AD-A255 250

ADST/WDL/TR-92-003010

AUG 1 1992

Strawman Distributed Interactive Simulation Architecture Description Document Volume II: Supporting Rationale Book I: Time/Space Coherence and Interoperability

Loral Systems Company
ADST Program Office
Orlando, Florida

31 March 1992

Prepared for

PMTRADE
Program Manager - Training Devices
Orlando, Florida

LORAL
Systems Company

92-21858

92 8 6 031

ADST/WDL/TR--92-003010

Strawman Distributed Interactive Simulation Architecture Description Document Volume II: Supporting Rationale Book I: Time/Space Coherence and Interoperability

Loral Systems Company
ADST Program Office
12423 Research Parkway
Orlando, Florida 32826

31 March 1992

Contract No. N61339-91-D-0001
CDRL A002

DTIC QUALITY INSPECTED 5

Prepared for

PMTRADE
Program Manager - Training Devices
Naval Training Systems Center
12350 Research Parkway
Orlando, Florida 32826-3275

LORAL
Systems Company

Per Form 50
Availability Codes
Dist | Special
A-1

LORAL
Systems Company

AD ST

REPORT DOCUMENTATION PAGE

Form approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	03/31/92	Version 0 03/91 -03/92

4. TITLE AND SUBTITLE

Strawman Distributed Interactive Simulation Architecture
Description Document Volume II: Supporting Rationale,
Book I: Time/Space Coherence and Interoperability

5. FUNDING NUMBERS

Contract No. N61339-91-D-0001
CDRL A002

6. AUTHOR(S)

Beaver, Ray; Brasch, Randy; Burdick, Chuck; Butler, Brett; Downes-Martin, Stephen; Eller, Tim; Ferguson, Bob; Gotuby, Dave; Huber, Joe; Katz, Warren; Miller, Bill; Morrison, John; Sherman, Richard; Stanley, Walt; Yelowitz, K.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Loral Systems Company
ADST Program Office
12443 Research Parkway, Suite 303
Orlando, FL 32826

8. PERFORMING ORGANIZATION REPORT NUMBER

ADST/WDL/TR-92-003010

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Project Manager for Training Devices PM TRADE
Naval Training Systems Center
12350 Research Parkway
Orlando, FL 32826-3275

10. SPONSORING ORGANIZATION REPORT

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The DIS architecture defines a time and space coherent representation of a virtual battlefield environment, measured in terms of the human perception and behaviors of warfighters interacting in free play. It provides a structure by which independently developed systems may interact with each other in a well managed and validated combat simulation environment during all phases of the development process. A fundamental objective of the DIS architecture is to provide a blueprint to guide development of a general purpose system meeting the needs of a wide range of users. The architecture must therefore support the broadest possible range of user needs. At the same time, the architecture and the associated standards must provide sufficient design definition to achieve the goal of transparent interoperability of a very wide range of simulators, simulations, and actual equipment operating on instrumented ranges.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	17. SECURITY CLASSIFICATION OF THIS PAGE	17. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std Z39-18
298-102

Table of Contents

1. Introduction

Volume II provides supporting rationale for the DIS architecture which is presented in Volume I. Some of the key technical problems inherent in DIS technology are identified and discussed in Volume II, in addition to supporting rationale for the proposed DIS Architecture as a suitable framework for describing, designing, and specifying "DIS-compliant" systems. Volume II is divided into two books, described below.

Book One addresses some of the issues surrounding the problem of interoperability and time/space coherence. This problem is central to DIS, and sharply differentiates the DIS environment from that of more traditional stand-alone simulation.

Book Two focuses on some of the major system components that must be considered in a DIS implementation—the network, Computer Generated Forces (CGF), and Higher-Order Models (HOM). Book Two also examines the software considerations that cut across all of these system components.

Taken as a whole, Volume II should be viewed as a collection of DIS position papers which provide supporting rationale for the architecture described in Volume I. In the two Books one will find in depth discussions of some of the relevant problems and in many cases potential solutions. One will also find minority opinions which shed light on the nature of the tradeoffs intrinsic to the world of Distributed Interactive Simulation.

The DIS architecture work draws heavily from the body of work in process by the DIS Conference for the Interoperability of Defense Simulation, which is jointly sponsored by the Army Program Manager for Training Devices (PM TRADE) and the Defense Advanced Research Project Agency (DARPA). Under the auspices of the DIS Conference, representatives of the military services, industry, and associated research organizations have been working toward definition of an industry standard for Protocol Data Unit (PDU) messages. These PDU messages provide the basic means of interaction between DIS simulation entities. The architecture described by the two volumes of this document is generally consistent with the work underway by the conference, but attempts to provide a capstone document which defines the context for the work underway. In some areas the strawman architecture extends beyond the work of the conference by proposing establishment of a specific reference model context for the PDU Standard, definition of some standard terminology for the components of the reference model, and creation of an additional standard governing the set of databases required to support DIS exercises.

Comments and recommendations for changes and improvements are welcome and encouraged. Comments may be submitted to:

Loral ADST Program Office
12443 Research Parkway

Suite 303
Orlando FL 32826
attn: DIS Architecture

Comments may also be submitted via the ADST Bulletin Board System. Post comments in the DIS Architecture Comments area of the BBS. This area is found within the following structure:

ADST Bulletin Board System
System Description and Technical Information
DIS Architecture
DIS Architecture Comments

If you are not already a registered user of the ADST BBS, send a registration request to:

Loral ADST Program Office
12443 Research Parkway
Suite 303
Orlando FL 32826
attn: BBS Administration

2. Interoperability and Time and Space Coherence in DIS

2.1 Introduction

This section defines the relationships between interoperability, time and space coherence, and DIS technology. We explore the nature of these relationships by describing the following: perception and cueing in simulation, the core principles of DIS technology (the operating paradigm), and manifestations of time/space disruption in DIS applications. We end this section by describing a correlation construct that promises to offer a consistent framework for discussing and ultimately measuring correlation between DIS entities. The later sections explore different dimensions of the interoperability problem.

For the purpose of discourse, we define our usage of the following three terms: actual battlefield, virtual battlefield, and physical realization. The *Actual Battlefield* is the combat reality that simulation technology attempts to replicate. Successful simulation will cause its participants to believe that they are immersed in the Actual Battlefield. The *Virtual Battlefield* (also referred to as the Electronic Battlefield) is the simulation illusion itself. The *Physical Realization* refers to the details and mechanics of the underlying networked simulation system which supports the illusion of the Virtual Battlefield.

Against the backdrop of these definitions, interoperability refers to the working together of the components of the Physical Realization to produce a harmonious and useful simulation exercise. To see this more clearly, one needs to reexamine the controlling vision for the the DIS architecture:

> *The DIS architecture defines a time and space coherent representation of a Virtual Battlefield environment, measured in terms of the human perception and behaviors of warfighters interacting in free play.*

Three key points are made concerning the vision.

First, we must decouple the notions of fidelity and time/space coherence. Fidelity describes how well the Virtual Battlefield maps to the Actual Battlefield. The degree of fidelity can be objectively measured in such things as weapon ranges, vehicle dynamic performance, and terrain database correlation to actual terrain. Time/space coherence is concerned with preserving intact the simulation illusion. The degree of time/space coherence is also subject to objective measures—such as correct sequencing of discrete events (missile launch, missile flyout, missile impact), and correlation of a vehicle's position as perceived by different entities on the Virtual Battlefield. However, one can have a highly coherent, logically consistent Virtual Battlefield, and yet have low fidelity.

Second, time/space coherence must be evaluated in terms of human perception and behaviors. Time/space coherence becomes a quality of the

representation of the Virtual Battlefield, found only in this end-product depiction and its associated impact on the warfighter participants. This means that measurements of time/space coherence in the Physical Realization of DIS technology—computers, devices, networks, CIG systems, etc—must be tempered with considerations of human perception. Attempted measurements of time/space coherence, based on analysis of these implementation mechanics—measurements such as network delay time, database correlation degree, CIG transport delay time—serve only as approximations of the true degree of time/space coherence, and one must weigh these numbers against warfighter participant acceptance of the simulation system and its intended application.

Third, while time/space coherence has an objectively measureable aspect, the degree required for a simulation task is application dependent. The DIS user, in the context of his intended application, determines his required degree of time/space coherence. This coherence requirement defines the threshold for interoperability. If the components of the Physical Realization support the time/space coherence requirement, we say the implementation is interoperable.

2.2 Perception and Cueing in Simulation

To explore further, we must illuminate the role of perception (subject response) and cueing (system stimulus) in simulation. One finds that acceptable quality perception and cueing is application dependent. Acceptable cueing is shaped by the expectations of the training session, combat doctrinal experiment, or weapon test.

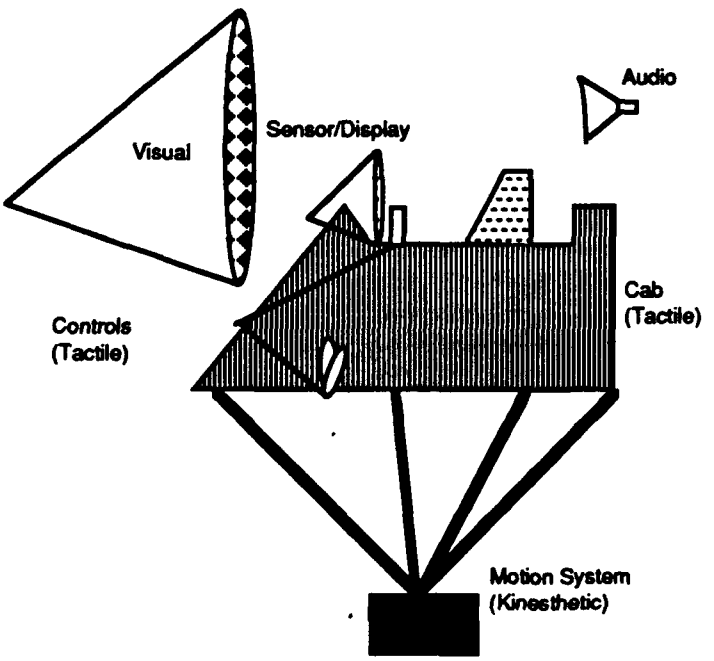


Fig 2.1: Simulator Cueing System

Fig. 2.1 depicts the components of the cueing system of a simulator. Establishment of acceptable time/space coherence means that the various cueing components collectively present an accurate picture of the state of the ownship simulator, as well as of the state of the Virtual Battlefield. "Accuracy" is defined by the simulation application and its objectives.

2.2.1 General Cueing Requirements

Within this wide range of potential expectations, the perception and cueing capabilities of any simulation implementation must demonstrate the following three capabilities:

- **Suspension of Disbelief.** The perception/cueing system must make the intended application "work". The simulated environment must present a rich variety of realistic cues to support the participant's total immersion into the Virtual Battlefield.

- **Selective Fidelity.** In any environment characterized by limited resources, successful systems are those optimized to meet their performance requirements while not dissipating resources on less important goals. The same holds true in simulation. The perception/cueing system must concentrate on faithfully rendering those aspects of reality which bear directly on the outcome of the intended application. These aspects must be enhanced, while other extraneous aspects, must be sacrificed.

