry of the current year and is identified as the `ceil` pointer in the array, instead of pointing to the last entry in each bin such as the `tail` pointer in Figure 15. The SPQ en queue operation accommodates these changes as demonstrated in Figure 33 by having separate branches to accommodate entries in the `headlist` or the `overflow structure`, which is subdivided for insertions before or after the `ceil` pointer. Counters are provided in the branches (`linear_head`, `linear_dist`, and `overflow`) to keep track of the number of insertions and the number of overly long scans (`head_over`, `bin_over`, and `bin_ceil_over`). These counters are used in the `OptimizeQueue` block to provide the information used by the heuristics to decide which adjustments are indicated. Note that the `OptimizeQueue` block is only executed if a dynamic activity threshold is exceeded. The thresholds are only adjusted by the `OptimizeQueue` block. Additional routines not diagrammed that are part of the hold operation, are `GetHead` and `FindHead`. `GetHead` is called every time the simulation clock advances, the majority of the time this is simply removal of the first element of the `headlist` leaving the next element as the head. The next most frequent operation occurs each time the `headlist` is emptied. In this case, the next bin of the overflow structure is transferred to the `headlist`. This is accomplished by setting the head pointer equal to the `bin head` pointer and the tail pointer equal to the `bin ceil` pointer. The `bin head`

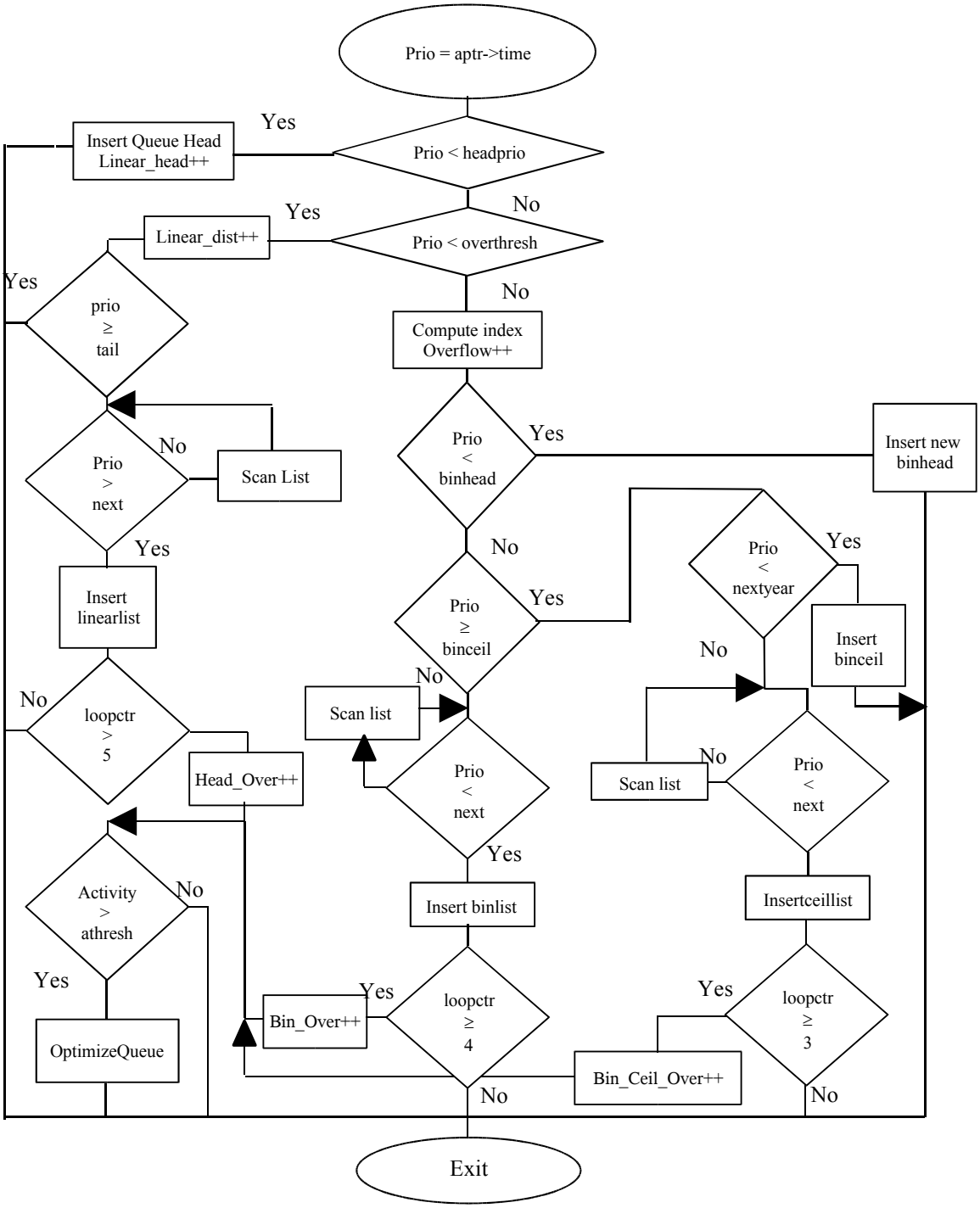



Figure 33. SPQ Insert Operation

pointer is then set equal to the next element of the bin ceil list and bin ceil pointer is moved to point to the last element in the list of the next year. The element pointed to by the tail pointer then has its next pointer set to NULL. If the next bin does not have any current year elements, the FindHead routine is called. This routine locates the next nearest element in the queue and adjusts all the bin head and ceil pointers to the new current year. For most queue distributions, FindHead is rarely called. The New_bin_count counter is provided to keep track of the number of headlist changes and the get-head counter is provided to track the total number of deletes from the head of the list. The FindHead routine provides two additional counters to provide additional feedback on the event distribution in the overflow structure these are skip bin and search which report the number of bins skipped to find the next element, and in the case where there are no elements were found in the current year, that a search for the lowest element of all the bins had to be conducted.

### 6.4.1 Activity Feedback Counters

The first three activity counters allow tracking of the individual paths and calculation of queue size and average insertion cost. New_bin_count allows the calculation of average number of get-head operations per head_list change. If this number is too low then it indicates that the bin-width is too narrow. The head_over and bin_over indicators are counts that give an indication of the bin_width being too wide and indicates an increase in the average number of compares per insertion. Bin_tail_over is an indication that the calendar year is too short i.e., not enough

bins. This also indicates an increased number of compare operations. `Skipped-count` is another indicator of narrow bins as well as an indicator of bursty data. It reflects the additional hold cost used in calculating optimal queue configuration. `Search` gives an indication of large gaps in data and has a cost proportional to the year-size or number of bins. This indicator triggers the recalculation of the number of bins.

### 6.4.2 Sensing Cost

The first cost of an adaptive algorithm is sensing. In Brown's case, he chose tracking of the queue size. For the SPQ case, since we also wanted to check other characteristics as well, the counts for each path were maintained separately. Different combinations of these counts could be used to determine queue size, activity, and where excessive operations occurred. Yet, we sought to keep the complexity of tracking information at the same level as Brown's. The other costs can be reduced by periodic sampling. The sampling strategy used is based on the observation that a change in the structure of the event list is not necessary until the distribution has substantially changed its characteristics. This is indicated by either excessive compares required for insertion, or excessive number of bins traversed between successive head lists. Therefore, insertion compare counts are monitored, and if they exceed a threshold, further tests are invoked. Likewise the number of bin changes are also monitored.

The SPQ is designed to minimize the total number of queue overhead operations by reducing the equivalent number of compares in the most frequently executed operations. The most frequent operation, other than examining the head of the queue, is removal of

the head element. The overhead other than the counting operations over a linear list, is that the next pointer indicates a end of list every time the head list empties. This signals the SPQ to transfer the following bin from the overflow structure to the head list. To optimize the queue, it is possible to trade off the frequency of changing the head list to the number of elements searched in the head list. For this purpose, the head list is initialized to contain 5 elements as a minimum to invoke this tax at most 20% of the time. The second factor is to minimize the number of operations required to move the list from the next bin to the head list. This was improved by having a pointer to the last element in the current year maintained during insertion. The result is to transfer the bin head and tail pointers to the head list and then setting the bin pointers to the head and tail of the next year. On the average the latter step requires one additional compare operation. In the case study due to the high insertion activity rates close to the head, this adjustment was found to occur even less than this design goal at only 7% of the HOLD operations.

The next most frequent operation was insertion at the head of the queue. In this case, the SPQ behaves the same as the linear list with only one compare required. The next choice is to determine whether the new priority will be inserted in the head list or the overflow. Once the decision is made to place the event in the overflow structure, the SPQ operates very similar to the calendar queue. The result of using a separate head list is that for all elements stored in the calendar structure there are two additional compares, but over 50% of the insertions have been avoided. Bin hashing calculations and all head deletes are recouped in the bin indexing operation. Another enhancement over the calendar queue is that the second pointer into each bin is not strictly a tail pointer, but rather a pointer to the

head or the last entry in the current calendar year. This adds one additional compare for those events inserted over a calendar year away, but minimizes the number of compares required to transfer bin data to the head list. Insertions over a calendar year away occur rarely by the resizing design of the calendar queue.

Each list body insertion is monitored for the number of compares required to find the insertion point. If this exceeds a threshold value then further evaluation is initiated. The first test performed in the evaluation is to determine whether enough operations have occurred since the last restructuring of the queue for a new restructuring to provide potential benefit. This is a simple threshold comparison based on values calculated during the previous restructuring. This test serves as a damping function to en sure the SPQ doesn't spend more operations adapting than it can save by restructuring.

### 6.4.3 Filtering Costs.

The second cost is analysis or filtering. In Brown's case it was a simple threshold test: if the queue size was greater or less than the thresholds, queue resizing was required. For SPQ, heuristics are used to first determine whether a change is required and second what the parameter values should be for the resizing.

Six counters are used to monitor the performance. These are `linear_head`, `linear_dist`, `overflow`, `delete_count`, `get_count`, and `new_bin_count`. The first three of these reside in the separate branches taken in the priority queue for any insertion. The last is incremented every time a bin is moved to the

head list to minimize the cost of tracking the queue performance. All other monitoring is invoked only when an insertion occurs outside of the expected range. These are considered overflow conditions and are recorded by sensor counters for each of the following conditions:

- `Head_over`: an insertion occurs in the head-list that takes over 5 compares.
- `Bin_over`: an insertion occurs in a bin that takes over 4 compares.
- `Bin_ceil_over`: an insertion occurs on the end of bin-list that takes over 3 compares.
- `Skip_count`: the number of empty bins skipped, and
- `Search`: the number of times a search had to be performed to find a new queue head.

When any overflow condition occurs, the first level of testing occurs. This first level is simply to determine if adequate activity has occurred in the queue to justify a change. It is a threshold that is proportional to the queue size denoted by *N*. This test is a comparison of the sum of the activity counters to the threshold. The second test is to determine if the operational cost to make a change is less than the cost of allowing the overflows to continue. If the threshold is exceeded then optimizing calculations are made to determine the predicted queue parameters. These are filtered and compared to the current parameters and if the changes indicate improved queue performance the queue is adjusted. In all cases, the expected performance improvement should outweigh the operational cost of making the change.

### 6.4.4 Correction Costs

The final cost of feedback is correction. This is very expensive since it means configuring a queue with corrected parameters then moving the contents from the old queue to the new one or adjusting existing contents accordingly. Another consideration for feedback is response time, or how soon are adjustments made after the distribution changes. To make the queue more responsive if a change is required, the activity threshold is reduced to a factor only several times larger than the current queue size from previously used nominal value of 100,000 operations. This has an effect of doubling the threshold until it exceeds 100,000, which is then used as the steady state sampling threshold. All heuristics have been grouped into one module for easier tailoring. The cost factors used in the cost minimization heuristics were determined experimentally by measuring the execution time of the individual operations and them normalizing them to the equivalent of the time taken to perform one additional compare on a linked list. These costs are provided in Table 9. Here ***B*** is the number of bins and ***N*** is the current queue size.

Table 9. Comparison Equivalent for Optimization Calculations

| **Operation** | Cost (normalized to compare equivalent) |
|---|---|
| Head_over | 7 |
| Bin_over | 9 |
| bin_ceil_over | 10 |
| Skip_bin | 6 |
| New_bins | 4 |
| Search | 3 * B |
| Width | 100 |
| Resize | 6 * N |

## 6.5 Integration of SPQ in OTBSAF

The general SPQ was implemented in extended precision floating point or the C-language DOUBLE datatype for the event priority. The OTBSAF clock used a 32-bit unsigned integer. This required a re-implementation of the SPQ. In addition, the initial implementation was modeled to some extent on the YACSIM [Jump 1993] implementation which included the queue entry pointers as part of the activity or entity record. OTBSAF used a general queue structure based on an array and therefore simply passed a pointer to the requesting entity and its priority as parameters in the function call. The receiving function then stored these in the heap, shrinking and expanding as necessary. Thus, the SPQ was modified to accommodate the OTBSAF calling structure and the integer priorities. Another implication of the Integer clock/priority is a somewhat limited range of values. Analysis of OTBSAF's scheduler showed that indeed it would have multiple entries of the same priority. OTBSAF used a millisecond resolution clock, and used 67 millisecond update cycle. For any more than 67 entities, this would imply that more than one entity had to be updated each millisecond, thus multiple entities would have to be scheduled with the same priority. This was rarely the case in the floating point implementation. The test cases of the SPQ were prepared for the general case so they were all based on floating point. One desirable characteristic of floating point numbers is their range tends to be self-scaling, this is not the case for the integer implementation. Thus all the comparison test cases for the OTBSAF implementations had to be scaled to fit that environment.

The revised implementation thus incorporated the following changes:

- A revised hashing implementation for integers,
- Creation of queue entry pool for providing storage for queued entities and their priority,
- Implementation of first in first out pointers for equal priority entries, and
- Addition of a test for equality during queue insertion.

Figure 34 provides a diagram illustrating the changes to the SPQ required for OTBSAF, including pointers for the equality lists ***heq*** and ***teq***, for the *head* and *tail* of each equality list. Initially the first change is both the bin width and priorities are all integers, the bin width in Figure 34 is set to 10. The other changes are the handling of equal priorities. Priorities **19**, **33**, and **101** each illustrate a different case. Priority **33** is in the bin as part of the current year.

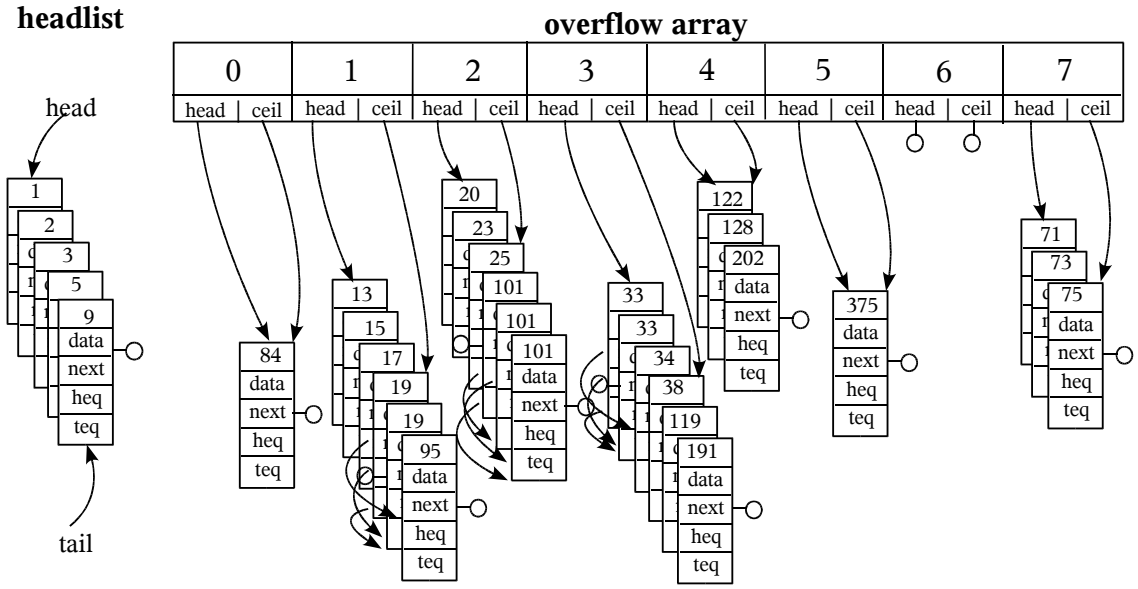


Figure 34. Revised SPQ for OTBSAF

In this case the ***next*** pointer points to the entry with priority **34** with the **h*eq***, and the ***teq*** pointers both pointing to the next priority **33** entry. The ***next*** pointer of the last priority **33** entry points to ***NULL*** or the empty location indicating the end of that equality list. During the insertion of any other entry into bin **3,** it is only compared to the priority **33** entry once and then skips to the priority **34** entry. In the case of priority **19** entry, it is the last entry of the current year so the ***ceil*** pointer points to the first priority **19** entry as the second entry is part of an equality list and is only referenced when either the first equal entry is removed or a delete operation of its data is required. Like the priority **33** entry, both the ***heq*** and ***teq*** pointers point to second member of the equality list. All entries to an equality list are made to the ***teq***, and all removals from the ***heq*** except the earlier mentioned deletes. Priority entry **101** illustrates the case when more than one additional entry is required. In this case, the ***heq*** points to the first equality entry, and ***teq*** points to the last equality entry and all intervening are tied to the successive entries by the ***next*** pointers.