- **Elimination of Erroneous Cues and Anomalies.** The cueing system must suppress erroneous cues and distracting anomalies. Intrusion of erroneous cues degrades the outcome of the simulation session, contributing to negative training and poor experimental results.

The last requirement, the elimination of erroneous cues, is where the onus of time/space coherence really comes home. This requirement can be further refined into the following:

- **Non-contradictory cue set.** The cue stimulations must be physically non-contradictory. The set of cues generated by an event in Virtual Reality must comply with the semantics of the event. For example, pulling back on the stick of a fighter-aircraft simulator should cause the following three cues to simultaneously occur: (1) visual scene rolls downward (to simulate upward pitch) (2) motion base tilts backwards (to simulate the cab pitch motion), and (3) the back and seat panels of the G-seat inflate (to simulate the increased g-loading experienced by the pilot). Failure of these three cues to respond as expected will cause pilot disorientation—distracting him from the simulation task of intent.

- **Correlated in time.** The cue stimulations must be acceptably sequenced and correlated in time. For example, a missile firing event in the same fighter-aircraft simulator should be accompanied by the following

sequence of cues: (1) motion platform/G-seat/G-suit stimulation to simulate the reaction forces of the missile leaving the airframe, (2) visual cues representing the missile flyout, (3) visual cues representing the missile's detonation.

2.2.2 Time/Space Coherence in Stand-Alone Simulation

Establishment of acceptable time/space coherence in a stand-alone simulator (not connected to an outside network) depends on faithful rendering of the cues corresponding to events caused only by the stand-alone host. One of the main obstacles to cue correlation in the stand-alone setting is the visual transport delay, the time that must elapse from when the visual scene is computationally determined--with its targets, terrain, cultural features, etc.--to when it is displayed. This time delay stems from the processing lags inherent in the CIG hardware. Typically, the lag ranges from about 200 ms for ground combat applications to 50 ms for high performance fixed wing applications.

Another obstacle to cue correlation is the use of high resolution sensors with Out The Window (OTW) Visuals. For example, Synthetic Aperture Radar (SAR) may have a range well beyond visual range, and yet exhibit a ground resolution measured in feet. Similarly, narrow field-of-view EO sensors such as FLIR's and LLLTV's may be able to see well beyond visual range under poor OTW visibility conditions (smoke, darkness) with very high resolution. More discussion is provided on this topic in paragrah 4.1.3.

2.3 DIS Paradigm

To understand the additional ramifications for time/space coherence to DIS technology, one must clearly understand the fundamentals of DIS. This subsection puts forth an axiomatic definition of DIS, and then highlights the time and space coherence problem in the light of this new paradigm.

The following four principles characterize DIS technology.

Distributed Autonomous Simulation Entities The Physical Realization of the Virtual Battlefield is distributed among disparate system components, disparate communication systems, and physically-separated sites. The collection of entities forming a DIS implementation are supported by a network of processing resources bundled into autonomous nodes—each node supporting one or several entities. Here, autonomy means that each host processor assumes sole responsibility to register its (battlefield) entity interactions with the Virtual Battlefield, and that it does so without depending on any higher level of coordination, synchronization, or control. A node processor may utilize network services and common servers to accomplish its tasks, but it does so on a demand basis, deciding for itself when to access the network.

"Ground Truth" Communication Protocol Each battlefield entity registers its interaction with the Virtual Battlefield by means of a common communication

protocol. Each entity transmits "ground truth", or an absolute frame-of-reference description of its current state and its combat operations with respect to the Virtual Battlefield.

Receiving Simulation Entity Determines Perception As events unfold on the Virtual Battlefield, perceptual cues are generated solely by the receiving simulation node. No network service or device participates in cue generation or display. Network resources function solely to maintain "ground truth." Depiction of that "ground truth" to the human subjects belongs solely to the host simulation node.

Deviation Reporting Simulation entities communicate only significant changes in their own state. This behavior carries two implications. First, entities communicate only about themselves: what they are doing and what is happening to them. Entities do not communicate their perceptions about the Virtual Battlefield or their perceptions about other players. Second, entities only communicate when appreciable changes have occurred to their state variables. Both of these points are slanted to ease the network communication burden. Point one has the additional impact of logically decoupling the interlocking pieces (entities) of the simulation, thus contributing to a cleaner object-oriented structure.

2.4 Time/Space Coherence in DIS—the "Fair Fight"

Time/space coherence for stand-alone simulators means primarily cue correlation. The consequences of poor cue correlation are negative training for training applications, and false experimental results for engineering design applications. In the realm of DIS technology however, additional causes for time/space disruption can occur, and the consequences of these disruptions can be more insidious—undermining the validity of the application.

DIS technology is vulnerable to time/space disruption from additional causes such as: network communication and processing latency, non-correlation of databases, and differences in Computer Image Generator (CIG) scene-rendering algorithms. Besides faulty cue correlation, the consequences of time/space disruption in DIS applications now include the specter of the "unfair fight"—a case in which each individual participant perceives a cohesive, plausible Virtual Battlefield, while time/space disruptions insidiously undermine the usefulness of the exercise. Assurance of a "fair fight" demands appropriately matching both fidelity and time/space coherence among the participating entities. The problem has corrupting dimensions because seemingly successful simulation exercises—characterized by consistent cue correlation for each of the participating entities—could have covertly tainted results for the intended application.

Successful attainment of the "fair fight" is an important aspect of interoperability.

2.4.1 Latency and Simultaneity Between Observers

Latency is one of the basic and most persistent causes of time/space disruption—persistent because it will always exist as a cause, given the distributed nature of the DIS paradigm.

On the Actual Battlefield, the occurrence of an event (plane crash, weapon release, bomb burst, etc.) is immediately and simultaneously apparent to all who possess the perceptual/sensor apparatus to observe it. There is an absolute battlefield frame of reference which defines ground truth. Each warfighter strains to perceive the absolute frame of reference, for accurate perception is a determinant of success or failure. However, on the Virtual Battlefield, observation of an event is dependent upon network communication.

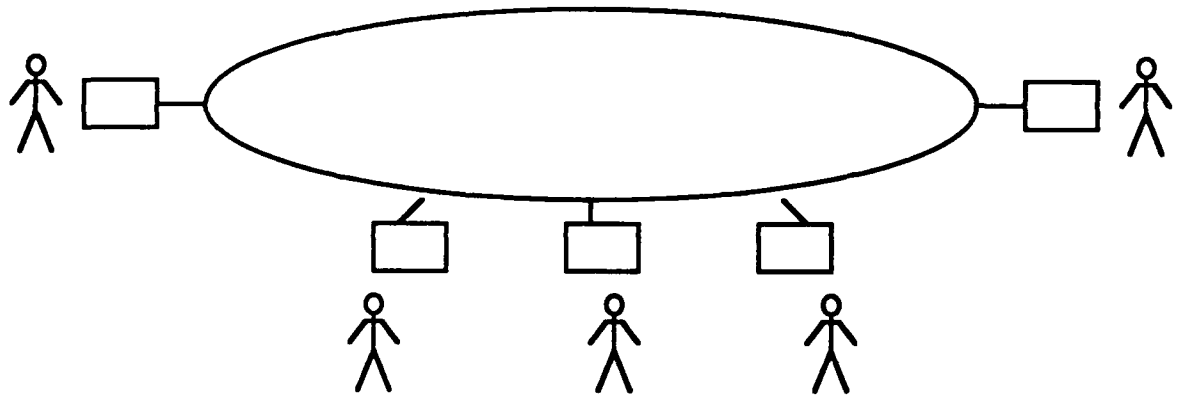


Fig 2.2: Simultaneity between observers

Figure 2.2 shows a virtual DIS network with human warfighters at several remote sites. The network exhibits "simultaneity" when all of the observers experience a given Virtual Battlefield event at the same time. However, due to the fact of network latency, simultaneity can only be approximated, never achieved. Clearly, an absolute frame of reference, common to all observers, can not exist since information takes time to flow through the network. While perfect simultaneity is not required for an effective simulation, reasonably good approximations of simultaneity are required. 300 and 700 ms have been suggested as upper bounds for air and ground applications, respectively.

2.4.2 Scene Integrity

Latency can be manifested as a loss of scene integrity.

The Virtual Battlefield is presented to the warfighter primarily through a sequence of visual frames. To understand how time/space disruption can affect the "fair fight", we need to examine the scene integrity of the presentation. We define "scene integrity" to be the logical consistency:

- Between components that make up a given "still" image frame, and

- Across a sequence of frames.

2.4.2.1 Scene Integrity Within a 'Still' Frame

Figure 2.3 depicts a scene presented to the warfighter. This scene is a composite of several different items:

- Ownship kinematic state
- Othership kinematic states
- Othership events (fire, detonation, etc.)
- Ownship events

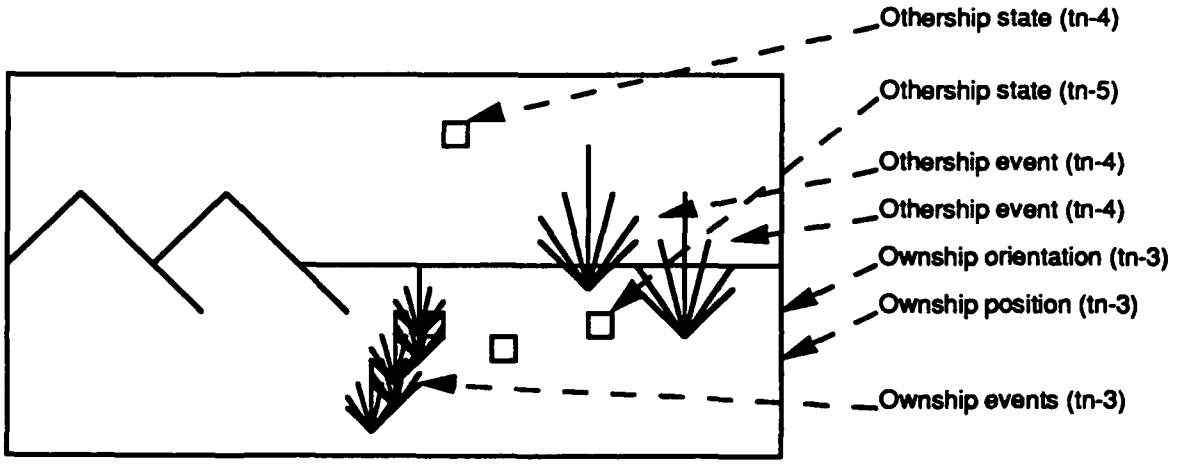


Fig 2.3: Scene integrity within a frame.

Each of these items has a "time" attached, relative to the current time (tn). For instance, the positions of the other vehicles might be older (have lesser time value) than the ownship position because they were computed before the ownship position, and communicated to the ownship host node under conditions of network delay.

2.4.2.2 Scene Integrity Across Frames

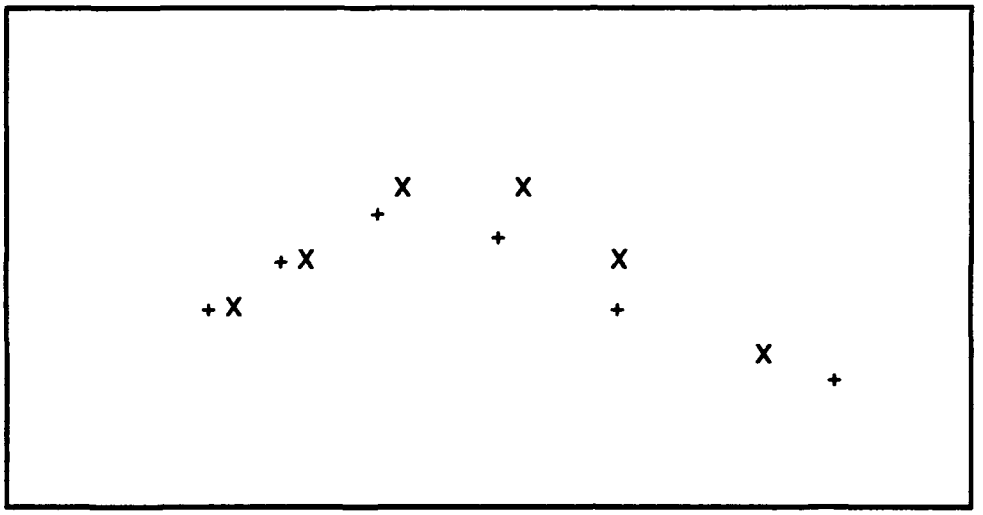


Fig 2.4: Sequential frames of display

Figure 2.4 depicts the sequence of positions ("track") of a given entity over successive frames. Several conditions which will be discussed in the following sections will cause the displayed and actual tracks to differ. In the worst case these displayed tracks may not reflect kinematic realities, destroying the simulation's illusion for the warfighter.

2.4.3 Contributions to Inequity

The consequences resulting from these latency-caused time/space disruptions may include a skewing of results or a tainting of the outcome from a DIS application. When warfighters are behaving according to their perceptions, and yet each warfighter's perspective is shifted from one another's because of time/space disruption on the Virtual Battlefield, then a recording of their aggregate actions may be confusing and inconsistent.

If the level of required time/space coherence is neither specified nor implemented to a stringent enough level, results from the experiment, exercise, or application cannot be trusted. An example best illustrates what can go wrong. Suppose one wishes to test the effectiveness of a new high-performance missile. A DIS application is designed to perform the test. The dynamic performance of the new missile is faithfully simulated. An exercise is conducted in which the friendly side, armed with the new missile, meets the enemy side, armed with more conventional weapons. The exercise is recorded and the outcome evaluated. However, suppose the record contains events such as friendly forces capitalizing on illegitimate firing opportunities because they "saw" the enemy through

conditions of network delay which lagged the enemy's successful and timely search for cover. The enemy would perceive successful evasion, while the friendly side would perceive successful missile impacts. What objective evaluations could missile designers make from such data?

2.5 A Correlation Construct

Time and space coherence can be represented notionally as a two dimensional correlation space, where the two axes are defined by time and space fidelity vectors for a given application or exercise. Each axis can in turn be viewed as a composite expression of the numerous factors that affect time and space coherence - time coherence by factors such as update rates, latencies, vehicle dynamics and network bandwidth, and space coherence by factors such as location, attitude, geometry, and appearance. Figure 2-5 illustrates.

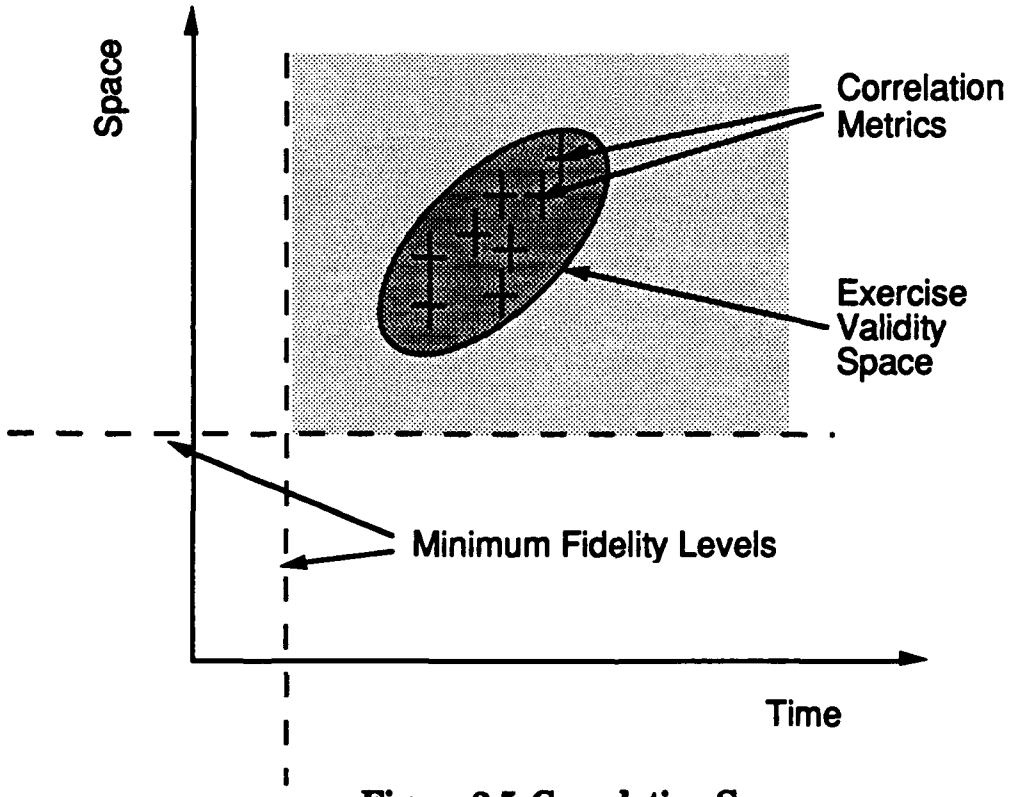


Figure 2-5: Correlation Space

Within this correlation plane we define a minimum level of fidelity on each axis that must be achieved for a given exercise (shown shaded in figure 2-5). The simulation entity correlation metrics are then plotted in the time/space plane, and tested to determine whether or not they fall within an "Exercise Validity Space", shown notionally in figure 2-5 as an ellipse. The notion is that the various entities that make up an exercise must lie relatively close together in the time/space plane for a valid exercise to take place. The location of the ellipse boundary will ultimately be determined by warfighter-in-the-loop experiments to test the correlation hypotheses for each application.

This correlation construct introduces the concepts of relative and absolute correlation. Relative correlation is defined as "closeness" in the time/space plane, the tendency of the metrics to cluster. Relative correlation says something about the similarity between two or more simulation entities, whereas absolute correlation measures the simulation entity against a fixed reference, such as an external source data base. In this sense absolute correlation describes the fidelity of a simulation entity. Clearly it is necessary to consider both aspects of correlation to determine interoperability for a given exercise.

Correlation Metrics

Interoperability between heterogeneous simulation systems requires that an adequate level of correlation be achieved for the intended application. Correlation in DIS is defined as time and space coherence of the simulation entities. It is our contention that correlation can be quantified by measuring the degree of coherence of the simulation Entity's Local Data Base and the Entity Processes.

Entity Local Data Bases are the private data bases associated with the simulation devices; they are derived from public, common source data bases. For example, terrain stored as a polygon mesh is typically proprietary to an IG vendor (the private data base) that derives from DMA or Project 2851 elevation grid data (the public data base). Entity Processes include the rendering algorithms, model dynamics and appearance functions, and in general all of the processing that is performed by a simulator from the retrieval of the local data base data to the output of the processed scene. Figure 2-6 is a simplified diagram of a DIS Entity, which is shown to consist of Entity Processing and a Local Data Base bounded by message and data base standards. Clearly both standards are necessary to permit a consistent and meaningful set of correlation metrics to be defined. The concept of correlation metrics is discussed further in Section 4.

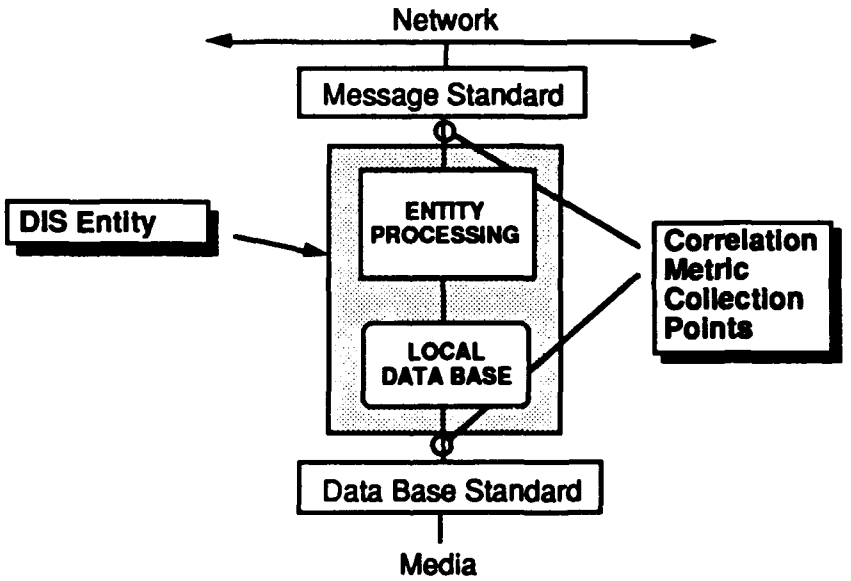


Figure 2-6: Correlation Metrics and DIS Entities

3. Temporal Correlation

3.1 Overview

The lag between when an event happens on the Virtual Battlefield and when it is presented to the warfighter by his simulator cueing systems can cause interoperability problems in DIS applications. Synchronization and timely delivery of messages are some of the most basic challenges in DIS technology. Because the system is distributed, each entity is highly dependent on accurate and reliable communication with external entities, so that it can maintain a current, coherent picture of the Virtual Battlefield.

This section addresses the symptoms, causes, and solutions of temporal disruption, and then discusses the means by which the new DIS architecture supports the implementation and coordination of the temporal coherence solutions.

3.2 Symptoms of Temporal Disruption

3.2.1 Tracking Error

Temporal disruption appears as *tracking error* when the images of external entities, presented to the warfighter by his simulator cueing system, are lagged in time and therefore differ in position from where the sending entities consider themselves to be.

3.2.2 Image Oscillation (Jumps And Jitter)

Image oscillation is a symptom of temporal disruption perceived as jumps and jitters. *Jumps* appear as entities blinking through space, from one position to the next, without seeming to cover the intervening space. A rapid sequence of jumps is perceived as *jitter*.

3.2.3 Out-Of-Sequence Events

Out-of-sequence events are occurrences on the Virtual Battlefield which defy logic by improper sequencing of cause-and-effect actions—like a missile destroying a target before it is even launched.

3.3 Latency—Cause of Temporal Disruption

The main source of temporal disruption in Distributed Simulation is latency—network communication latency, internal processing latency, and the combined end-to-end latency. Tracking error is an obvious symptom resulting from this latency. In this section, we will explore the multiple facets of latency, and come to understand how latency contributes to tracking error, as well as the other symptoms of temporal disruption.

3.3.1 Network Latency

(In order to focus on network issues, we will assume in this subsection that all other time latencies in the system are zero. The validity of this assumption—that network latency can be separately considered—will be be evaluated when combined latency effects from multiple sources are examined.)