During debugging of this implementation with OTBSAF, instances of more than 40 entries for the same priority were observed. Another advantage of this method of equality storage is that the whole equality list is manipulated as a single entry for queue resizing, and minimum priority searches etc. The implementation of the equality list as a FIFO list is important for repeatability purposes as established in theorem 5.1.

# CHAPTER 7 EXPERIMENTAL COMPARISON OF ALTERNATIVES

## 7.1 Experimental Configurations

OTBSAF is being used to demonstrate the concurrent model approach. The prototype developed has many of the characteristics postulated for the Concurrent Model approach. In addition, it models the elements of the simulation down to the individual entities as opposed to unit level of most constructive models. It also provides several scheduling, strategies, and queues that can be used for validating the concepts.

### 7.1.1 Concurrent SAF

To prepare OTBSAF to demonstrate the concepts, various modifications had to be made. The initial experiments were aimed at verifying that two separate simulations could be ran simultaneously in real-time and generate identical data. The first step was to incorporate the modifications recommended by SAIC for repeatable SAF. The next was to modify the scheduler to use release time, real-time scheduling, and to synchronize the simulation clock to the real-time clock. The last modification was made to the random number generator to provide the same random number to both simulations. This entailed setting up one simulator as a master and the second as a slave. The master generates the random numbers used both locally as well as transmitted to the slave to used for the

slaves calculations. Another approach on synchronizing the random numbers was just to use the same seed for both simulations and count on them to remain in sync for the simulation run. To generate two identical runs, a scenario was prepared and saved. A simulation run was set up by loading the same scenario in both simulators, initialized with the same random number seed and then synchronized the start of the simulation. Results from these initial runs indicated that the information required for setting up and synchronizing both simulations to generate identical data was much less than the amount of data generated by the simulations. The quantity of random numbers required was small in relation to the other data to set up the simulation, but required the master to lead the slave in execution. Further experiments would concentrate on using the common seed approach rather than a single random number source. These master slave runs used OTBSAF as Pocket SAFs for connivence as it took fewer computers and processes.

The first experiments were conducted with two modified Pocket SAFs. Two separate loggers for the Pocket SAFs were executed as separate processes on computer A with synchronization traffic transferred via the pipe from the master Pocket SAF which is set as (Exercise 1, Database 1) which separates the packets from those generated by the slave pocket SAF which is set as (Exercise 2, Database 2). The loggers reside on computer B where they monitor the Ethernet traffic generated by the simulations on computer A. Post simulation runs for the packets collected during each exercise were compared and they were found to have the same position at the same time for the entire simulation run.

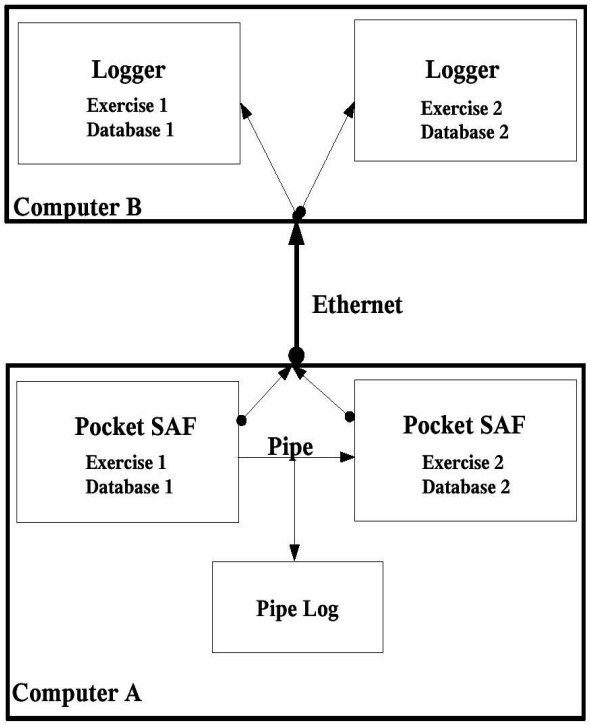



Figure 35: Concurrent SAF

### 7.1.2 Remote SAF Operator

The next step of validating the Concurrent Model approach was to introduce the Operator-in-the-loop into the experiments. This would provide two additional advantages. It would demonstrate the benefits and feasibility of using the Concurrent Model approach for a remote SAF operator application, as well as using a human-being for the DAE function. Using a human operator for the DAE function could satisfy the enhanced situational awareness application, as well as provide a basis for a knowledge base approach for automating the function.

The initial challenge was to determine the minimum data necessary to cause both simulations to generate the same output. Observing that the only independent source of

change to the simulation would be introduced by the Operator-in-the-Loop, the OTBSAF interface to the operator would be that source. This interface is provided by the SAFstation or GUI. The Pocket SAF mixes both simulation generated changes as well as operator initiated changes. This appeared to be a disadvantage as it could require more hardware platforms, however it was found that both a SAFstation and a SAFsim could be ran in a multitasking mode on the same hardware, although requiring more memory it could still serve to isolate the operator generated changes from the simulation generated changes while using a single hardware platform.

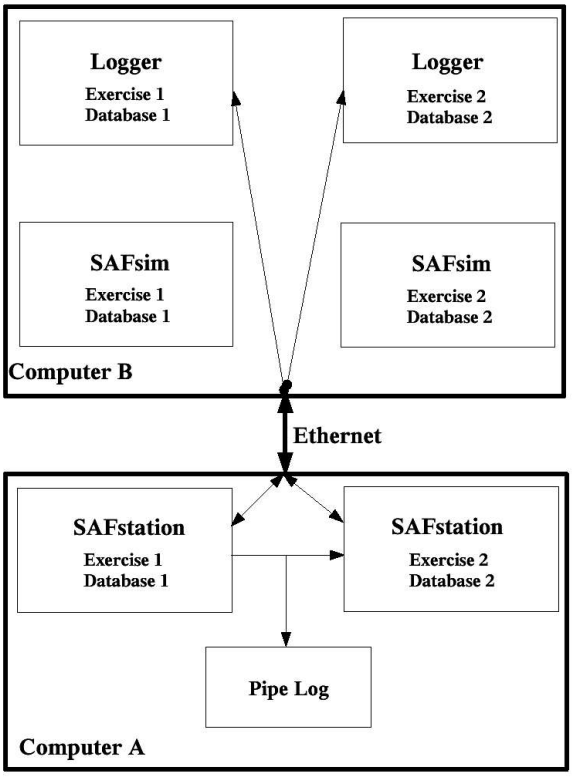



Figure 36: Remote SAF Operator

In the second set of experiments, the pocket SAFs were replaced by separate SAFstation and SAFsim processes. The SAFstations communicated via the DIS protocol across the Ethernet as a normal OTBSAF exercise. As provided by OTBSAF, separate exercises and databases allow separation of the simulations. For connivence, the concurrent SAFstations are executed on the same processor with the inter-process communication occurring via a pipe. The transactions occurring for each exercise are logged by the respective loggers. To compare the transactions of the exercises to the transactions between the concurrent simulations, the packets are converted to the same format.

### 7.1.3 SPQ used for Case Study

Initial performance comparisons were based on total user time over the execution of the model for a repeated simulation run. The results demonstrate the improvement gained from the changes to the queue implementation. Since the data listed indicates the wall clock execution time of the entire simulation, queuing overhead is just one component. Thus, a 21.8% reduction in wall clock time for the entire simulation by using SPQ rather than calendar queue is quite significant. This is due to the event processing time of simulation events remaining constant while the queuing overhead continues to be reduced. For this reason, and to further determine whether additional changes may be beneficial, the queue implementations were further instrumented as described below.

### 7.1.3.1 Queue Instrumentation.

For each path through the queue maintenance routines, overhead statistic counters were added. When this path included a loop structure, the loop count was included. Counts for paths with no loops were accumulated and reported for the total simulation. Paths with loops had their results reported upon exit from the loop. The results will be presented in Table 13. Figure 18, 19, and 47 provide the distributions in graphical form.

Table 10. Statistical Counter definitions

| Counter | Description |
|---|---|
| *Linear_head* | The number of insertions at the head of a linear list |
| *Index_head* | The number of insertions at the head of a bin found after calculating the index. |
| *New_headbin* | The number of insertions at the head of the queue for the calendar queue and also part of the index head count. |
| *Linear_tail* | The number of insertions at the tail of a linear list. |
| *Index_tail* | The number of insertions at the tail of a bin found after calculating the index. |
| *Index_empty* | The number of insertions into a previously empty bin. |
| *Linear_distrib* | The number of insertions in the interior of a linear list. The average number of elements into the list where the insertion took place. |
| *Index_distrib* | The number of insertions in the interior of a calendar bin. The average number of elements into the bin where the insertion took place. |
| *Get_head_linear* | The number of element removals from the head of the linear queue as the event is activated. |
| *Get_head_bin* | The number of element removals from the head of the calendar queue as the event is activated. |
| *Get_head_empty* | The number of times the get_head action empties a bin. |

#### 7.1.3.2 Statistic Counter Definitions.

To collect statistics counters were used in various paths. These counters are only present if selected at compile time. Each counter is described in Table 10. There are 6 counters related to head-of-queue operations, 2 related to tail-of-queue operations, 2 related to the distribution, and 1 indicating the first addition to a bin.

### 7.1.4 SPQ Evaluation with Statistical Distributions.

The first question the modeling tool designer or an advanced modeler would ask is how will the SPQ benefit a discrete event simulation application. In terms of execution overhead, SPQ can exhibit less overhead and be better behaved than other alternatives. The case study in Section 2.6.4, the design of the SPQ in Section 6.4, and the experimental results presented in Section 7.3.3 provided greater insight into the dynamics of Queue utilization. In this section, we provide the simulation designer with additional data to make an informed decision on priority queue management policies.

An important issue is understanding why the model exhibited a head-end skewed distribution. The RDMS model consisted of several sources of messages that were being transferred though the network to multiple destinations. This drove the nominal queue size as there was at least one event queued for each source for the life of the simulation. This basically made the model well-behaved from Calendar Queue length viewpoint, however, it had the hidden characteristic that short events were not introduced until after the size of the queue was stabilized. Shorter HOLDs were introduced as each message transferred through the network, and became more common as they competed for fewer

and fewer resources. Initially, they competed for one of 39 communication channels, which shared 3 I/O processors which communicated over one high speed parallel bus. Naturally the one bus could only keep up with all the traffic if it was utilized by each transaction for a short duration. This bus contention was the source for the smallest queue advance steps and since it only had the 3 I/O processors competing for the bus at any given time, at most there were only 3 extremely short holds present in the queue at a given instant. Since the bin width was computed by averaging over 25 samples, these short holds were swamped out by the longer ones already in the queue. This lead to a head list that had a lot of activity near the front of the queue. It also explains why the linear queue improved to an average insertion of 176 compares instead of the expected $N$=3000/2 =1500 compares based on the nominal Queue size. These observations are readily apparent once the modeler is alerted to the performance sensitivities of the event list structure. The total simulated time period is driven by long holds, for example the length of the call in a Personal Communications System (PCS). The most frequent steps are usually the much shorter events such as the signaling, and switching overhead events, or the intermixing of digital traffic with the standard voice traffic [Larocque 1996].

While Erickson[Erickson 2000] showed that the Calendar Queue could be optimized in the static case, Ronngren [Ronngren 1997] demonstrated that some distributions could expose the weaknesses of the Calendar Queue. The SPQ, while optimized for the case where 50% of the insertions occur near the head of the Queue, it was also developed to take advantage of the strengths of the Calendar Queue while minimizing the impact of it's weaknesses. Utilizing the event list insertion distributions described by Ronngren

[Ronngren 1997] we have compared the SPQ to the Calendar Queue. We have also included a distribution that approximates the input distribution of the RDMS model presented in Section 2.6.4.

## 7.2 Description of Scenarios

The scenarios used to test these concepts were various platoon movements about 10 minutes in duration. Results appeared equivalent for various locations of the map. The one used to collect the data for presentation here is indicated in Figure 37.

In this case, it shows the screen shots for two exercises operating with 2 different PO databases. In the title bar of the upper screen, it is identified as *Exercise 1* using *PO Database 1* on the *Ft. Knox* terrain database. The lower screen is identified as *Exercise 2* using *PO Database 2* again on the *Ft. Knox* terrain database. This is a platoon cross country march along the route indicated as the line route ***r1*** on the map background. In the upper right hand corner, there is the presentation of the real-time clock in Greenwich Meridian Time (GMT) to the nearest second. Since each screen is updated independently they could possibly vary by a second depending on the instant of capture. The symbol and ID that is partially cut off on the left side of each screen identifies the unit as an armor platoon 100A with tanks 100A11, 100A12, 100A13, and 100A14. The tanks are starting in a wedge formation at the left end of ***r1.*** This route crosses a natural barrier, a river, which the platoon will negotiate by using the bridge where the road crosses the river. In this case although the real-time clock is moving, the units are in position, but waiting for the order to move as indicated by the highlighted button just to the left of the

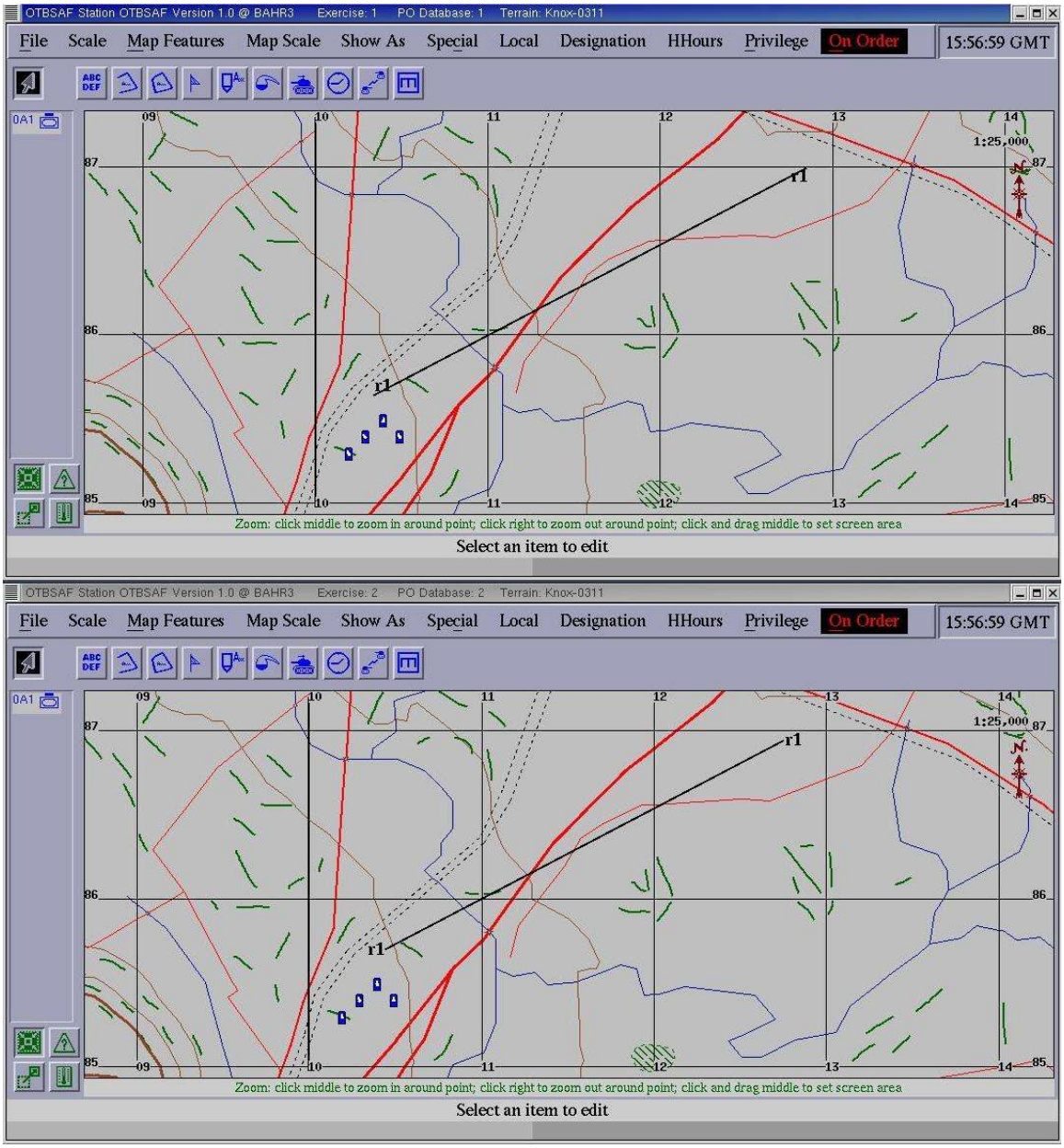


Figure 37. Screen Shot 1

clock display. This was the technique used to delay the start of the test run until everything is orientated for data capture. The above screens are actually the setup for the Remote SAF Operator, but are very similar for to those used for the first experiment the Concurrent SAF as well.

## 7.3 Presentation of Results

The next four sections presents the experimental results gathered in the evaluation of the Concurrent Model approach and priority queue data structures. First, we present the results of the Concurrent SAF experiment. Second, we present the results of the RSAFO experiments for two different scenarios. Next we present the experimental results for the generalized SPQ data structure in comparison to the Calendar Queue, and finally, we present the execution results of an integer adaptation of the SPQ to be used as a priority queue for OTBSAF.

### 7.3.1 Concurrent SAF

Two data sets were collected for the Concurrent SAF. They included the logs of each independent simulation, and a copy of the data transferred over the pipe between simulations. Since the goal was for both simulations to indicate the same exact behavior, the comparison of the data points was somewhat uninteresting upon success. With success, the difference in location for all the points measured was 0, as was the average, and standard deviation. This was true for all four vehicles for a run of about ten minutes and about 700 position updates. However, the results on the data transferred between the two simulations,was rather disappointing. While the total bytes transferred were reduced by 75%, the number of packets was the same order of magnitude as the number messages logged. In this case, the ratio was about 2 messages for every random number that was generated. This totaled about 400 messages. One observation on the data collected was that in many cases there was more than one message for the same vehicle

with the same timestamp and the same location. This number was not the same for both simulations, even though when the duplicates were eliminated, there were the same number of reported data points.

Table 11. Message Counts for Remote Operator

| Message Type | Local | Remote | Pipe |
|---|---|---|---|
| acknowledge | 24 | 36 | |
| aggregate_state | 26 | 26 | |
| entity_state | 4,059 | 3,517 | |
| po_delete_objects | 1 | 1 | 1 |
| po_line | 48 | 52 | |
| po_link | 26 | 26 | 1 |
| po_objects_present | 143 | 143 | |
| po_overlay | 102 | 102 | 6 |
| po_parametric_input | 116 | 116 | |
| po_parametric_input_holder | 51 | 52 | |
| po_point | 78 | 78 | 1 |
| po_simulator_present | 141 | 141 | |
| po_task | 1,150 | 1,153 | 15 |
| po_task_authorization | 15 | 15 | 1 |
| po_task_frame | 147 | 147 | 15 |
| po_task_state | 1,486 | 1,495 | 9 |
| po_unit | 202 | 202 | 6 |
| po_variable | 9,028 | 9,000 | |
| start_resume | 6 | 6 | 1 |
| stop_freeze | 12 | 12 | 4 |
| transmitter | 780 | 784 | |
| **TOTAL** | **17,641** | **17,104** | **60** |

### 7.3.2 Remote SAF Operator

The data collected for the RSAFO, was the data transmission logs of each independent simulation, a copy of all the data transferred over the pipe between the two simulations and a sequence of screen shots showing the SAF Operator display for the concurrent exercises. In this case, the data transferred over the network and through the pipes had the same format. This allowed a packet-to-packet comparison of each source. Table 11

shows the results of this comparison. Message types are the DIS204 message types [IEEE 1995] as translated from the message headers. The packets transmitted from the Local source are the same as standard OTBSAF without modification and thus providing the baseline for comparison. The Local column refers to the messages transferred as exercise one, and the Remote column refers to the messages transferred in exercise two. The Pipe column refers to the messages transferred from exercise one to exercise two over the pipe. The result of interest for dropout immunity in that the last message transferred through the pipe occurred at relative time=:01:35.683 of the total relative time=:12:54.806 of the exercise. Thus, for this scenario the pipe between the two generators only had to be available for the first 96 seconds. One problem we had was as reported in [Cheung 1994] with the DIS timestamp implementation. They do not seem to properly implemented for clock synchronization. We were able to identify common points in each data stream that we used for evaluation synchronization. We were able to adjust the timestamps for further comparisons. This did rule out latency experimentation at this time.

The key item of these results is the ratio of the total number of the messages transferred on the local network which was 17,641 in the first column versus the those transferred over the pipe which was 60 in the third column. This yields a packet reduction ratio of 294-fold. This is greater than two orders of magnitude improvement and is in the range envisioned for the Concurrent Model Approach. While there are some variations in the results that can be explored, in most cases the number of remote messages is very close to

the number of local messages, and the results as portrayed on the following series of screen shots is also indicative of the desired behavior by maintaining congruence.

Table 12. Message counts for Benchmark

| | Local | | | Pipe | | |
|---|---|---|---|---|---|---|
| **Message Type** | **Packets** | **Bytes/Tot** | **Bytes/Per** | **Packets** | **Bytes/Tot** | **Bytes/Per** |
| aggregate_state | 119 | 19,040 | 160 | | | |
| detonation | 404 | 41,334 | 102 | | | |
| entity_state | 8,788 | 1,546,688 | 176 | | | |
| fire | 404 | 38,784 | 96 | | | |
| po_delete_objects | 106 | 4,576 | 43 | | | |
| po_fire_parameters | 642 | 349,248 | 544 | 72 | 2,304 | 32 |
| po_line | 464 | 62,592 | 135 | 12 | 1,008 | 84 |
| po_link | 132 | 147,840 | 1,120 | 12 | 1,152 | 96 |
| po_objects_present | 5 | 5,848 | 1,170 | | | |
| po_overlay | 1,272 | 91,584 | 72 | 60 | 4,320 | 72 |
| po_parametric_input | 1,614 | 542,656 | 336 | 48 | 3,072 | 64 |
| po_parametric_input_holder | 1,032 | 66,048 | 64 | 48 | 2,496 | 52 |
| po_point | 573 | 57,300 | 100 | | | |
| po_simulator_present | 35 | 3,500 | 100 | | | |
| po_task | 9,870 | 1,250,304 | 127 | 204 | 13,872 | 68 |
| po_task_frame | 1,639 | 400,532 | 244 | 180 | 43,920 | 244 |
| po_task_state | 13,144 | 5,106,848 | 389 | 96 | 5,376 | 56 |
| po_unit | 2,640 | 1,710,720 | 648 | 120 | 16,320 | 136 |
| po_variable | 17,007 | 6,096,276 | 358 | | | |
| signal | 388 | 27,936 | 72 | | | |
| transmitter | 3,234 | 336,336 | 104 | | | |
| **TOTALS** | **63,512** | **17,905,990** | | **852** | **93,840** | |
| **RATIO (Local/Pipe)** | | | | **75** | **191** | |

Table 12 provides another snapshot into the relative performance of Concurrent Model approach. This was captured during the running of the benchmark for 10 platoons containing a total of 40 vehicles. The total execution time of this scenario was 4 minutes and 40 seconds. It was also a situation of intense object creation. In this case, the packet ratio fell to 74.54-fold although the bytes transmitted ratio was $\boldsymbol{B_r} = \boldsymbol{N/I}$ = 7.5 MB/93.8 KB = 190-fold. The remote pipe was not run due to the difficulties in getting the benchmark to run with independent SAFgui and SAFsim stations. These problems were

similar to those reported by Roberts [Roberts 1998]. A key difference in this scenario over the one reported in Table 11 is the unrealistically short move to engagement scenario employed to quickly stress the system as indicated by the inversely balanced ratio of task state to entity state messages.

Congruence for this scenario is demonstrated visually by the series of screen shots shown in Figure 37, Figure 38, and Figures 54 through Figure 61, that are described below and presented here and in the Appendix. Analysis of the data is presented in Figures 39 through Figure 46. We will continue the discussion of that data after the description of the screen shots.

The scenario started with the configuration depicted in Figure 37 with time advancing as shown by the GMT clock and the vehicles moving as indicated by their positions on the map display. In Figure 54, the vehicles are shown after they have arrived at the starting point of their route and re-orientate into their wedge formation. In Figure 55. they adjust their route to head toward the bridge. Figures 56, and Figure 57 show them continuing to the bridge. Figure 58 through Figure 38 are at a different scale that the earlier figures. Figure 58 shows them crossing the bridge and Figure 38 shows them reaching their destination at the far end of route ***r1***.

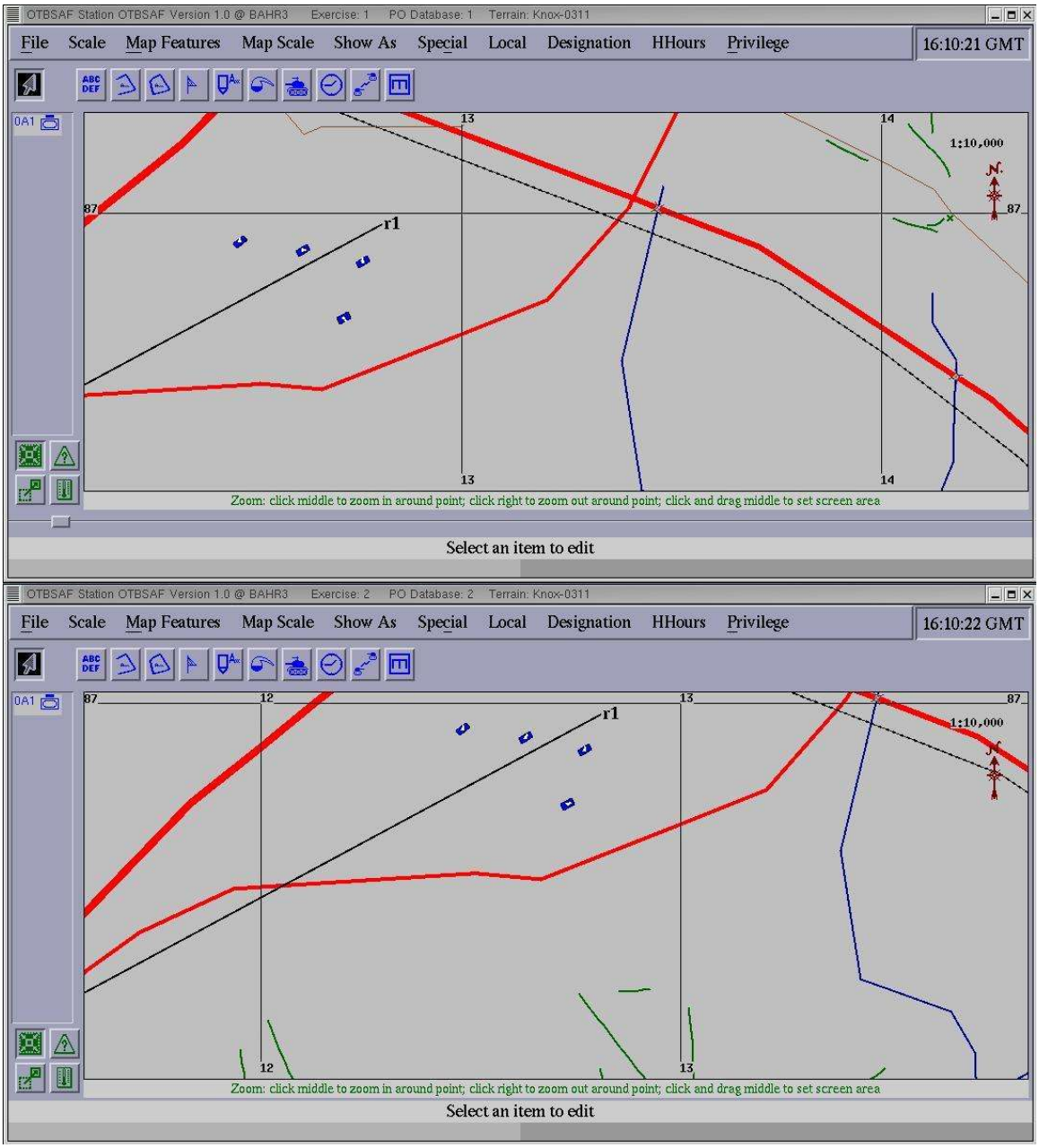


Figure 38. Screen Shot 10

Congruence is evaluated for this scenario from a behavioral viewpoint. We are define a behavioral congruence function $\Psi_B(E_L, E_R) = \sqrt{(x_L - x_R)^2 + (y_L - y_R)^2 + (z_L - z_R)^2}$ for each sample, for each vehicle. Since $\Psi_B$ is a magnitude, the $\Gamma_B$ truth function evaluates to $\Gamma_B$ is TRUE if and only if $\Psi_B \leq \delta$ and FALSE otherwise. For this discussion, $\delta = 25$

meters is selected as it represents an acceptable tolerance for the application. Figure 39 through Figure 42 are plots of $\Psi_B$ ***versus t*** for each vehicle displayed in the screen shots. Note that three of the vehicles had short periods of time where $\Psi_B$ exceeded $\delta$ with a worst case of about 47 meters. The average behavioral congruence $\Psi_B^{ave} = \frac{\sum_{i=1}^{n} \Psi_{Bi}}{n}$ where $n$ denotes the total number of samples for each vehicle was $\Psi_B^{ave}$ of A11 = 3.333, $\Psi_B^{ave}$ of A12 = 3.620, $\Psi_B^{ave}$ of A13 = 1.090, $\Psi_B^{ave}$ of A14 = 3.395 with the average for all vehicles combined yielding $\Psi_b^{ave}$= 2.859.