Network latency will always be with us, given the fact that information must flow over media (softwired or hardwired) between geographically-dispersed sites in a DIS networked implementation. This section will explore some of the dimensions of network latency.

	T	O	O-S-E
REA	✓	✓	
L	✓		✓
VL			✓
Q	✓	✓	
L + REA	✓	✓	
L + REA + Q	✓		

Legend:

T: tracking error

O: oscillation

O-S-E : out-of-sequence events

L: network latency

REA: Remote Entity Approximation

VL: variable network latency

Q : quantization of time

✓ : sufficient cause

✓ : combined cause

Fig 3.1: Table Of Network Latency Effects

The Table in Fig 3.1 indicates the different aspects of network latency and the temporal disruption symptoms that are associated with them. A thick check marks indicate "sufficient" causes of the indicated effects. The effects would always be present in some degree due to the presence of the cause. The thin check

marks indicate combined causes that work in conjunction with each other to produce the indicated effects.

3.3.1.1 Different Frames of Reference

Primarily, network latency prohibits the existence of an absolute frame of reference of the DIS virtual battlefield.

An observer's (entity's) *frame of reference* is defined by the sequence of packets it receives. In this subsection, we will show that the packet sequence can be different for each observer, and thus *each observer has his own unique frame of reference on the virtual battlefield.*

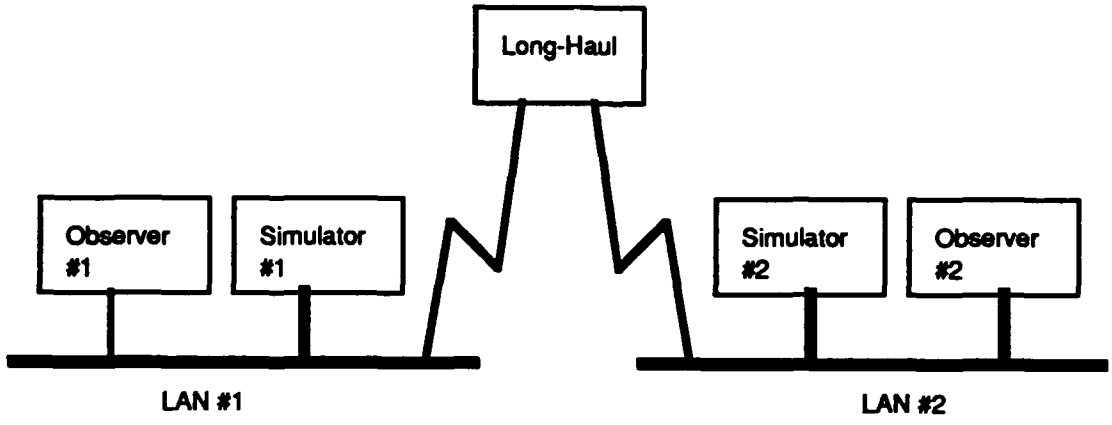


Fig 3.2: Two Lans Connected By Satellite Long-haul Connection

Figure 3.2 shows a DIS network composed of two broadcast LANS connected into a WAN. To heighten the effects of latency, the figure depicts a long-latency (≈ 100 ms. delay) satellite link to connect the two LANS.

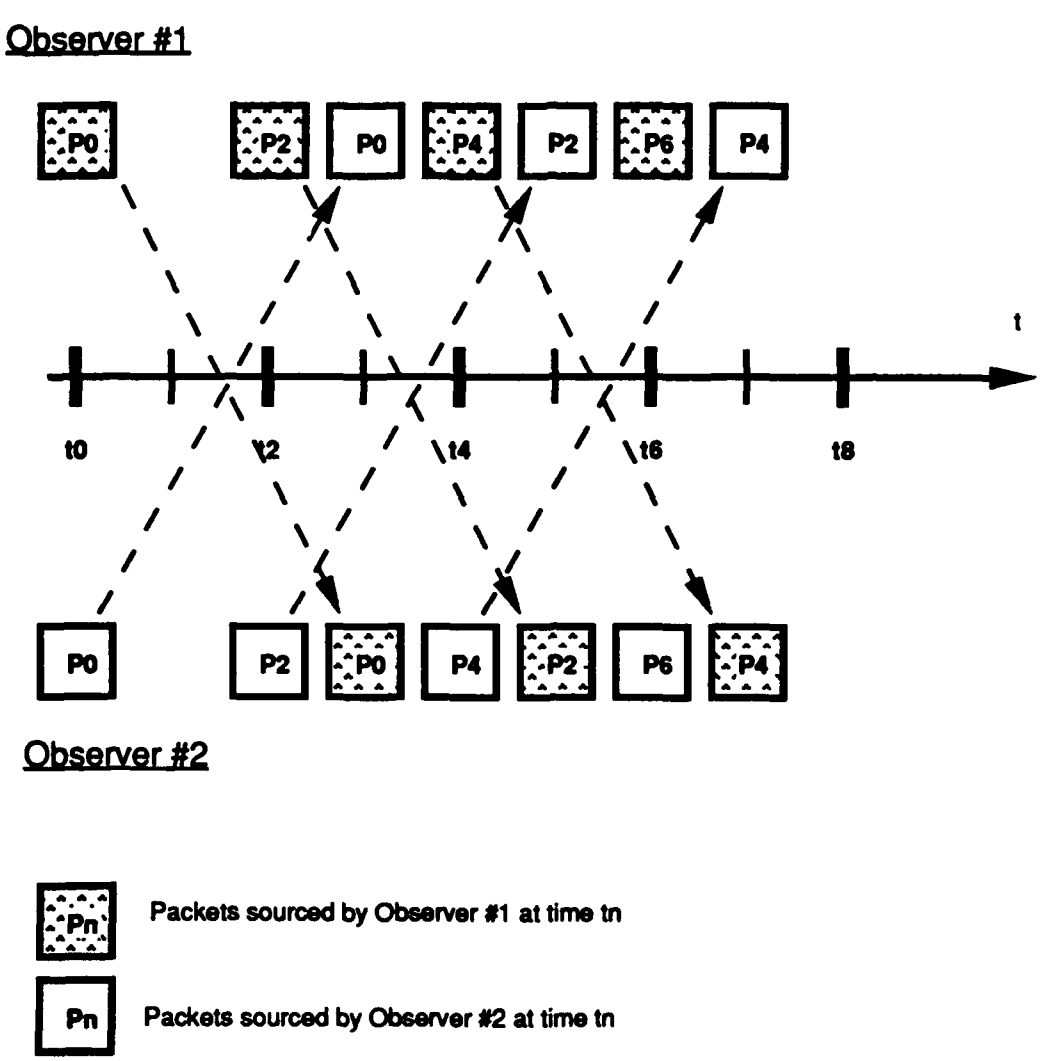


Fig 3.3: Divergent Frames of Reference

Figure 3.3 shows the data streams seen by observers on each of the LANS. The top sequence shows what observer #1 sees, and the bottom sequence shows what observer #2 sees. Note that network latency causes remote traffic to be dislocated in time with respect to local traffic. This causes the two network frames of reference to be different, causing observers on each of the LANs to observe different scenes.

3.3.1.2 Remote Entity Approximation (REA)

Network bandwidth is a scarce resource. The fact of this scarcity has been enshrined in DIS by the principle that a battlefield entity only communicates significant change of its state. In the intervening time, DIS entities must account for each other (in an approximate fashion) by using a DIS-defined extrapolation

technique (known here as Remote Entity Approximation, or REA) to project out a consistent time/space view of the kinematic state of all pertinent external entities.

Much more will be said about REA, but at this point it suffices to describe REA as a common approximation model which is shared by the host entity and all external entities which need to account for the host's kinematic state. The REA is characterized by a common extrapolation technique, and a host-entity-determined error threshold. The host entity maintains its "true" kinematic state on the Virtual Battlefield by means of a high-fidelity kinematic model. In addition, the entity maintains an approximated kinematic state by means of the common extrapolation technique. The host entity continuously compares these two states. When they differ by more than the allowable error tolerance, the host entity corrects its internal extrapolation state by updating with the true state, and continues subsequent extrapolation from this updated state. In addition, the host entity broadcasts its true state to all external entities, so that they in turn may update their approximated state, and henceforth extrapolate from the new true state

As the Table in Fig. 3.1 indicates, REA by itself can contribute to tracking error and oscillation, even without the presence of network latency. Figure 3.4 illustrates how this can occur.

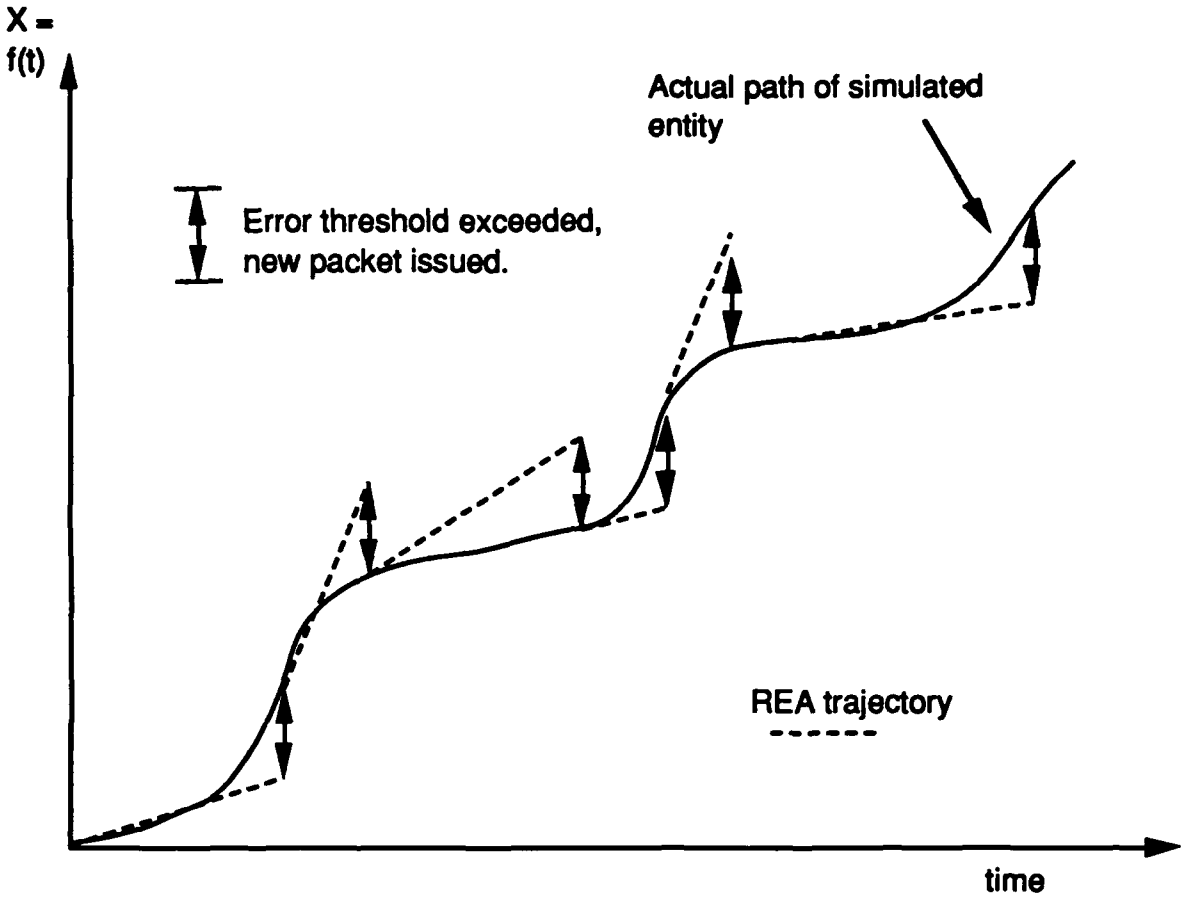


Figure 3.4. - **REA Tracking Error and Oscillation.** Each double arrow represents the time at which the REA position differs from the entity's actual position by a value greater than the threshold. A new packet is sent by the entity, and if the entity is instantly placed at its updated position, a visible position jump the size of the threshold will result. Also, between the times of the position updates, the remotes entity's position model will differ from the actual position model—representing tracking error.

3.3.1.3 Fixed-Time Latency

Fixed-time latency, by itself, can cause both tracking error and out-of-sequence events.

The occurrence of tracking error is due to the lagged information that the receiving entity receives from the sending entity. Kinematic state updates do not immediately reach their destination. Therefore receiving entities, by applying just-received information to the time of reception, instead of to the time of

transmission, must present a perspective that is lagged in time from the true entity position.

Out-of-sequence events can occur, even under conditions of fixed-time latency, when the path-traversal times associated with various entity-to-entity connections vary per connection. This kind of variation depends on the network topology, implementation, and the presence of switching devices. An example of this phenomenon is the case of a missile attack by entity "A", against entity "B", as viewed by entity "C". The missile entity is controlled by the node which supports entity "A". Missile PDUs are broadcast which describe the missile's flight path. At the terminus of the missile flight, "A" issues the detonation PDU which described the hit. Entity "B" broadcasts a stream of PDUs which describe its positional track. Assume these PDUs arrive at "C" such that the missile PDUs are significantly delayed compared to the PDUs from "B". This case may result in "C" perceiving the destruction of "B" (a "B"-issued PDU which describes damage to "B") before the missile even hits "B", and before the consequent detonation.

3.3.1.4 Variable Latency

Another dimension of network latency is its variableness.

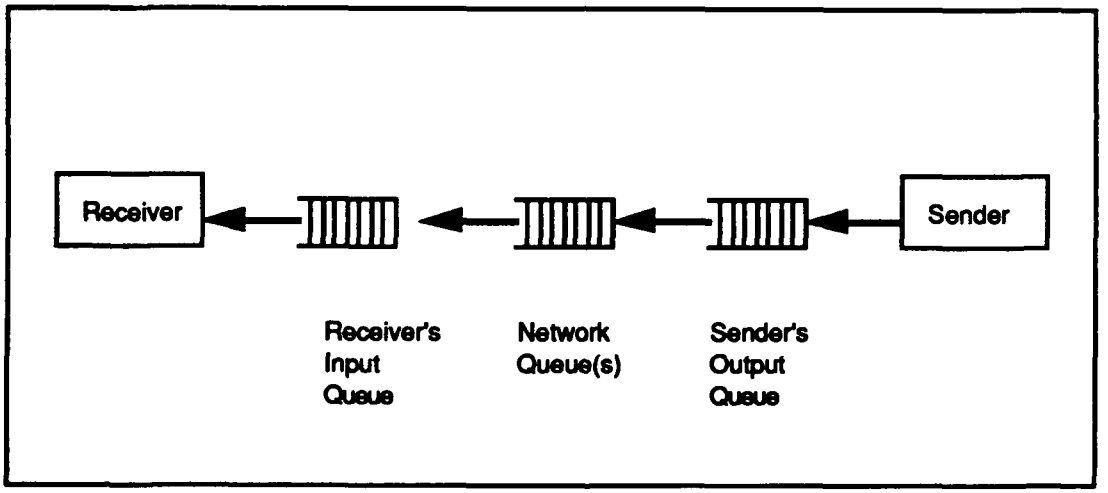


Fig 3.5: Variable Latency Due To Service Queues

Figure 3.5 illustrates one cause of this variableness. It shows a simple queueing model of the path from sender to receiver. The latency for this path traversal may vary because the service times for each of the queues may vary. This is especially true in the case of long-haul networks, packet-switched networks, and networks with "intelligent" gateways or "intermediaries."

Variable latency by itself may cause the perception of out-of-sequence events coming from *a single* entity. Individual PDUs emitted from a single entity may experience different transit delay as they traverse the network. The potential exists that the packets may arrive at a receiver out of order.

To better understand this phenomena of incorrectly-sequenced events, refer to Figure 3.6. The figure shows the stream of PDUs from one transmitting entity out to all receiving entities. This stream (when received) will define the receiver's frame of reference about the sending entity.

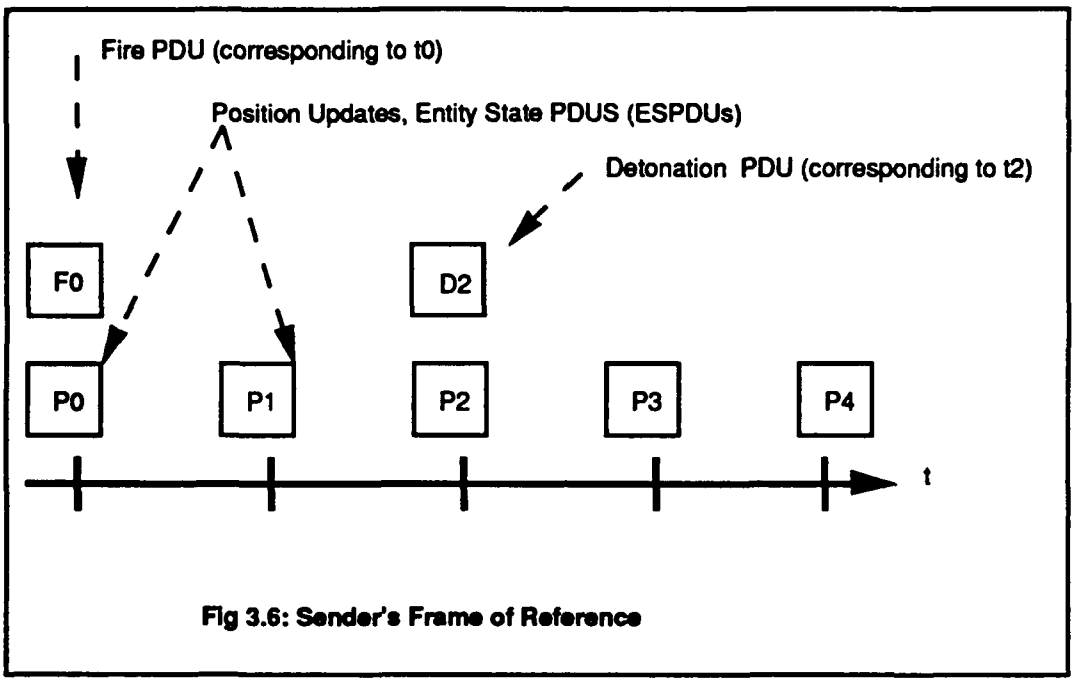


Fig 3.6: Sender's Frame of Reference

Note that the packet stream is composed of two different types of PDUs:

- Entity state PDUs (ESPDUs), which define a kinematic state as a snapshot in time, and
- "Event" PDUs, such as the firing event and subsequent detonation event.

The crux of the problem is that:

- The correct sequence of arrival of ESPDUs and event PDUs cannot be guaranteed,
- Event PDUs and ESPDUs are related in time. (In the figure, the fire PDU "F0" occurs when the entity attains the firing position/orientation described in ESPDU "P0" at time t0. The consequent detonation occurs at time t2, when the entity attains the position/orientation described by ESPDU "P2".)

3.3.1.5 Fixed-Time Latency with REA

We now begin to investigate some of the combined causes.

Fig 3.7 illustrates how latency and dead reckoning contribute to jumps and tracking error that is substantially greater than what would be present under each of the conditions separately.

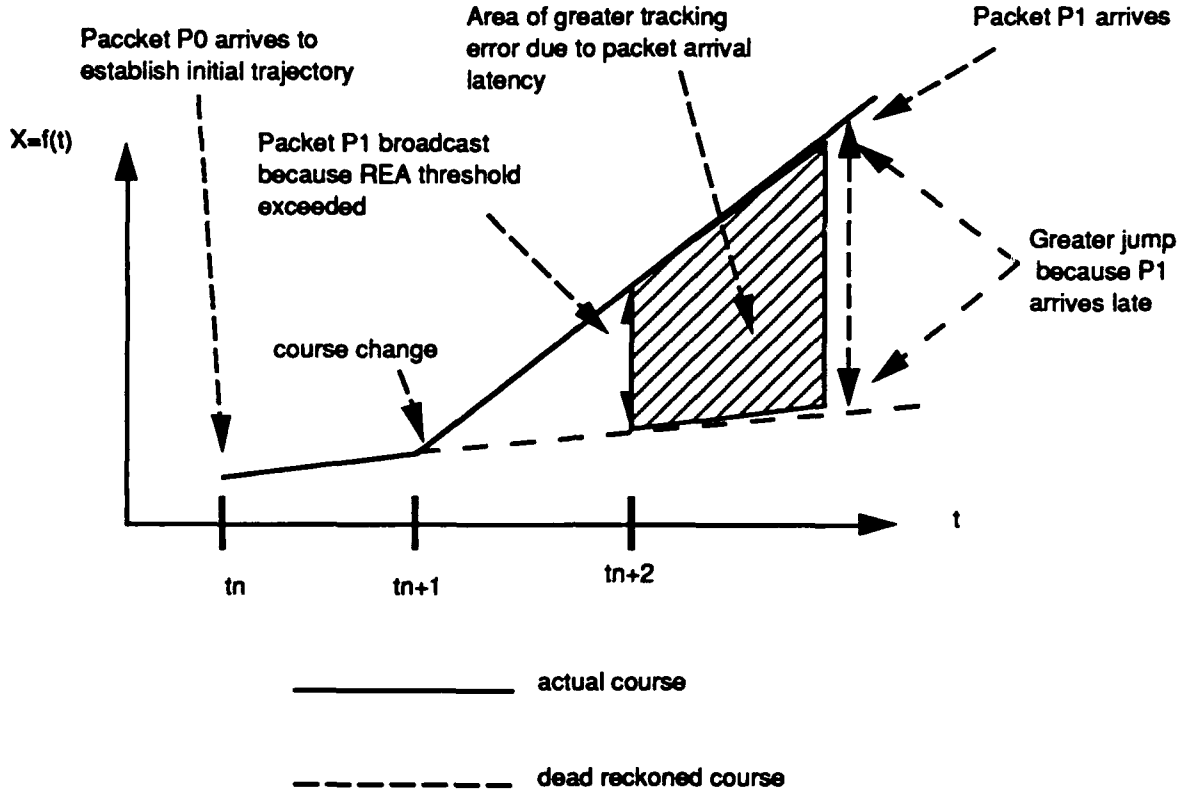


Fig 3.7: Latency And REA On X=F(T)

At time tn, a kinematic state update (in the form of packet P0) establishes the REA trajectory shown by the solid line from tn to tn+1. At time tn+1, a change in course by the simulated entity causes a new course update to be required. Because of network latency, the new course information, contained in packet P1, does not arrive immediately, but is instead delayed. When packet P1 does arrive, the old course has been extrapolated ahead. The sudden correction from the dead reckoned position back to the correct position is perceived as a jump—greater than what would have occurred under REA without delay. And the tracking error that occurs between the time of P1's transmission and its reception is also greater than what would have occurred under REA without delay.