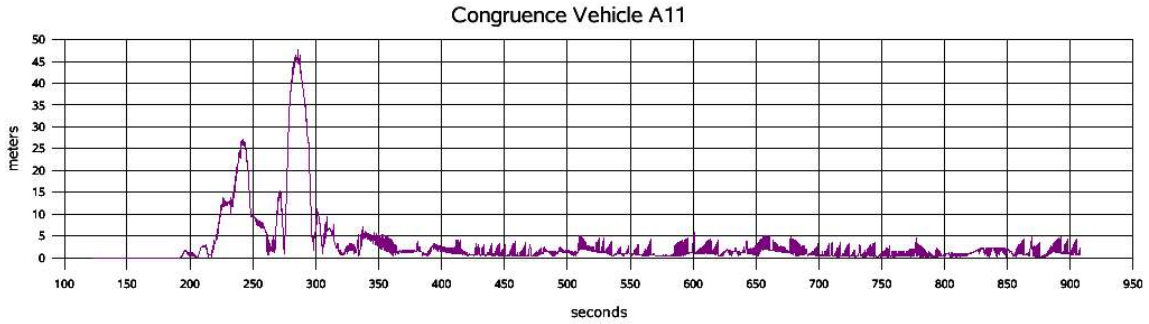



Figure 39. Behavioral Congruence for Vehicle A11

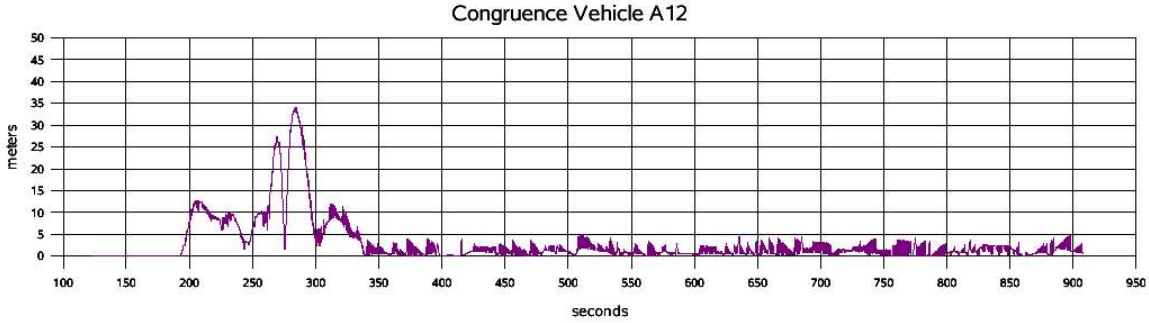



Figure 40. Behavioral Congruence for Vehicle A12

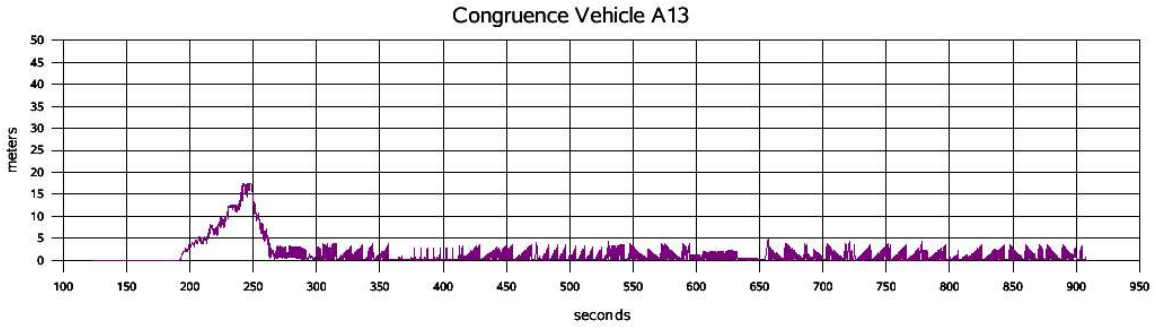



Figure 41. Behavioral Congruence for Vehicle A13

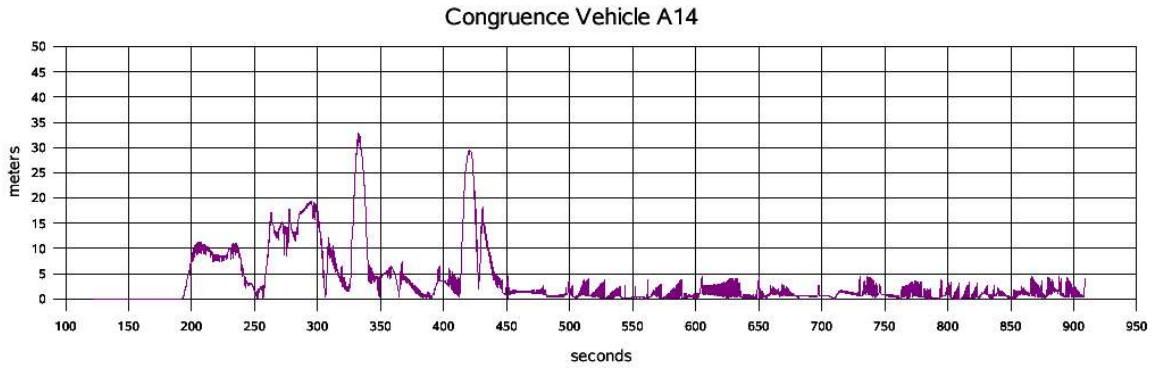



Figure 42. Behavioral Congruence for Vehicle A14

Figure 43 through Figure 46 provide the percentages of $\Psi_B$ ***vs*** $\delta$ for each vehicle. Note that over 90% of the samples exhibit a $\delta$ within 5 meters and less than 1% exceed 25 meters. The value of an acceptable $\delta$ would be established to meet the needs of the user. For this demonstration, 25 meters was established as it is close enough to select the correct vehicle. At $d=25$ meters there is no loss of generality because deviations below this are are approaching the limitations of position location systems. Only since the advent of GPS could we even get within 25 meters except with very precise surveying techniques. The most probable cause for the observed deviations, is the variation of the random numbers used by the two generators. Although seeded to start with the same

value, the effects of collisions on the network could cause a change in the sequence used by the individual models.

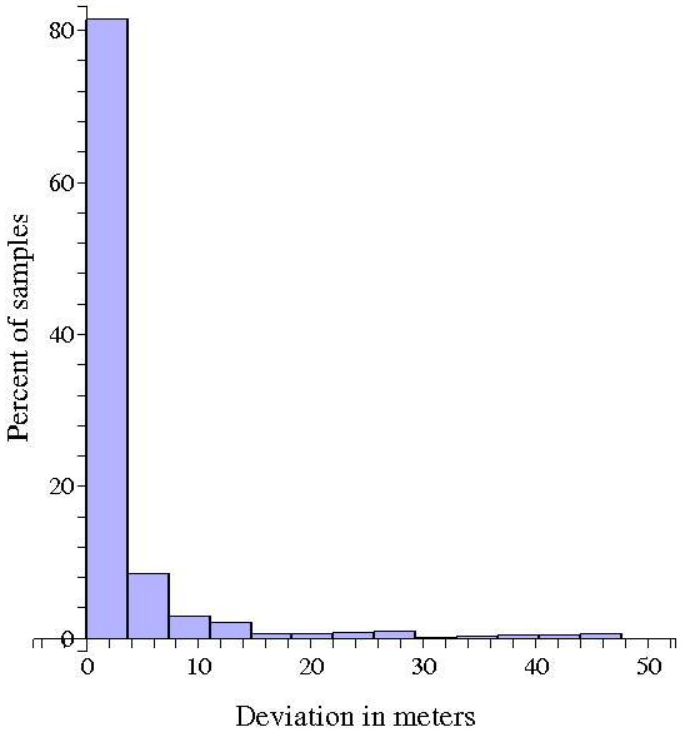



Figure 43. Congruence Histogram for Vehicle A11

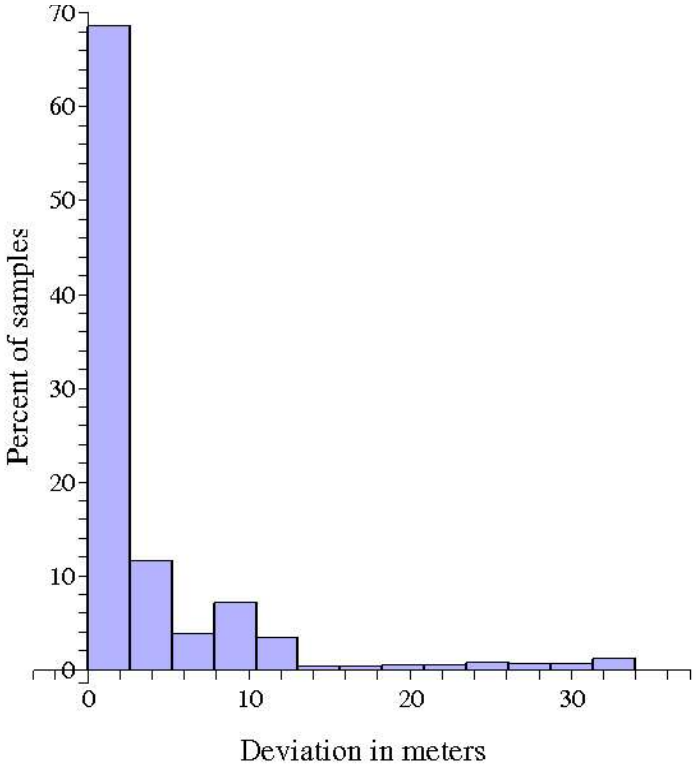


Figure 44. Congruence Histogram for Vehicle A12

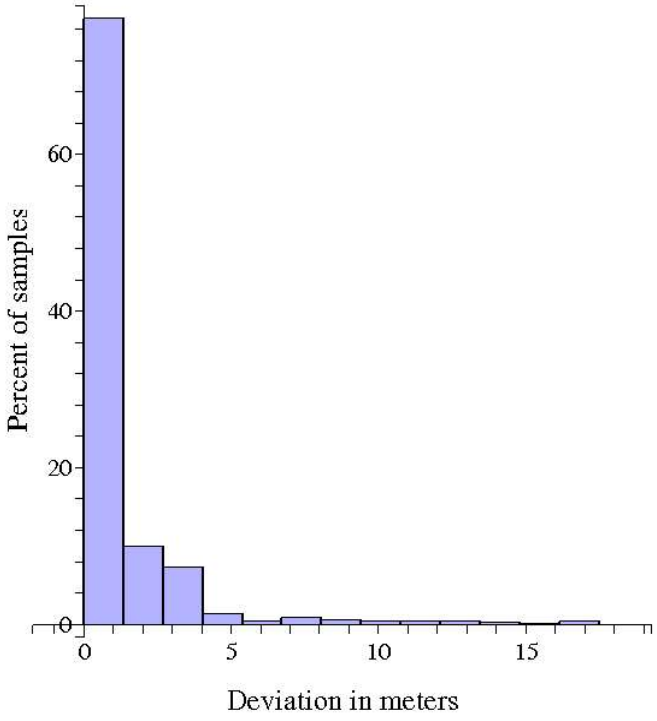


Figure 45. Congruence Histogram for Vehicle A13

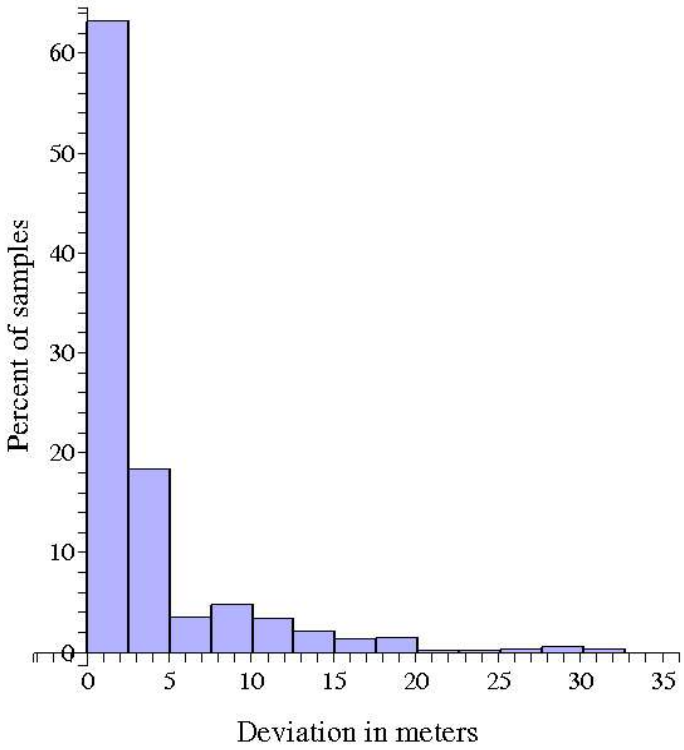


Figure 46. Congruence Histogram for Vehicle A14

### 7.3.3 SPQ

Since Congruence was shown to be maintained using the Concurrent Model approach, the research focus now shifts to processing optimization. To establish a basis of comparison, it is possible to discuss cost of operations in terms of equivalent linear queue sizes. For most models the developer of the model can roughly estimate the equivalent linear queue length for the steady state operation. We intend to show that this implementation has a relatively small penalty for short queue sizes and is of $\boldsymbol{O}(1)$ after the average length exceeds the cost of the indexing operation.

Table 13. Path Counts and Comparison Equivalents

| OPERATION | LINEAR | CALENDAR | SPQ |
|---|---|---|---|
| Linear_head | 4.52M | | 4.52M |
| Index_head | | 4.64M | 726K |
| newheadbin | | 4.52M | 527K |
| lineartail | 879 | | 303K |
| indextail | | 89.6K | 495K |
| indexempty | | | 263K |
| lineardistrib*average | 2.82M*176 | | 655K*2.31 |
| indexdistrib*average | | 2.62M*10.9 | 373K*2.76 |
| TOTAL INSERTIONS | 7.35M | 7.36M | 7.35M |
| Total Index | | 7.36M | 1.59M |
| Compare | 497.60M | 28.69M | 2.54M |
| getheadlinear | 7.35M | | 7.35M |
| getheadbin | | 7.35M | |
| TOTAL DELETIONS | 7.35M | 7.35M | 7.35M |
| findheaddist<br>ave | | 70<br>1.01 | 19.8K<br>2.02 |
| width samples | | | 108 |
| width change | | | 3 |
| resize change | | 10 | 4 |
| TOTAL Compare Equivalent | 536M | 155M | 53.3M |

As shown in Table 13, the calendar queue outperforms the linear queue for the RDMS simulation by reducing the the number of comparisons (Figure 18, and 19) required to insert into the body of the queue. The SPQ further improves performance by taking advantage of the low overhead of the linear queue for the `getheadlinear` operation. It further reduces the number of compares, as shown in Figure 47, to insert events in the body of the queue. Both the calendar and SPQ queues introduce additional costs for

monitoring and adjustment, but this is more than compensated for by the reduction in the number of compares required to order the queue.

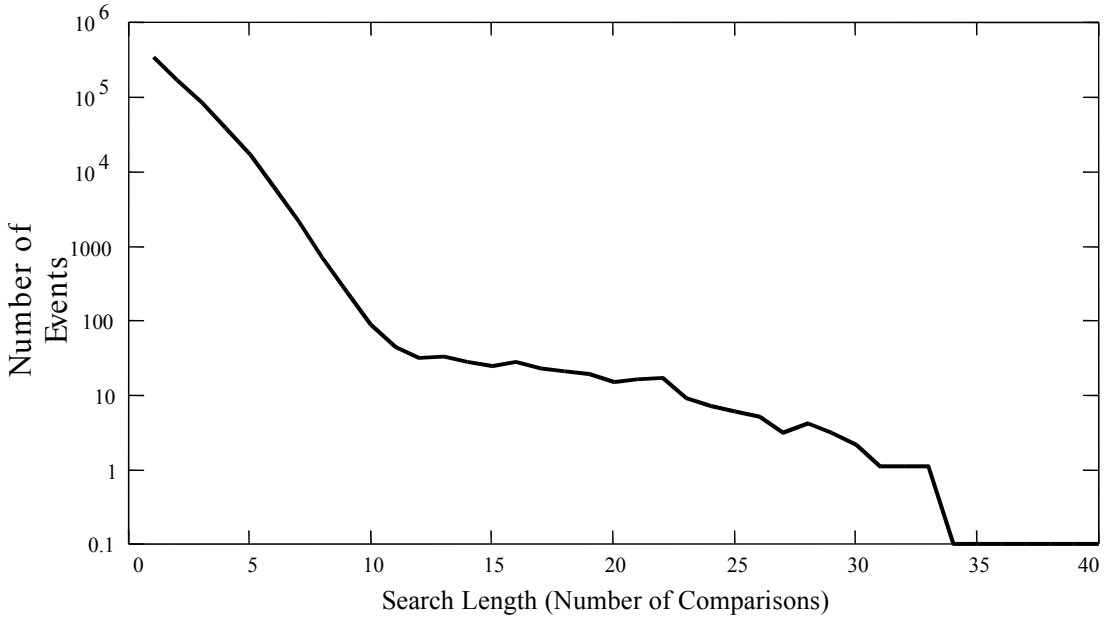



Figure 47. Search Length Distribution in a SPQ

Figure 48 provides the performance of the SPQ for 10 different distributions for queue sizes varying from 25 events to 500,000 events. Figure 49 provides the results for the Calendar Queue for the same distributions for the queue sizes of 25 events to 50,000 events. Note the substantial performance advantage for the SPQ with the RDMS distribution as expected, while it avoids the problems with the Camel distribution. Figures 50, and 51 show the performance for the Up/Down trials where each distribution is used to fill the Queue then successive `get head` operations are performed until the Queue is emptied. In this case, the SPQ outperforms the Calendar Queue because it doesn't resize until the queue either has large numbers of compares on insertion or skips large numbers of bins on the `get head` operation.