3.3.1.6 Quantization Of Time

Real-time computer simulations mark time by discrete steps, or "frames". The "frame rate" (number of frames processed per second) of a simulation may be constant or may vary, depending on design and instantaneous computational load. The Distributed Simulation paradigm allows simulations with different frame rates to run unsynchronized in networked exercises. This means that

sending entities compute and transmit information according to an internal frame structure which may not be synchronized with that of receiving entities. Also, the information arrives at the receiving entity after a duration of network traversal that again may be unsynchronized with the frame structure of the receiver.

Taken as a whole, these characteristics (discrete marking of time, unsynchronized frame rates, and unsynchronized network traversal time) may cause temporal disruption, or may exacerbate disruption due to latency-related problems. To illustrate, we examine two cases for simulated vechicles in which there exists positional discrepancy between their host entity and remote entities. Both cases concern REA.

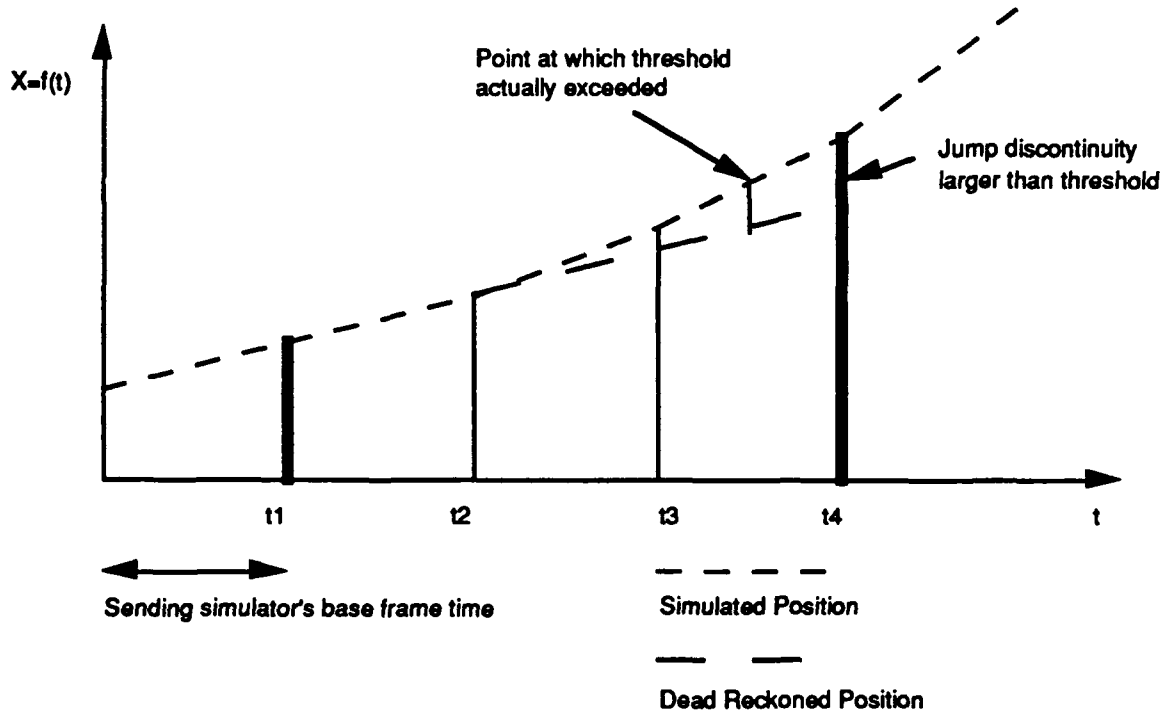


Fig 3.8: Exacerbated Position Error Due To Time Quantization

Figure 3.8 illustrates the greater error that can result when a continuous phenomenon like vehicle position is digitally simulated by marking time in discrete steps. The figure shows the position of the simulated entity as a function of time (x-position only to simplify the analysis). ESPDUs are issued at t1 (to start) and at t4. The t4 communication is necessary because the threshold has been exceeded. Note however that the threshold was actually exceeded earlier, sometime in the middle of the previous simulation frame. The resultant jump discontinuity between dead-reckoned position and actual position exceeds the threshold by a substantial amount because the initial threshold departure occurred in the middle of the frame and was consequently not processed immediately upon occurrence.

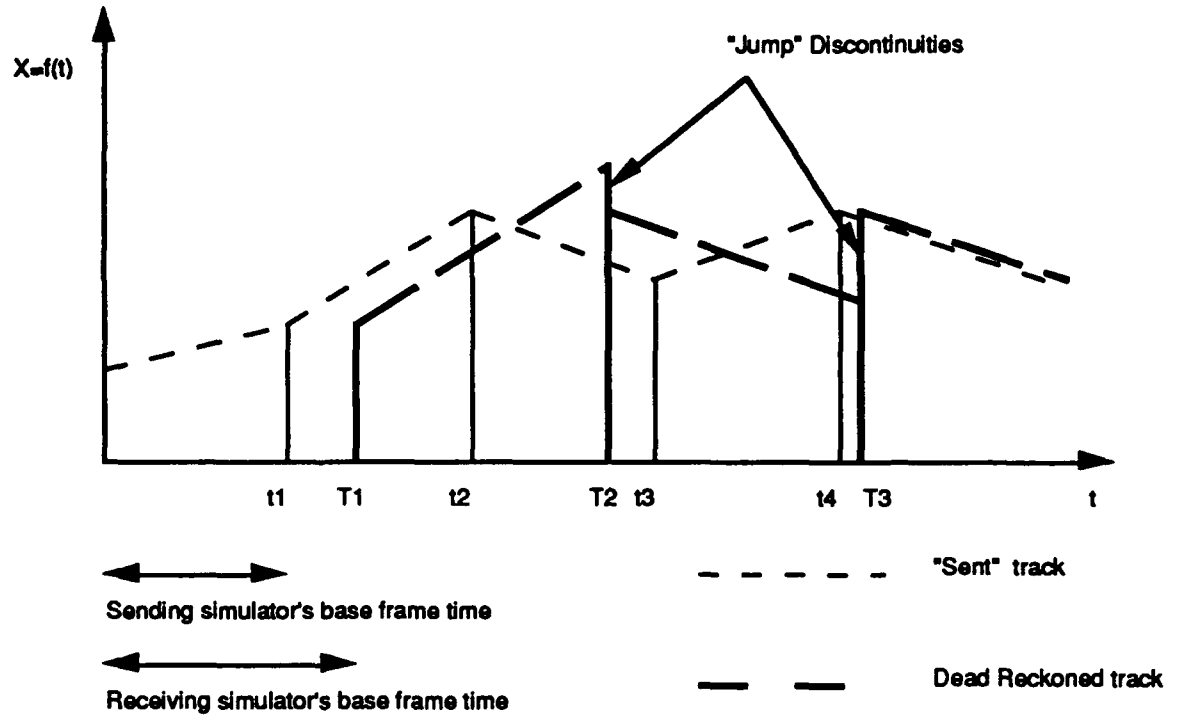


Fig 3.9: Exacerbated Error Due to Time Quantization

Figure 3.9 shows the effects of different base frame rates between sender and receiver. The inherent latency between when the sender transmits the true track data to when it is received is exaggerated by the differences in frame rate (lack of synchronization). The receiver cannot process the data until the beginning of his next frame.

3.3.1.7 Algorithmic Differences

Another source of disruption of time/space correlation comes from the use of different algorithms by disparate simulation entities to process network information for rendering. Section 3.4 describes many algorithms and methods applied to DIS network messages to help provide a coherent view of the Virtual Battlefield. Though the algorithms are effective in the local simulation node, a visual scene will diverge from the visual scene of another simulation node if the other node is using different algorithms. Figure 3-10 shows how the viewed trajectory of an airplane can differ depending on what kind of smoothing is used on the incoming appearance packets.

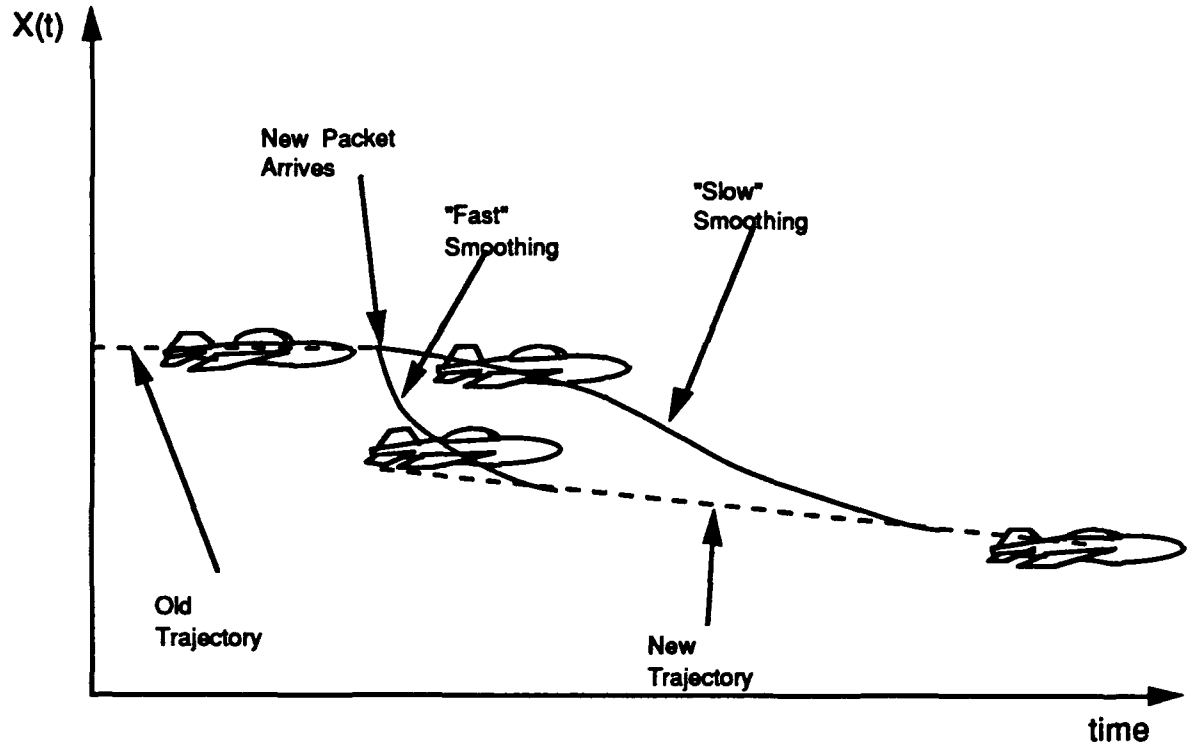


Figure 3.10 - **Algorithmic Differences Cause Time/Space Inconsistencies** . The above picture shows how two different simulation nodes can view the same plane in two different places because of different smoothing algorithms. If one node uses a "fast" smoothing algorithm, and the other uses a "slow" algorithm, the position of the plane will be different for a short time after a packet is received.

The Entity Appearance packet in the DIS protocol provides a data field for describing the dead reckoning algorithm being used by the transmitting entity. Standardization of other algorithms such as smoothing, forward reckoning, etc., can be accomplished by use of a similar mechanism, or, algorithm specification can become part of the session database. In either case, it is important to minimize the inconsistencies caused by mismatched algorithms to the greatest extent possible.

3.3.2 Internal Processing Latency

In this section, we will examine the visual anomalies that arise due to latencies within a given simulation host.

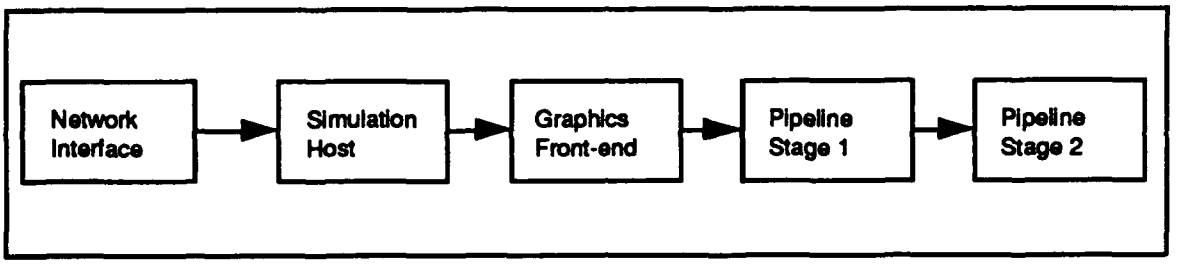


Fig 3.11: Internal Pipeline For CIG Display

Figure 3.11 shows a subsystem architecture of a networked simulator. From left to right:

- A network interface.
- A simulation host, usually running ownship dynamics.
- A graphics front-end, which may perform scene management and ballistics.
- Graphic pipeline stages (here we show 2).

There are latencies between each of the subsystems. We will show that there will be anomalies, however minor, in the network's frame of reference *due solely to these intra-simulator latencies, not due to any network latencies.*

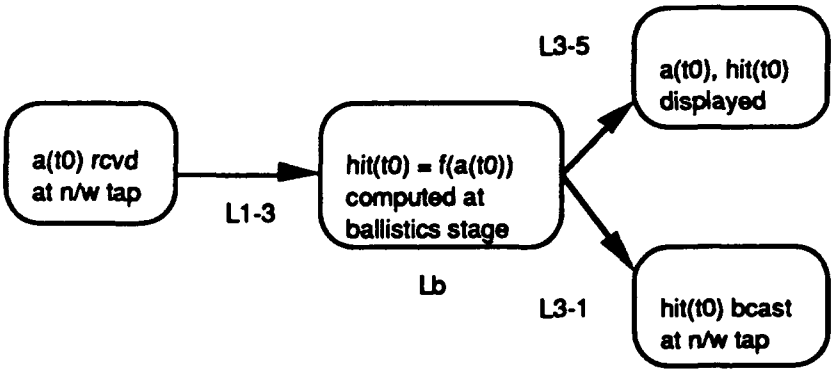


Figure 3-12: Causality Diagram

Figure 3-12 is a "causality" diagram of the events that occur when a simulator so structured simulates direct fire upon another simulated entity. In the Figure and the following explanation,

- *a(t0)* means the "appearance packet corresponding to time t0"
- *hit(t0)* means the "detonation packet computed using *a(t0)* as input"
- *L1-3* means the "latency from subsystem #1 to subsystem #3"

At left, *a(t0)* is received. It takes *L1-3* time to arrive where ballistics are performed. It takes *Lb* time to compute ballistics against *a(t0)*. After a hit is determined, it takes *L3-1* time to get the detonation onto the network.

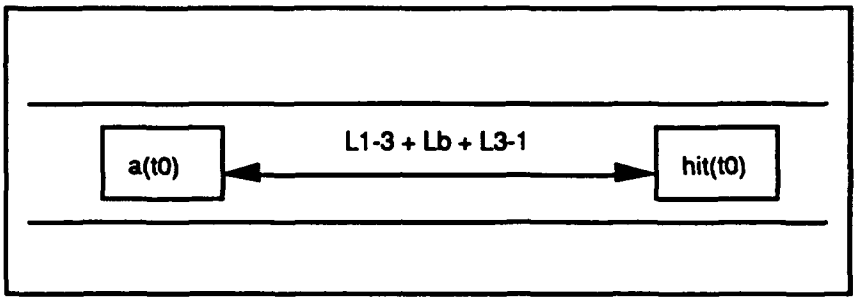


Figure 3-13: Network Frame of Reference

The network's frame of reference is pictured in figure 3-13. Note how the entity appearance packet *a(t0)* and the hit packet *hit(t0)* are displaced in time *(L1-3) + Lb + (L3-1), independent of and unrelated to any network latencies.*

Using *(L1-3) = Lb = (L3-1)* = 67ms, the hit and the entity appearance are displaced in time approximately 200ms, For a fast-moving A-10, 200ms = 1/5sec, at 400m/sec = 80m dislocation.

3.3.3 End-to-end Latency

In this section, we will use our direct fire example again, this time with a network between shooter and victim. We will superimpose the external network latency effects on top of the internal processing latency effects.

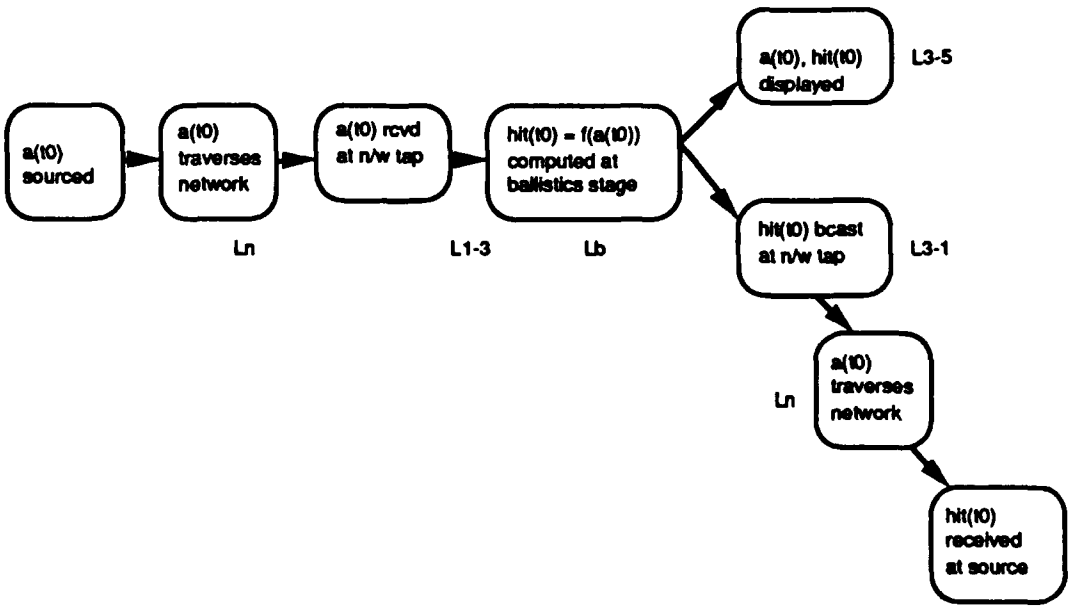


Figure 3-14: Causality Diagram with Network Latencies

The causality diagram is updated to include network latencies; see figure 3-14. The network frame of reference on the victim's net thus looks like figure 3-15.

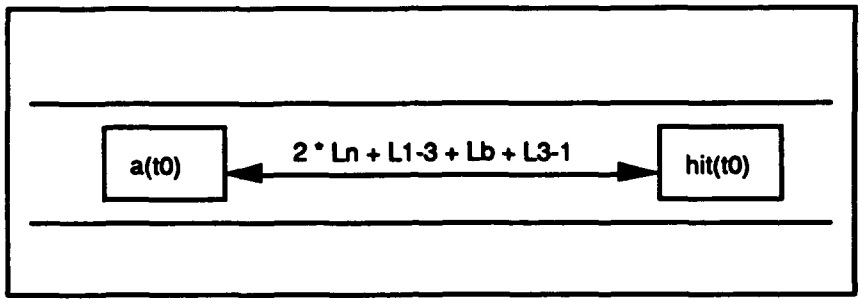


Figure 3-15: Network Frame of Reference on Victim's Net

For a fast LAN (Ln=2ms), total displacement = 204ms, and the extra 4ms causes only 1.6 m additional dislocation. However, with a long-latency satellite delay (Ln=250ms), total displacement = 700 ms. For a fast moving A-10, the additional 500ms latency causes 200m additional dislocation.

3.4 Correlation solutions

In this section, we will examine several algorithms that purport to solve some of the problems we have examined above. Each has advantages and disadvantages, in some cases introducing different, and perhaps less objectionable anomalies.

3.4.1 Timestamping

The DIS Protocol specifies the use of a timestamp field in most DIS packets. There are two types of timestamp values, relative and absolute. Both timestamps indicate the time that the data in the packet is valid.

Absolute timestamps are used when all simulators are synchronized. This can be accomplished through a variety of mechanisms such as common radio reception of a synchronizing signal by simulation nodes, hardwiring a timing signal between nodes (only useful in Local Area Networks), etc. Absolute timestamps in DIS are represented by Universal Time Coordinates (UTC), with an accuracy of 32 bits. By placing an absolute timestamp in a network packet, receiving nodes can determine the time at which the data is valid relative to the time at which the effects will be painted. This technique is described in detail in [Katz, 1992]. There are limitations on the use of absolute timestamps to correct for network and processing latencies. These are discussed in detail in section 3.4.5, Forward Reckoning Algorithms.

Relative timestamps are used when the simulation nodes are unsynchronized, and indicates the time, relative to a particular node's internal clock, that data is valid. It can be used to determine the time correction necessary to account for unsynchronized frames. By relating a remote node's relative timestamp to a local

node's internal clock, a reference point is obtained. When the next packet arrives from that remote node, the relative timestamp can be subtracted from the stored relative timestamp of the previous packet, arriving at the time delay between remote node packet transmissions. This delay can then be added to the reference point, which yields a time, relative to the local simulator, at which the remote information is valid. This mechanism will alleviate much of the jitter associated with observing simulations at non-harmonic frame rates. See the section on Timebase Correction for further details.

Timestamps may be used to correct some of the symptoms of temporal disruption described in section 3.2. The following sections - Dead Reckoning Algorithms, Smoothing, Timebase Correction, and Forward Reckoning - all allude to the use of timestamps. The Entity State PDU, which is the packet of primary concern, contains a timestamp field.

3.4.2 Remote Entity Approximation Algorithms (REAs)

Insufficient network bandwidth can be a problem in networked simulations. The following sections will show how REAs (better known as Dead Reckoning Algorithms, DRAs) can alleviate this problem by trading off computation load for bandwidth.