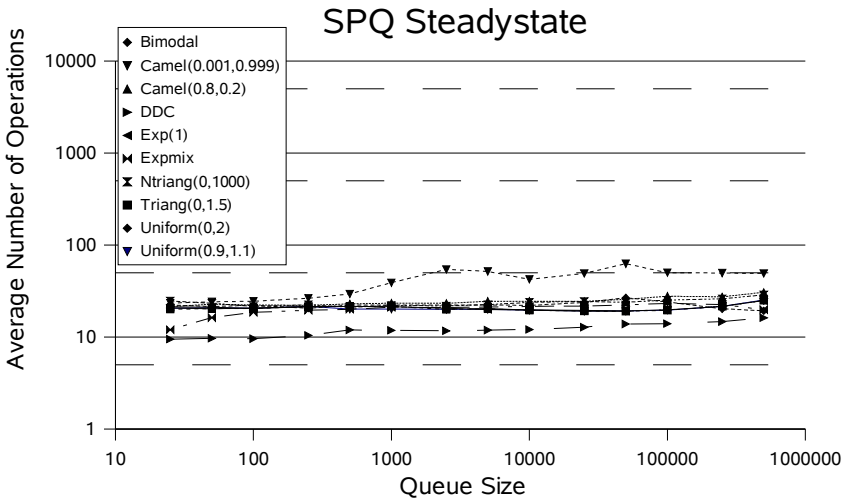



Figure 48. SPQ Performance for Classic Hold

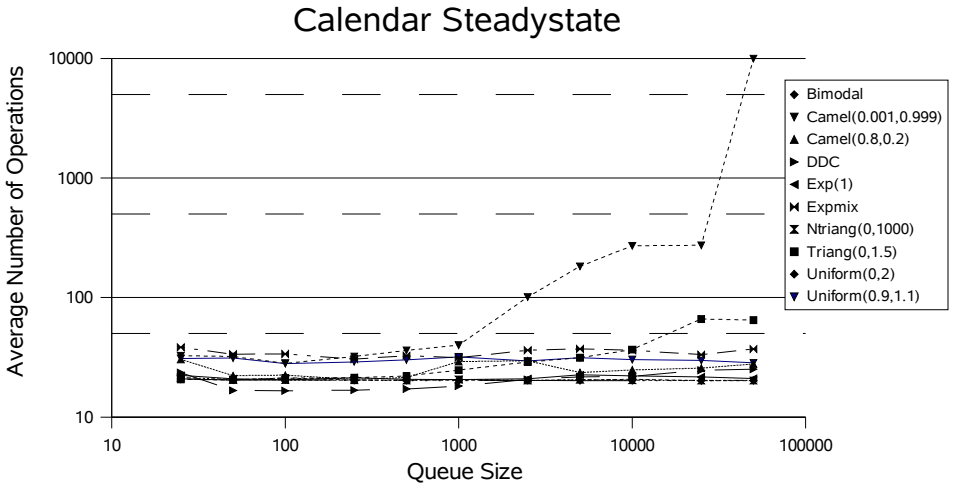



Figure 49. Calendar Queue for Classic Hold

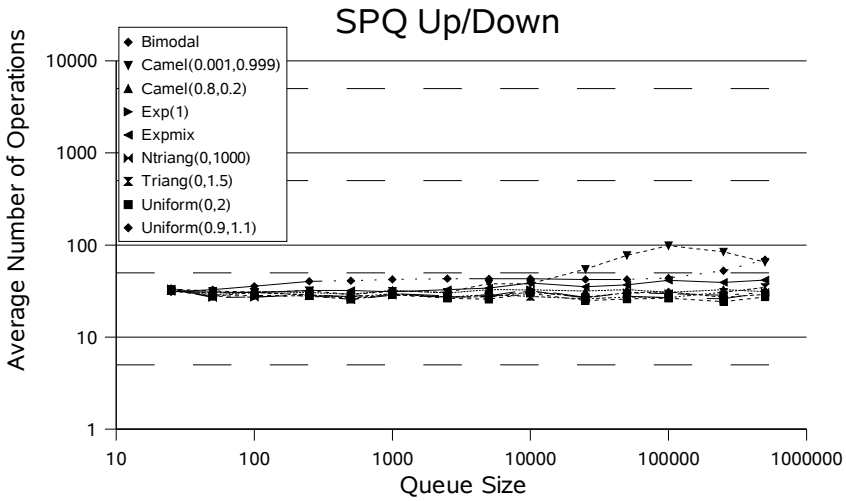



Figure 50. SPQ Performance In Up/Down

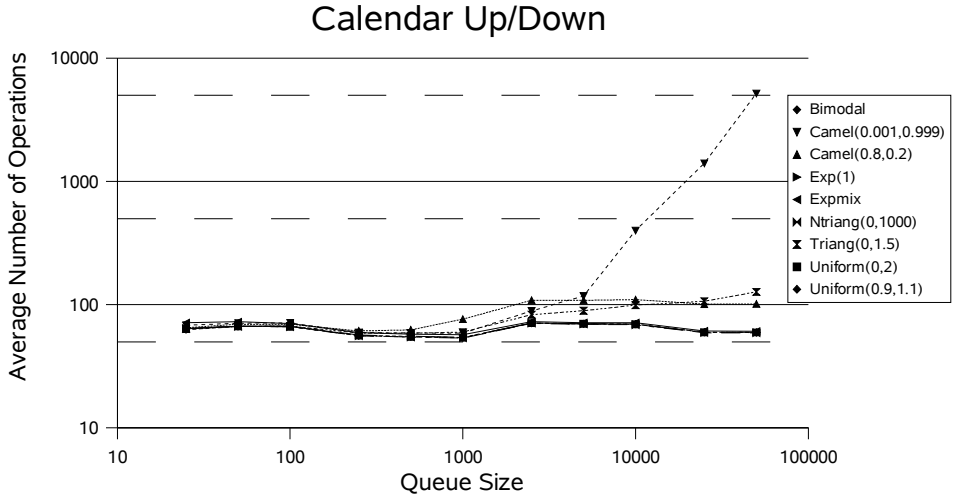


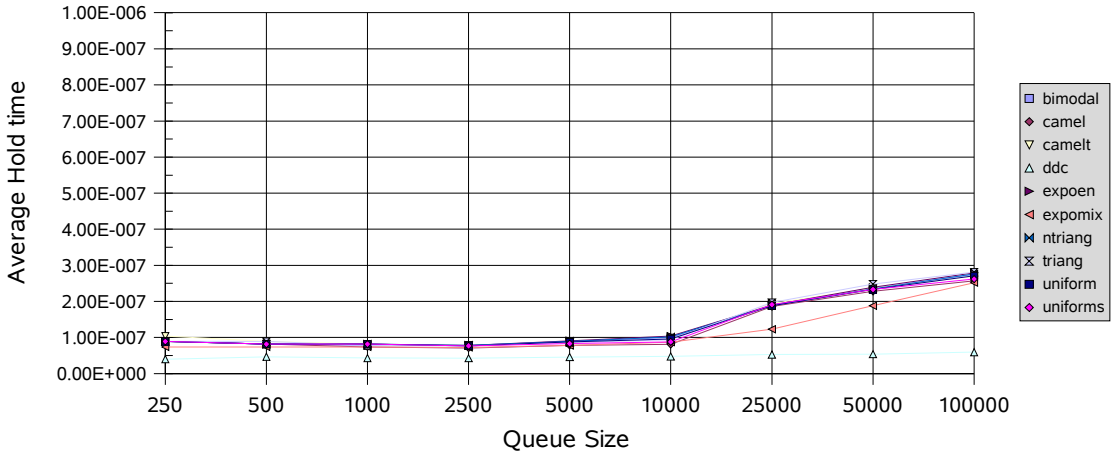

Figure 51. Calendar Queue Performance for Up/Down

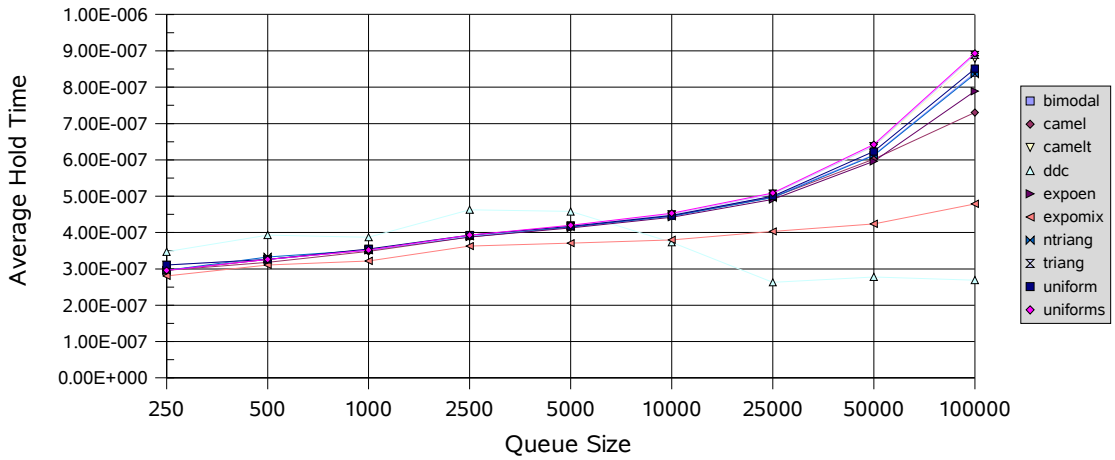
### 7.3.4 SPQ Integrated into OTBSAF

Two situations were evaluated to determine SPQ's benefits to OTBSAF and ultimately to the concurrent model approach. Initially, the OTBSAF priority queue was replaced with the SPQ and the resulting version of otherwise standard OTBSAF was benchmarked on the same computer originally used in Chapter 3 for Table 7. In this case, it was only ran on the fastest computer Bahr3. The same procedures were used as used previously for the earlier benchmarks, except in this case they were repeated for each version of OTBSAF 10 times. Again OTBSAF topped out at 70 platoons or 280 vehicles for all iterations. The modified OTBSAF topped out at 75 platoons or 300 vehicles for 3 of the iterations and 74 platoons or 296 vehicles for the other 7 iterations. We can interpret these results as an improvement in vehicle capacity of 6%.

To directly compare the performance of the two priority queues, a configuration similar to that used for the general SPQ study was established. Since both of these queues used integer priorities, the distributions had to be scaled and converted to integers. To verify the conversion, the revised SPQ results were compared to those presented in Figure 48, and found to be comparable. In this case, we measured the queue performance by preloading an array with all the priority increments prior to starting the get minimum insert the new entity at priority minimum plus the previously stored value. The goal was to minimize the amount of time to create the values used in the next cycle. The results are shown in Figures 52 and Figure 53, for the average hold times for nine different queue sizes from 250 entries to 100,000 entries for 10 different distributions. As shown, the SPQ was much faster than the conventional Priority Queue which required from 1.9

to 10.84 times as long with an average of 4.23 times as long over all the different queue sizes and distributions.



# CHAPTER 8 CONCLUSIONS

## 8.1 Summary

The Concurrent Model approach addressed in this dissertation is an advanced simulation technique developed to address the tradeoffs between continuous updated status of all elements and the limitations on the availability of that information due to the restrictions of bandwidth, delay, and dropouts. This dissertation has shown through analysis and experimental results that local models can generate the desired information with a much lower update rate than DIS-based. Furthermore, techniques for generating the required number of entities were postulated and explored. This technique made use of the Discrete Event Simulation approach and the basic data structure used for implementation of a Discrete Event Simulation, the event list or priority queue, was explored and significantly refined.

The SPQ was shown to be stable and give improved performance for a large class of distributions commonly used to compare event list implementations. It was incorporated into the OTBSAF scheduler where it again showed that it measurably improved the performance of the application. By using the operation cost minimizing approach to

bound the knowledge based heuristics it provides a superior near constant time performance priority queue algorithm.

The modifications to the initial SPQ algorithm for integration into OTBSAF demonstrated its effectiveness when applied to an integer restricted resolution clock environment. The addition of equality FIFO lists as part of the structure maintained the O(1) performance while providing strong clocking characteristics required for efficient repeatable simulations used in the Concurrent Model Approach.

## 8.2 Major Results

The concurrent model approach was developed and applied to the Remote SAF Operator application. To address scaleability and efficiency of the concurrent model approach, a superior priority queue, the SPQ, was developed as an event list for discrete event simulation. The Concurrent SAF experiments demonstrated that repeatability could be ensured by using a single random number source. However, it did not provide the performance benefits that were seen in the RSAFO.

### 8.2.1 Bandwidth Reduction

First, the theorems for Temporal Congruence $\Psi_T(t, E_R)$ and Behavioral Congruence $\Psi_B(E_L, E_R)$ were established and proven as the correctness factors of this approach. It was proven that the *Necessary and Sufficient Conditions for Behavioral Congruency are "* $\vec{S}(E_L)$ is behaviorally congruent to $\vec{S}(E_R)$ if the models are repeatable and they are

given the same inputs in the same order." It was also proven that *Temporal congruency* is obtained when "$\vec{S}(E_L)$ is temporally congruent to $\vec{S}(E_R)$ if the simulations are soft real-time synchronized to GPS time, and all changes are received ***s*** seconds before scheduled execution time, and are processed in the same order as transmitted. Where ***s*** is congruence setup time."

Next using the RSAFO application, it was demonstrated that the Concurrent Model Approach could provide the ratio $B_R$=***190***-fold improvement over the dead reckoning approach. This was demonstrated by modifying OTBSAF to serve as human interface and local and remote simulations of the concurrent model approach. Experimentally it was demonstrated that $B_R$ exceeded 100 by various factors depending on nature and length of the scenarios.

For the RSAFO application we chose a magnitude Behavioral Congruence function of $\Psi_B(E_L, E_R) = \sqrt{(x_L - x_R)^2 + (y_L - y_R)^2 + (z_L - z_R)^2}$ for each sample, for each vehicle. Since $\Psi_B$ is a magnitude, the $\Gamma_B$ truth function evaluates to $\Gamma_B$ is TRUE if and only if $\Psi_B \leq \delta$ and FALSE otherwise. The chosen deviation allowed was $\delta = 25$ meters. The computed average deviation was $\Psi_b^{ave} = 2.859$. The individual deviations were within 5 metes over 90% of the samples. And the models maintained congruence over 99% of the time.

Theorems for Bandwidth reduction, latency and dropout immunity were established and proven as the performance factors for this approach. It was proven that "The concurrent model approach provides *reduced bandwidth demand."* The demonstrated bandwidth reduction is more than adequate to enable feasible mixed live and virtual training with Embedded Simulation. It was proven that "The concurrent model approach provides a *latency hiding ratio* of $\frac{t_{prot}+t_{wait}}{T_{tot}}$." Latency hiding provides realism while using globally distributed participants in the exercise. It was proven that "The concurrent model approach provides *outage immunity ratio* of up to $\boldsymbol{B_R}$." Dropout immunity provides benefits for both training and enhanced situational awareness. For the RSAFO test scenario the scenario was totally immune to any outage that occurred after the first 96 seconds as no further data was required to complete the remote generation of the scenario.

### 8.2.2 Order(1) Priority Queue

Since since short HOLD time events can occur much more frequently in a simulation than long HOLD times, adaptive queue management techniques that capitalize on this characteristic can significantly reduce queue overhead. Adaptive techniques were successfully applied by developing the SPQ to allow nearly constant time performance for these distributions. An important characteristic is low-overhead sensing of queue performance, which in turn triggered the adaptive measures required to bring the queue back within the optimal range of operation. A key benefit of these results is that they

allow the prospective user to have confidence that the event queuing distribution will not drastically change the execution time of the simulation model.

SPQ performance whether analyzed from an operation count or a timing exercise, can show a reduction in overhead of greater than 50% in comparison to the calendar queue, and will perform no worse than the better of the linear queue or the calendar queue individually. Analysis of the SPQ shows that it will perform comparably to a linear queue for less than eight events and will exhibit nearly constant time performance for larger queue sizes. SPQ performance can be shown to be better than that expected for binary queues with less complexity in the primary paths of execution. The SPQ performs well not only for the distribution of the case study but also for the distributions commonly used to compare queue performance for the event list application.

The modified SPQ as applied to OTBSAF again demonstrated a performance improvement. It provided a 6% percent increase in the number of vehicles that could be simulated. It did this by improving the efficiency of the primary event list used in OTBSAF. In queue performance comparisons it provided execution time reductions of from 47.3% to 90.8% with a mean of 76.4% over the various test distributions. Thus, on the average the SPQ only took 23.6% of the execution time as compared to the priority queue supplied with OTBSAF to perform the overhead function of maintaining the execution order for the simulation.

## 8.3 Future Work

Follow on work would include developing automated techniques to convert an operators interactions with his equipment to commands and parameter changes to a SAF model that represents the operators system in the situation database. A suggested approach would be to adapt model based reasoning as a method of determining the factors for the congruence stream $\vec{G}_0$ for individual entities.

Remote SAF uses independent random number generators with a common seed, but postulates multiple simulators to provide the total view. It also postulates independent simulation rates at various sites. This would imply different random number sequences unless a different method of synchronization is employed. Another alternative would be to only use a random numbers for the SAF source and to use the most likely values for each of the clones, then issue corrections when the congruence delta exceeded predetermined thresholds. Development of application resilient techniques to the implementation of random characteristics in the concurrent model approach while maintaining congruence is essential for the extension of this approach to extended training scenarios. For situational awareness applications, random numbers should be replaced by fixed values. For training purposes a random host simulation with feedback correction for the clones or highly synchronized pseudo-random generators could be investigated to provide the desired divergence.

Additional work needs to be performed to implement latency compensation. The current DIS timestamp generation capabilities of OTBSAF are not sufficient for this purpose. Follow-on work for the SPQ includes tuning the heuristics for special cases where the dynamics of the distribution raise the average insertion cost appreciably.

# APPENDIX: EMBEDDED SIMULATION SYSTEMS

## System Environment

The primary purpose of a combat vehicle is enable the user/operator to perform his combat mission more effectively. While training certainly contributes to the combat effectiveness of the equipment crew, it is not the primary purpose of the combat system. Therefore , any additions, or modifications to the system for training purposes has to compete for space, power and weight against other possible mission enhancement modifications. However, if the training technology can be incorporated from a dual use viewpoint, that is mission enhancement as well as training enhancement, it becomes a very desirable candidate for system inclusion. For this reason, most of the proposed technologies are being addressed, not only from training enhancement purposes, but also from mission enhancement potential. In addition, anytime the training system can take advantage of existing combat equipment components, or the modifications will not seriously impact space, power, weight, and reliability the training role in itself provides increased effectiveness for the equipment. Yet, as a component of the system it must survive the same environment of the system and if it fails in must not decrease the reliability of the system for its primary mission.

## Advanced Technologies for Embedded Simulations

The technologies being developed as part of creating useful and powerful embedded simulations should address all significant issues integrating simulation technology with operational systems which include:

1. Sensor fusion – Sensor data, from electronic to human, will have to be fused together in the simulation environment to provide the warfighter with an intuitive view of the situation.

2. Visualization – Visualization of the different elements of the situation must be designed in such a way as to be ergonomically appropriate for the warfighter.

3. Human behavior representation – Much of the modeling and simulation will involve human behaviors. While Computer Generated Forces (CGF) technology is becoming mature, techniques to gather and learn the appropriate behaviors are still quite primitive. This will be necessary to quickly create the forces necessary to visualize the real world through the virtual world. Furthermore, inclusion of emotions, degraded states and variability of performance will also need to be incorporated.

4. Real-time simulation – In an operational context, real-time performance that is, bounded response time, will most definitely be a requirement.

5. Computer Networking – Much of the data exchanges will be done on an operational internet where communications between warfighters at all levels and in all directions will have to be made securely, reliably and fast. Enhancements to

current distributed simulation protocols such as HLA will have to be made to make them operation-ready.

6. Wireless Communications - The inter-unit communication will be performed within the constraints of wireless protocols.

## Advanced Technologies for Embedded Simulations Current Status

Over the period of October of 1996 through September of 2001 Headquarters United States Army Simulation, Training, and Instrumentation Command (STRICOM) concluded a research program addressing the technologies for embedded simulation called the Inter-Vehicle Embedded Simulation Technology Science and Technology Objective (INVEST-STO) [Bahr 1997A] [Bahr 1997C] [Bahr 1998] [Bahr 2002]. This Science and Technology Objective investigated the following technologies with the listed results.

All of the functions of training case 1 have been demonstrated on multiple combat vehicles that include the Marine LAV [Riley 2000], the Army's Abrams Tank [Klingensmith 1998] and Bradley Fighting Vehicle. In addition to the standalone capabilities they have also been networked together with components [Pollock 1999] of the Close Combat Tactical Training (CCTT) environment and used for collective training. The technologies required for the merge of live and virtual simulations have been studied but still need considerable development. In most cases concepts have been

demonstrated, but not at a level sufficient for engineering development. Individual areas are discussed below:

a. Geometric pairing: A training/operational testing instrumentation system based on GPS Interferometry has been developed and in use by the National Guard. This geometric paring system was developed by SRI.

b. Aim point determination: As Shiavone reports [Schiavone 2000] [Dolezal 1998] the concept has been demonstrated but further analysis of the targeting output of sensor systems is required. This will require collection of targeting information from actual combat systems.

c. High definition terrain database**:** The JFTB has demonstrated the collection of 1 meter resolution databases with processing completed within 24 hours of the flights for data collection. The integration of this information into Simulation databases still needs to be investigated. With the inclusion of onboard sensors in modern combat vehicles we need to develop technologies to integrate in near real-time updated information with the prepared databases.

d. Live Virtual Terrain Registration: Gelenbe *et al* [Gelenbe 2000] have demonstrated a method of registering a virtual scene with a a live view to the level necessary to allow virtual targets to inserted in the live view with sufficient resolution for training. Better databases could improve this technique, but this technique does not require the sophisticated instrumentation of previously demonstrated techniques.

e. Communication reduction techniques: Progress has been demonstrated on the Concurrent Player Model approach [Bahr 1996] as reported by McHale and Braudaway in [McHale 1998] [Ourston 1998]. Current work has demonstrated two independent platforms maintaining duplicate scenarios with very low synchronization costs. This is expected to be demonstrated by providing for remote SAF operation within the next year. Henninger *et al* [Henninger 2000] have reported techniques for improving models to be used for this purpose.

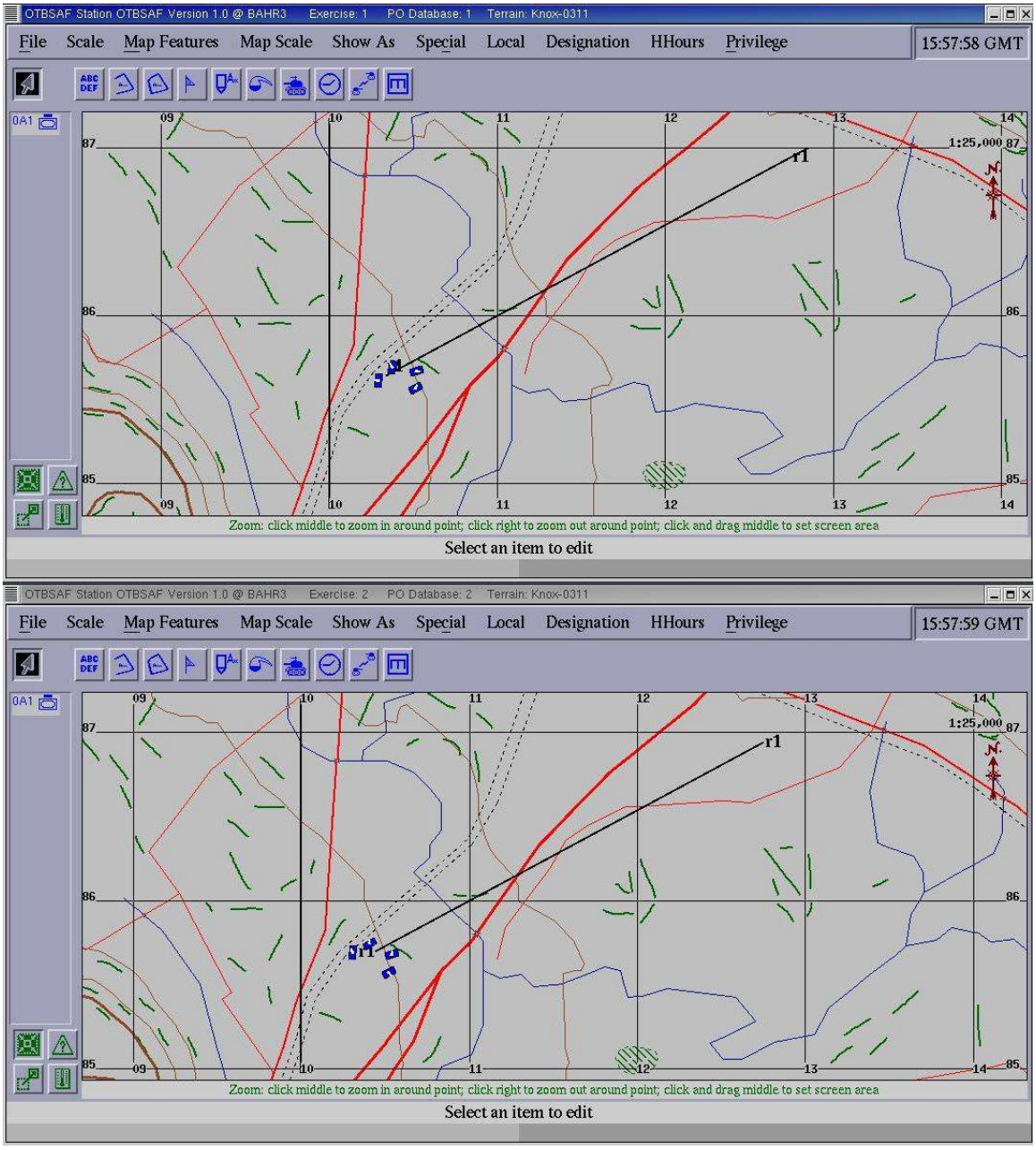


Figure 54. Screen Shot 2

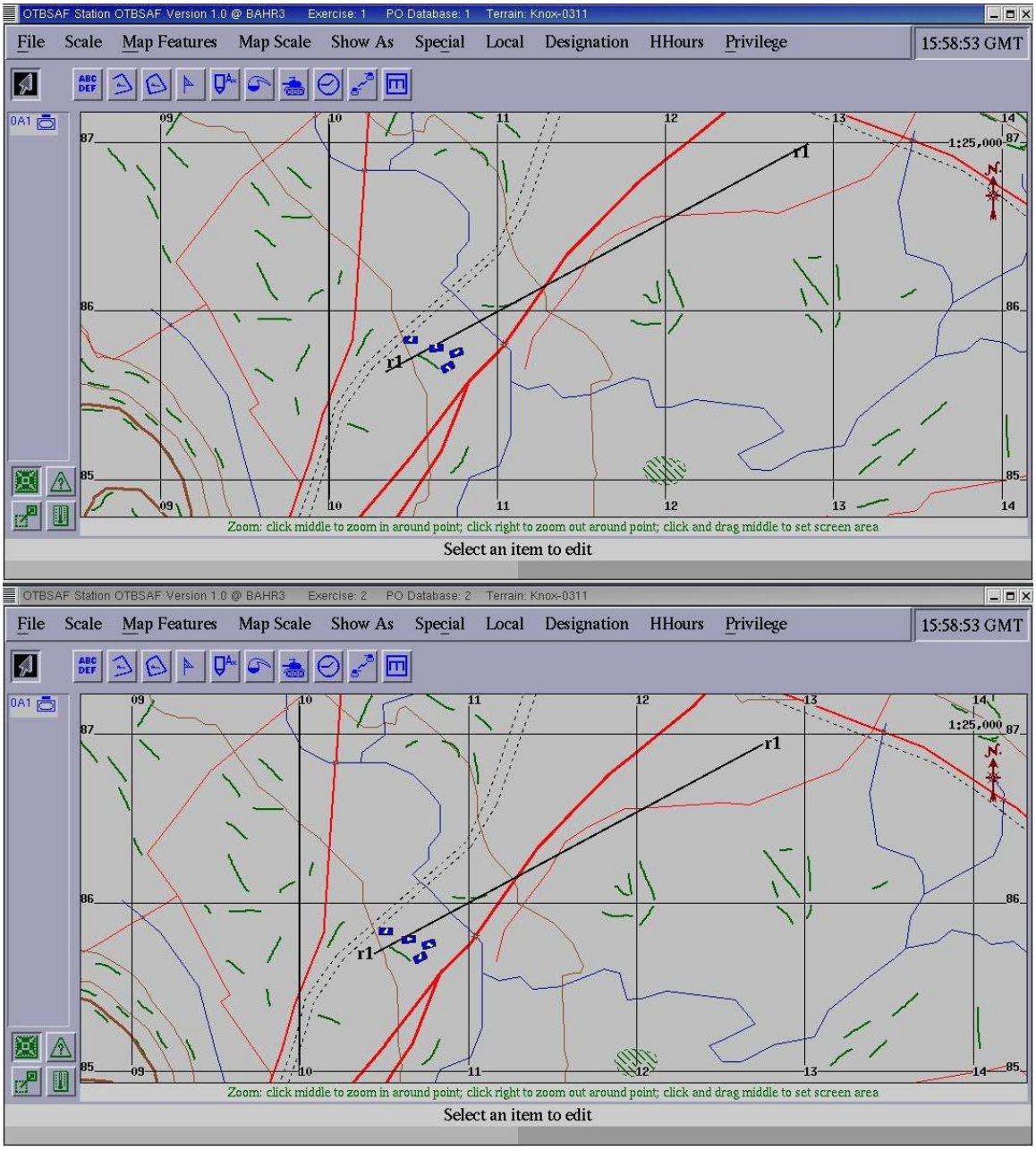


Figure 55. Screen Shot 3

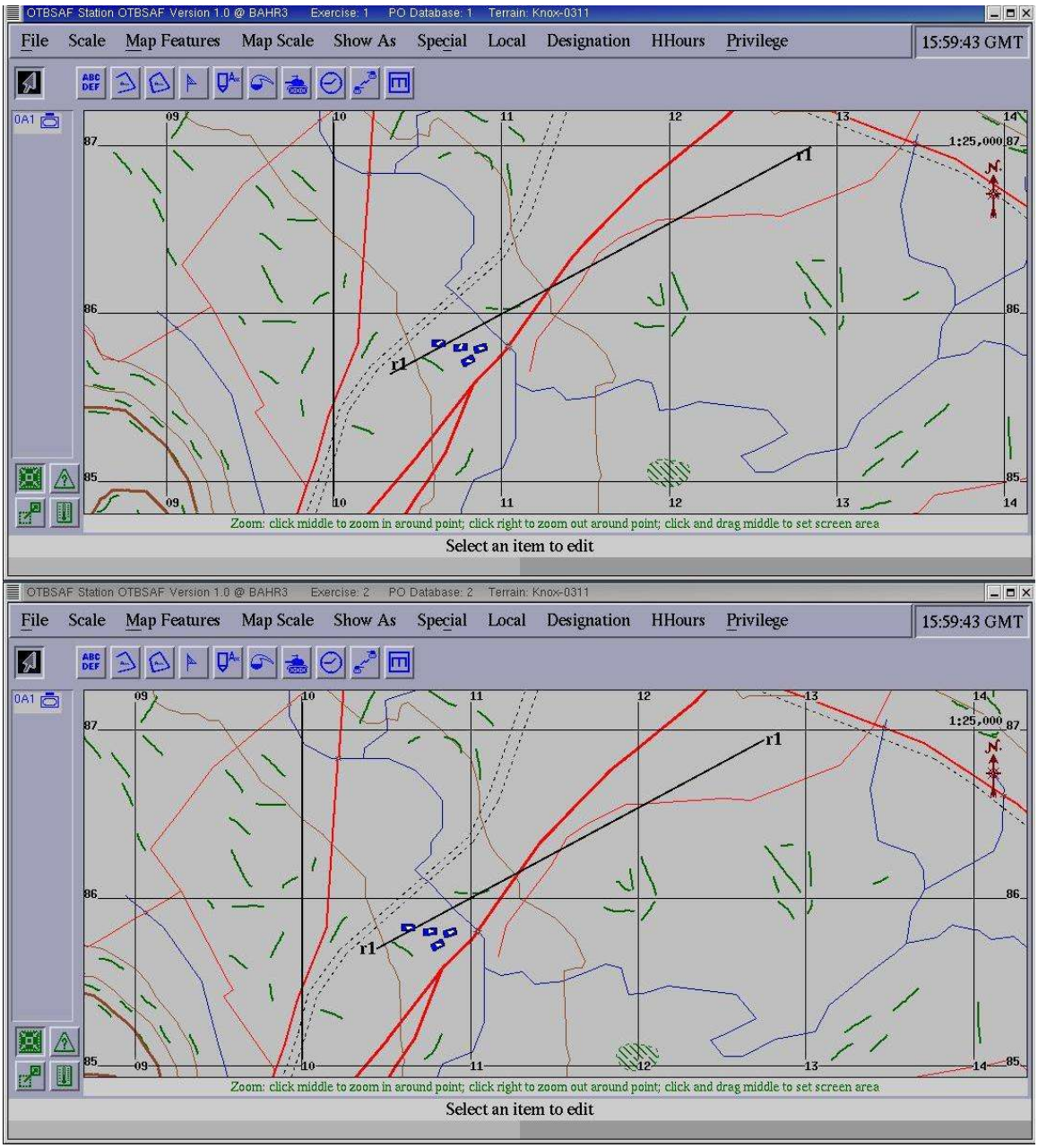


Figure 56. Screen Shot 4

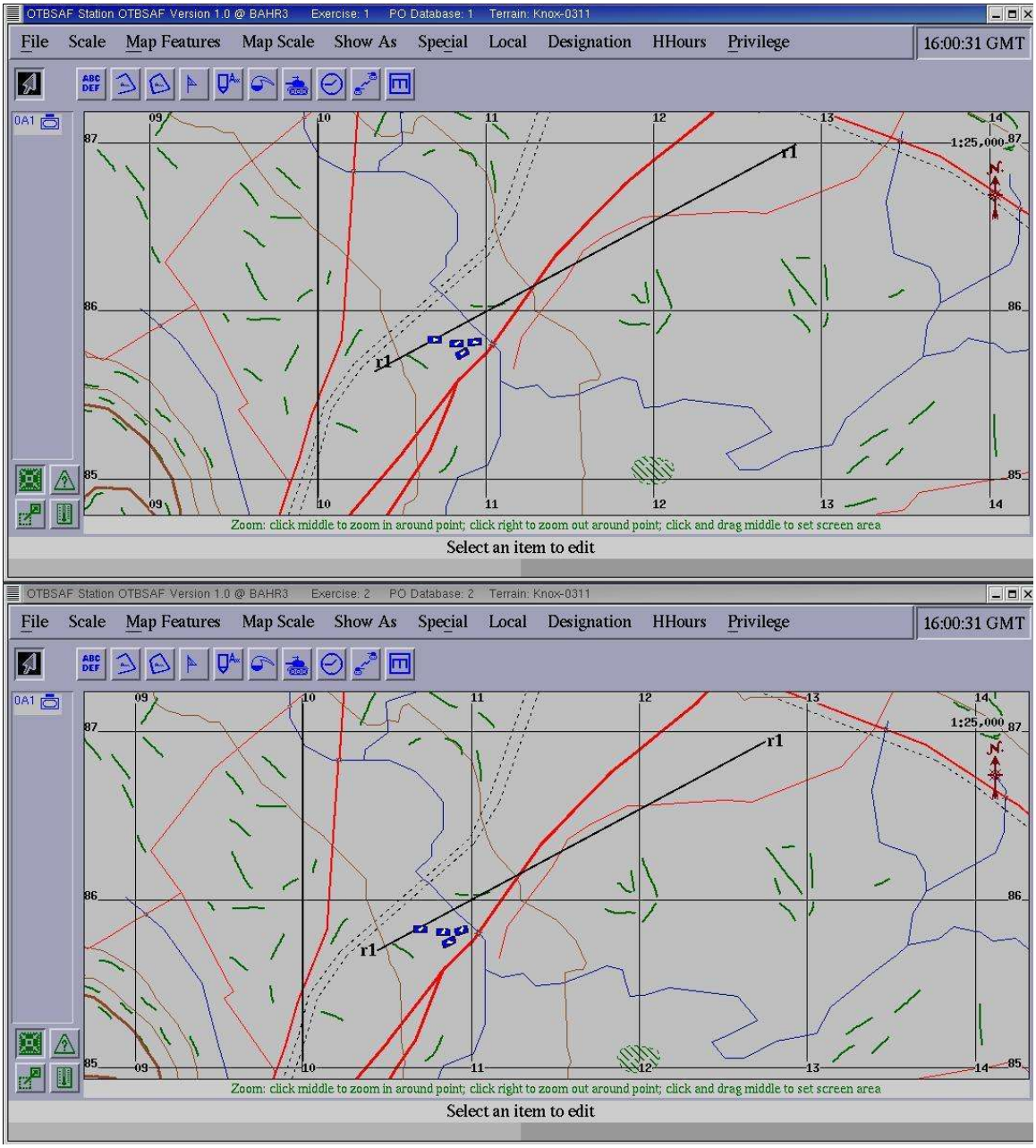


Figure 57. Screen Shot 5

Figure 58. Screen Shot 6

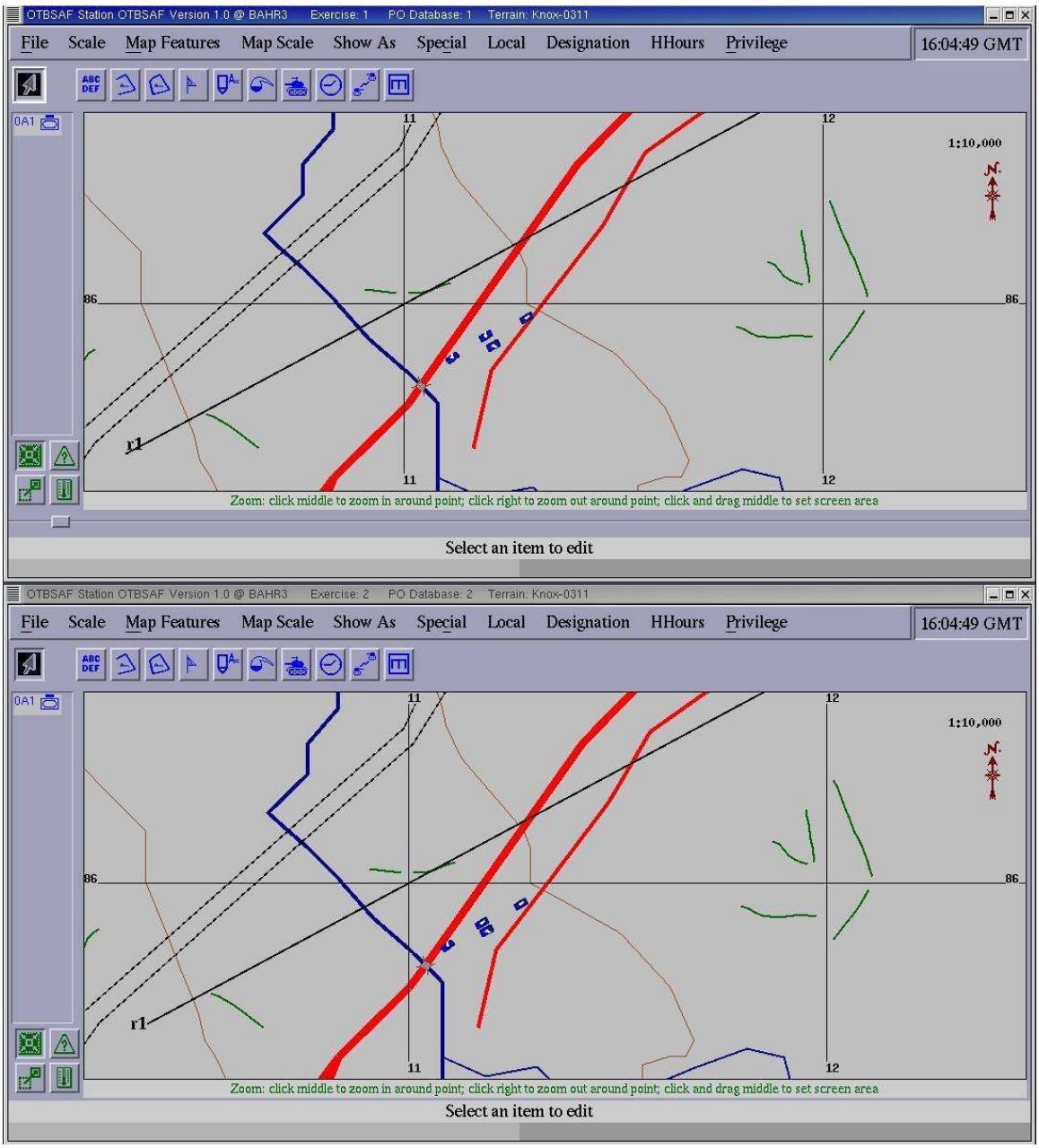


Figure 59. Screen Shot 7

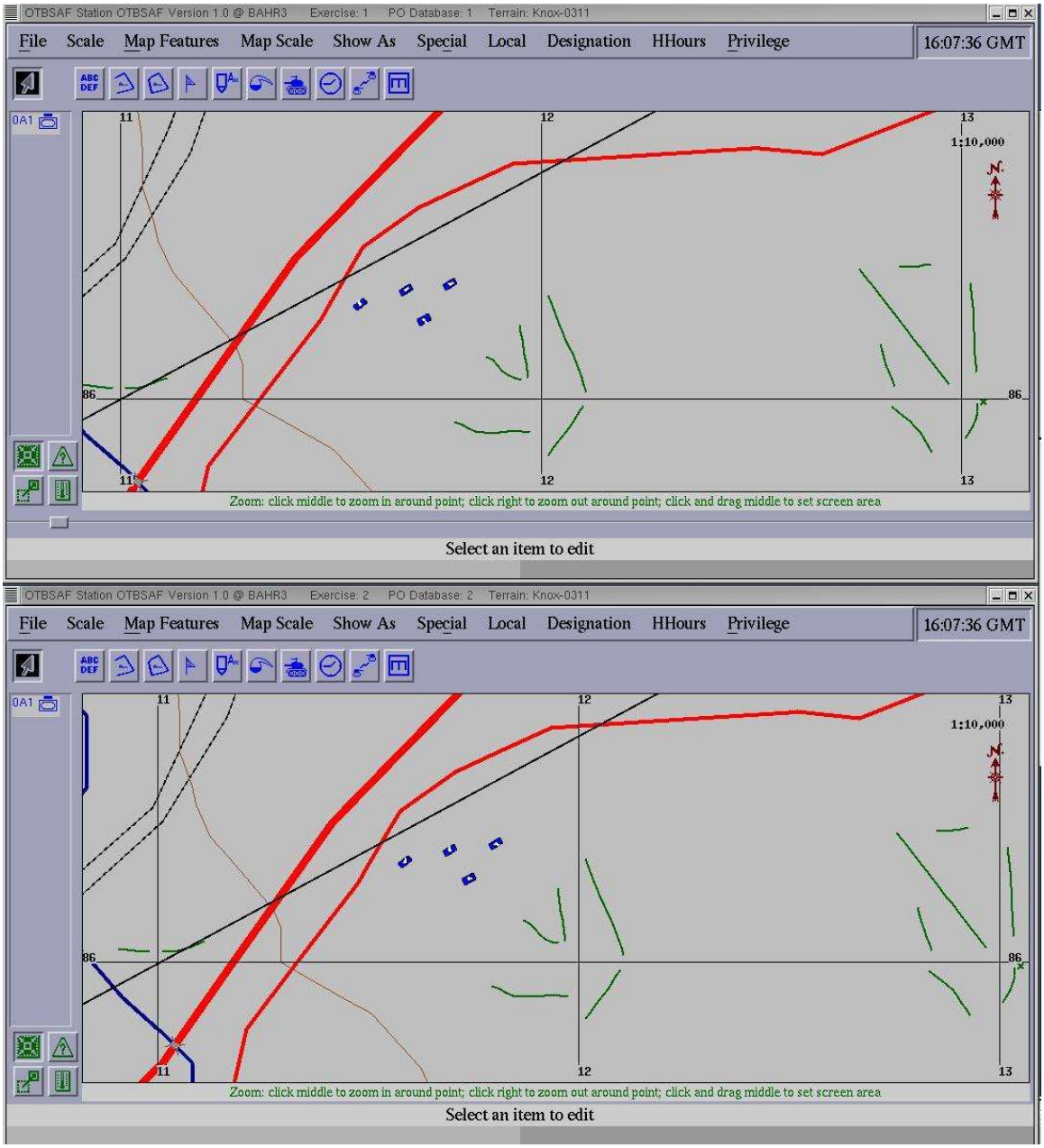


Figure 60. Screen Shot 8

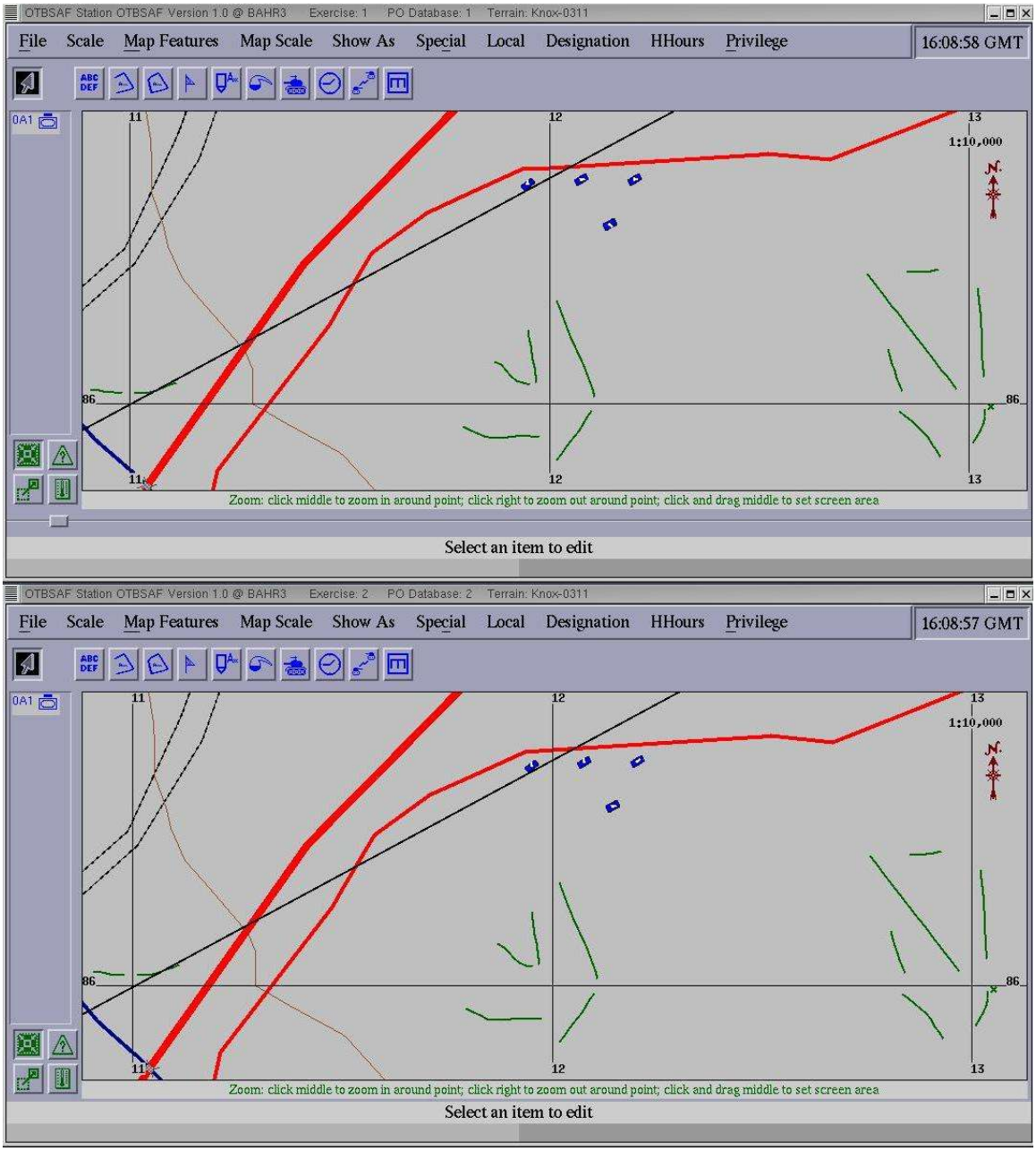


Figure 61. Screen Shot 9

# LIST OF REFERENCES


[Abate 1999] C. Abate, H. Bahr, "Embedded Simulation," *Armor Magazine*, June-July 1999.

[Abbott 1992] R. Abbott, H. Garcia-Molina, "Scheduling Real-time Transactions: A Performance Evaluation," *ACM Transactions on Database Systems*, September 1992.

[Ahn 1999] J. Ahn and S. Oh, " Dynamic Calendar Queue," in *Thirty-Second Annual Simulation Symposium*, April 1999.

[Bahr 1994A] H. A. Bahr, "Combined Event and Process Simulation Model of a Distributed Data Collection System," in *Conference Record of IEEE Southcon Conference*, 1994.

[Bahr 1994B] H. A. Bahr, *Distribution Adaptive Priority Queue Algorithm For Discrete Event Simulation*, Masters Thesis, University of Central Florida, Orlando, FL, 1994.

[Bahr 1996] H. A. Bahr and R. F. DeMara, "A Concurrent Model Approach to Scaleable Distributed Interactive Simulation," in *Proceedings of 15th Workshop on Distributed Interactive Simulation*, Fall 1996.

[Bahr 1997A] H. A. Bahr, "Embedded Simulation for Ground Vehicles," in *Proceedings of Simulation Interoperability Workshop*, Spring 1997.

[Bahr 1997C] H. A. Bahr, C. Abate, J. Collins, "Embedded Simulation for Army Ground Combat Vehicles," in *19th I/ITSEC Conference Proceedings*, December 1997.

[Bahr 1998] H. A. Bahr and C. Abate, "Embedded Simulation," in *Proceedings of 20th I/ITSEC Conference*, Fall 1998.

[Bahr 2002] H. A. Bahr and G. Holifield, "Embedded Simulation: INVEST-STO and Beyond," in *Proceedings of 1st SAWMAS Workshop*, October 2002.

[Bahr 2004] H. A. Bahr and R. F. DeMara, "Smart Priority Queue Algorithms for Self-Optimizing Event Storage," *Simulation Modeling Practice and Theory*, March 2004.

[Baker 1999] T. Baker, "Time Management in Distributed Simulation Models," in *The Simulation Technology and Training (SimTecT99) Conference*, March 1999.

[Balarin 1997] F. Balarin, A. Sangiovanni-Vincentelli, "Schedule Validation for Embedded Reactive Real-Time Systems," in *Design Automation Conference 97*, 1997.

[Bassiouni 1997] Bassiouni, M. A., etal, "Performance and reliability analysis of relevance filtering for scalable DIS," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 1997.

[Brown 1988] R. Brown, "Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem," *Communications of the ACM*, October 1988.

[Carothers 1997] C. Carothers, R. Fujimoto, R. Weatherly, A. Wilson, "Design and Implementation of HLA Time Management in the RTI Version F.0," in *Proceedings of the 1997 Winter Simulation Conference*, 1997.

[CBO 2003] Congressional Budget Office, "The Army's Bandwidth Bottleneck," Congress of United States, CBO Study, 2003.

[Cheung 1994] Cheung, S., Loper, M., "Synchronizing simulations in distributed interactive simulation," in *Proceedings of the 26th conference on Winter simulation*, 1994.

[Chou 1994] P. Chou, G. Borriello, "Software Scheduling in the Co-Synthesis of Reactive Real-Time Systems," in *Design Automation Conference*, 1994.

[Collins 2000] R. Collins etal, "System for Video Surveillance and Monitoring," Robotics InstituteCarnegie Mellon University, Tech Report, 2000.

[Dahman 1996] J. Dahman and R. Lutz, "HLA Time-Management and Distributed Interactive Simulation," in *Proceedings of 14th Workshop on Distributed Interactive Simulation*, Spring 1996.

[Dolezal 1998] M. Dolezal, "Correlating Live Aimpoints and Virtual Targets," in *SISO Simulation Interoperability Workshop*, Spring 1998.

[Emshoff 1970] J. R. Emshoff, Jr., and R. L. Sisson, "*Design and Use of Computer Simulation Models*," New York, MacMillan, 1970.

[Erickson 2000] K.E. Erickson, R.E. Lander, and A. LaMarca, "Optimizing Static Calendar Queues," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, July 2000.

[Evans 1967] G. W. Evans, G. F. Wallace, and G. L. Sutherland, "*Simulation Using Digital Computers*," Englewood Cliffs, NJ, Prentice-Hall, 1967.

[Fujimota 1996] R. Fujimota and R. Weatherly, "HLA Time Management and DIS," in *Proceedings of 14th Workshop on Distributed Interactive Simulation*, Spring 1996.

[Fujimoto 1990] R. M. Fujimoto, "Parallel Discrete Event Simulation," *Communications of the ACM*, October 1990.

[Fujimoto 1996] R. Fujimoto and R. Weatherly, "HLA Time Management and DIS," in *Proceedings of 14th Workshop on Distributed Interactive Simulation*, Spring 1996.

[Fujimoto 1998] R. Fujimoto, "Time Management in the High Level Architecture," *Simulation*, December 1998.

[Fujimoto 1999] R.M.Fujimoto, "Parallel and Distributed Simulation," in *1999 Winter Simulation Conference*, December 1999.

[Fullford 96] Fullford, Deborah A., "Distributed interactive simulation: its past, present, and future," in *Proceedings of the 28th Winter Simulation Conference*, December 1996.

[Gelenbe 2000] E. Gelenbe, K. Hussain, B. Foss, N. Lobo, H. A. Bahr, "Simulation Driven Virtual Objects in Real Scenes," in *22nd I/ITSEC Conference Proceedings*, December 2000.

[George 1997] L. George, P. Minet, "A FIFO worst case analysis for a hard real-time distributed problem with consistency constraints," in *International Conference on Distributed Computing Systems*, May 1997.

[Gerber 2001] W. J. Gerber, "Real-Time Synchronization of Behavioral Models With Human Performance in a Simulation," Ph.D. Dissertation, University of Cental Florida, Orlando, Florida, 2001.

[Goblick 1996] T. J. Goblick, et al, "Technical Requirements and Feasibility Analysis of 21st Century Live Play Simulation," Massachusetts Institute of Technology, Lincoln Laboratory Report ADS-1, 1996.

[Gonnet 1976] G. H. Gonnet, "Heaps Applied to Event Driven Mechanisms," *Communications of the ACM*, July 1976.

[Gonzalez 1998] A. Gonzalez, R. DeMara and M. Georgiopoulous, "Vehicle Model Generation and Optimization for Embedded Simulation," in *Proceedings of Simulation Interoperability Workshop*, Spring 1998.

[Henninger 1998] A. Henninger, W. Gerber, R. DeMara, M. Georgiopoulos & A. Gonzalez, "Behavior modeling framework for embedded simulation," in *20th I/ITSEC Conference Proceedings*, December 1998.

[Henninger 2000] A. Henninger, A. Gonzalez, W. Gerber, R. Demara, M. Georgiopoulos, "On the Fidelity of SAF's: Can Performance Data Help?," in *22nd I/ITSEC Conference Proceedings*, December 2000.

[Henninger 2000B] A. Henninger, "Neural Network Based Movement Models to Improve the Predictive Utility of Entity State," Ph.D. Dissertation, University of Central Forida, Orlando, Fl, 2000.

[HLA 1996] , "HLA Time Management Design Document Version 1.0", 1996.

[HLA 1998], U. S. Department of Defense, "Time Management," High Level Architecture Interface Specification Version 1.3, ch. 9, 1998.

[IEEE 1995] Standards Committee on Interactive Simulation, "IEEE Standard for Distributed Interactive SimulationÑ Application Protocols", 1995.

[Jones 1986] D. W. Jones, "An Empirical Comparison of Priority-queue and Event-set implementations," *Communications of the ACM*, April 1986.

[Jump 1993] J. R. Jump, "YACSIM Reference Manual, Version 2.1", 1993.

[Klingensmith 1998] M. Klingensmith, C. Hobson, "Incorporation embedded Simulation and Training into an Existing Vehicle Architecture," in *SISO,Simulation Interoperability Workshop*, Fall 1998.

[Lamport 1978] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, July 1978.

[Laplante 1997] P. A. Laplante, "*Real-Time Systems Design and Analysis An Engineer's Handbook (2e)*," Piscataway, NJ, IEEE Press, 1997.

[Larocque 1996] G.R. Larocque, S.J. Lipoff, "Application of discrete event simulation to network protocol modeling," in *5th IEEE International Conference on Universal Personal Communications*, September 1996.

[Lawlor 2002] Lawler, M, "Tactical Concepts Come to Life," *Signal Magazine*, November 2002.

[Lehoczky 1996] J. Lehoczky, "Real-time Queueing Theory," in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1996.

[Lin 1991A] Y. Lin, E. Lazowska, "A Study of Time Warp Rollback Mechanisms," *ACM TransactIons on Modeling and Computer Simulations*, January 1991.

[Lin 1991B] Y. Lin, E. Lazowska, "A Time-Division Algorithm Simulation for Parallel Simulation," *ACM Transactons on Modeling and Computer Simulations*, January 1991.

[McDonald 1998A] I. B. McDonald, H. A. Bahr, "Research on Cost Effectiveness of Embedded Simulation and embedded Training," in *Proceedings of SISO: Simulation Interoperability Workshop*, Spring 1998.

[McDonald 1998B] I. B. McDonald, H. A. Bahr, "Research on the Cost Effectiveness of Embedded Simulation and Embedded Training An Update," in *Proceedings of SISO: Simulation Interoperability Workshop*, Fall 1998.

[McDonald 2000] I. B. McDonald, H. A. Bahr, C. Abate, "Cost Effectiveness of Embedded Training On Army Ground Vehicles," in *22nd I/ITSEC Conference Proceedings*, December 2000.

[McHale 1998] V. McHale and W. Braudaway, "Developing Synchronized Player Models for Embedded Training," in *Proceedings of 20th I/ITSEC conference*, Fall 1998.

[Mellichamp 1983] D. A. Mellichamp, "*Real-Time Computing With Applications to Data Acquisition and Control*," New York, Van Norstrand Reinhold, 1983.

[Ourston 1998] D. Ourston, "Staying With the Live Vehicle: the Sychronized Player Model," in *SISO,Simulation Interoperability Workshop*, Fall 1998.

[Petrasko 1993] B. E. Petrasko and F. Meraud, "Distributed Event-Oriented Simulation Using a Hypercube Architecture," in *IASTED International Conference on Modeling and Simulation*, March 1993.

[Pollock 1999] E. Pollock, M. Riley, M. Falash, H. A. Bahr, "Use of Legacy Training Systems in the Development of Embedded Simulation," in *Proc of International Training and Education Conference*, April 1999.

[Riley 2000] M. Riley, "An Evolutionary Approach to Embedded Training," in *22nd I/ITSEC Conference Proceedings*, December 2000.

[Roberts 1998] M.D. Roberts, J.McCutcheio, J.W.Willcox, J.G. Steele, "A Comparison of ModSAF and Enhanced ModSAF Scalability," in *DOD HPC Modernization Program UGC Proceedings*, June 1998.

[Ronngren 1993] R. Ronngren, J. Riboe and R. Ayani, "Lazy queue: New approach to implementing the pending event set," *Int. J. Computer Simulation*, August 1993.

[Ronngren 1997] R. Ronngren and R. Ayani, "A comparative study of parallel and sequential priority queue algorithms," *Transactions on Modeling and Computer Simulation (TOMACS)*, April 1997.

[SAIC 2001A] SAIC, "OTBSaf Ver 1.0 Version Description Document", 2001.

[SAIC 2001B] SAIC, "OTBSaf Ver 1.0 User's Manual", 2001.

[SAIC 2001C] SAIC, "OTBSaf Ver 1.0 Software Architecture Design and Overview Document", 2001.

[SAIC 2001E] SAIC, "OTBSaf Ver 1.0 Programmer's Reference Manuel", 2001.

[Schiavone 2000] G. Shiavone, "Aim point determination," Institution for Simulation and Training, University of Central Florida, Final report, 2000.

[Schriber 2001] T. J. Schriber, and D.T. Brunner, "Inside Discrete-Event Simulation Software: How It Works and Why It Matters," in *2001 Winter Simulation Conference*, December 2001.

[Sun 1996] J. Sun, J. Liu, "Synchronization Protocols in Distributed Real-Time Systems," in *Proceedings of the 16th ICDCS*, May 1996.

[Sun 1997] J. Sun, J. Liu, "Bounding Completion Times of Jobs with Arbitrary Release Times, Variable Execution Times, Resources," *IEEE Transactions on Software Engineering*, October 1997.

[Tan 2000] K.L. Tan, and L.J Thng, "SNOOPY Calendar Queue," in *2000 Winter Simulation Conference*, December 2000.

[Tiernan 1995] T.R Tiernan, "Synthetic Theater of War - Europe (STOW-E)," RDT&E Division, Final Report, 1995.

[Valle 1997] T. Valle, J. Watson, "Communications Techniques for Embedded Training," in *Proceedings of Simulation Interoperability Workshop*, Fall 1997.

[Vaucher 1975] J. G. Vaucher and P. Duval, "A Comparison of Simulation Event List Algorithms," *Communications of the ACM*, April 1975.

[Wieland 1999] F. Wieland, "The Threshold of Event Simultaneity," *Transactions of The Society for Computer Simulation International*, March 1999.

[Wise 1971] I. E. Wise, "*Encyclopedia of Instrumentation and Control*," New York, McGraw Hill, Inc, 1971.