3.4.2.1 Distributed Interactive Simulation Background Information

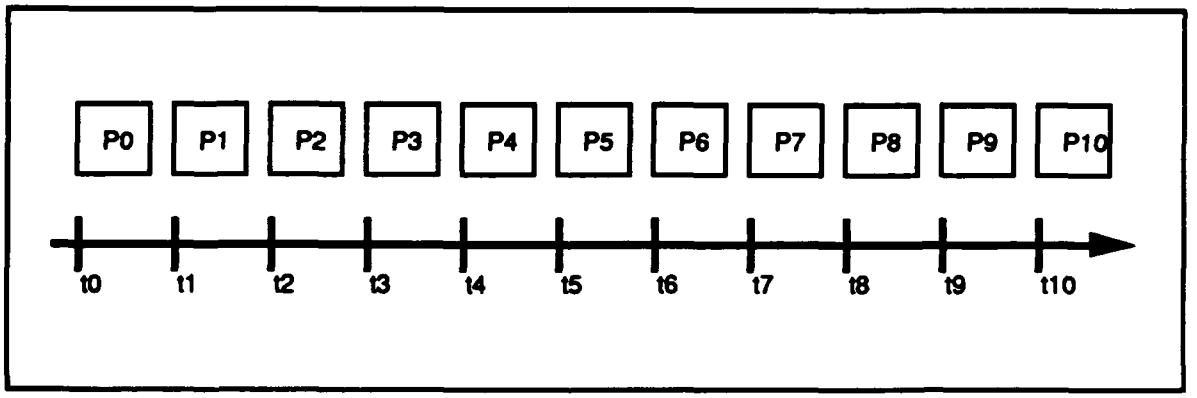


Figure 3-16: Entity States

Figure 3-16 shows a sequence of an observed entity's entity states P0 through P10. ("P0" means "the entity state that corresponds to time t_0.") These are transmitted on the network, and are received at the observer. For simplicity's sake in the following discussions, we will assume:

- Both observer and observed run at the same basic frame rate, which is indicated by the time markings in the Figure,
- Both observer and observed are synchronized, and

- The network has zero latency.

Each position update must contain sufficient information to render the observed entity, including:

- Model Description (entity type),
- Position,
- Orientation, and
- Position and Orientation of articulated parts.

These state variables must be expressed in a data representation expected by the particular CIG. Such a representation is usually optimized for computation efficiency, not size. In general, modern CIGs use silicon which expects:

- Consistent coordinate systems (*e.g.*, right-handed)
- IEEE floating point numbers
- Direction Cosine Matrices for rotational transformations of points (*e.g.*, world to screen coordinate transformations).

3.4.2.1.1. Trade-Off Between Potential Bottlenecks

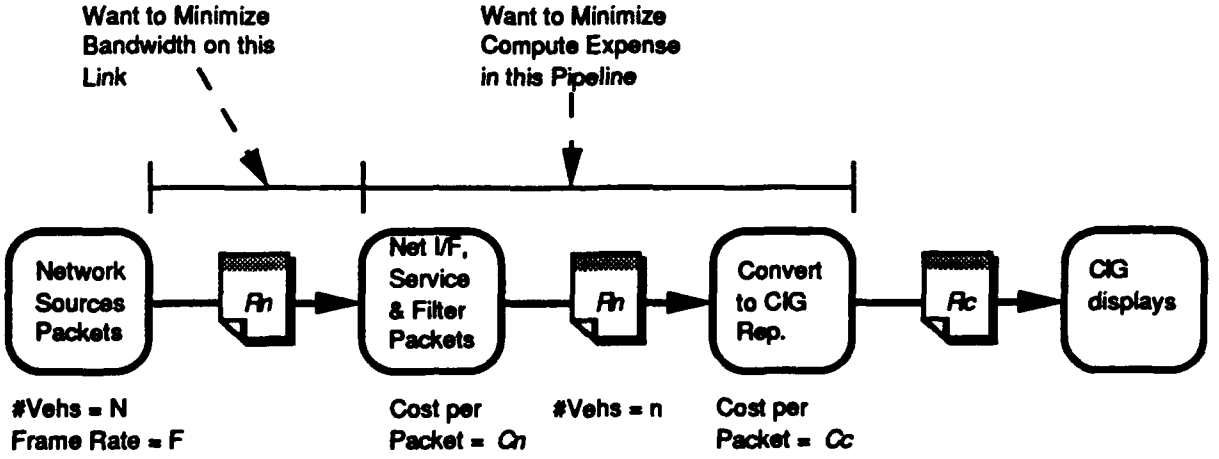


Figure 3-17: Processing at the Observer

Figure 3-17 shows a highly simplified model of the processing at the observer. Briefly, the N entities on the network source packets, each at the basic frame rate R. Each update is expressed in a specific representation R_n, the External Data Representation (XDR) for entity state updates on the network. The Network Interface services each update, and as a performance optimization, reduces the number of "interesting" entities (perhaps those within visible range) to n. The n

entity updates are translated to the preferred CIG representation R_c, and are then displayed.

There are two bottlenecks. One is the network bandwidth, and the other is the local computation power available to perform the processing. Next, we quantify those bottlenecks.

B= Network Bandwidth

M= available MIPS/MFLOPS

N= number entities

P= packets per second. For our example, equal to the basic frame rate.

R_n =representation of entity state update on network (XDR)

R_c =representation of entity state information necessary to display entity on CIG

S= size of each entity state update = $f0(R_n)$

C_n= Cost of entity state update service time = $f1(R_n)$

C_c= Cost of conversion from entity state format to CIG format = $f2(R_n, R_c)$

Thus, bandwidth on the network is:

$$B = NPS = NPf0(R_n)$$

Here, we can see that bandwidth on the network is a linear function of the size of R_n. Thus it is an overriding desire to make R_n as compact as possible.

The computation cost without the filtering optimization is:

$$M = NPC_n + NPC_c$$

If we have the network interface filter out "uninteresting" entities,we reduce the number of CIG format conversions, but arguably we increase the per-packet processing slightly.[1]

[1] CIGs typically have a maximum number of moving models and effects they can display. Processing updates from any more entities is thus pointless. Those systems without CIGs, such as SAFOR, do not have the luxury of such an optimation.

$$M = NPC_n' + nPC_c$$

Since the costs are functions of the network representations...

$$M = NPf1(R_n) + nPf2(R_n, R_c)$$

Here, we see that the computation cost is an increasing function of C_c, the cost of conversion from R_n to R_c. Thus the desire to make this conversion as painless as possible. [2]

In summary, one trades off two potential bottlenecks in choosing an *external data representation* R_n for the state variables in the entity state updates; bandwidth, and computation cost.

3.4.2.1.2. Bandwidth Not Sufficient to Support Large Exercises

As an example, the SIMNET Semi-Automated Forces Proof of Principle (SAFPOP) exercise consisted of over a thousand vehicles. The minimum information necessary to describe the state of one of those vehicles for one frame is approximately 320 bits (assuming 32 bits for each of X, Y, Z, roll, pitch, yaw, entity type, entity id, and each of 2 articulated parts). Assuming each 320-bit entity state is preceded by a 96-bit Ethernet header: we have

$S = 416$ bits, $P = 15$Hz, $N = 1000$, thus

$B = (416 \text{ bits})(15 \text{ Hz})(1000 \text{ entities}) = 6.24$ Mbits/sec.

SAFPOP should have consumed over 6Mbit/sec. However, it was conducted over a 10Mbit/sec Ethernet, with an actual effective bandwidth of no more than 3Mbit/sec, and over a handful of 56kbit data links. How was this accomplished?

---

[2] For this reason, early versions of SIMNET had $R_n = R_c$, using Direction Cosine Matrices for 3D rotations, and sin/cos pairs for 2D rotations. Later versions used BAMs for 2D rotations.

3.4.2.2 DRAs Trade-off Bandwidth vs. MIPS

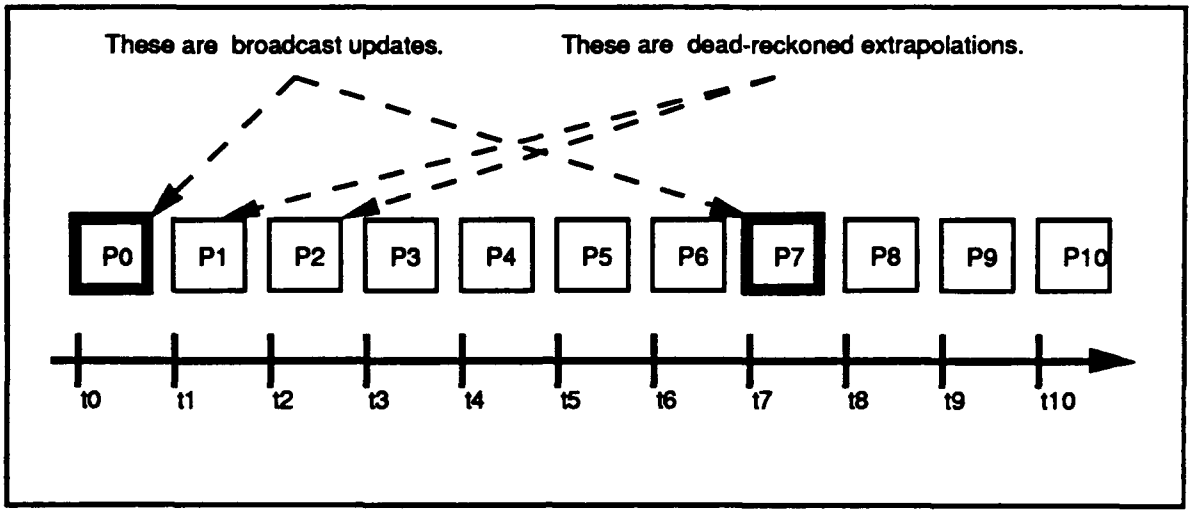


Figure 3-18: Dead Reckoning

3.4.2.2.1. How DRAs Work

In order to employ DRAs to save network bandwidth, the observed entity and its observer agree upon:

- New state variables and their XDRs. These are time-derivatives of the previous state variables (*e.g.*, $\frac{dX}{dt}$)
- A set of state equations (*e.g.*, $X' = X + \frac{dX}{dt} \bullet \Delta t$)
- A set of "thresholds" for selected state variables. (*e.g.*, 1 meter)

The observed entity:

1. Uses its internal kinematics and dynamics models to compute the "actual" values of the state variables.

2. Runs the simple state equations to compute the"lower fidelity" values of the state variables.

3. Computes the discrepancy between the "actual" state variable values and the "lower fidelity" state variable values. The observed entity guarantees that it will send out an entity state update only if

the discrepancy exceeds the agreed-upon threshold.[3] (P0 and P7 in Figure (above).)

The observers make a decision each frame:

1. If an update is received, use its state variable values.

2. If no update is received, run the simple state equations and use the resultant state variable values. (P1-P6, and P8, etc. in the Figure are such extrapolations.)

In SIMNET, this cut the number of updates from 15Hz, to 1Hz to 3Hz.

3.4.2.2.2. DRAs in a Manned Simulator

Figure 3-19 is a version of our previous data flow diagram, updated to include Dead Reckoning Algorithms. Briefly, entity state packets update DRA data structures. In the absence of network updates, the DRAs update the data structures each frame. Also each frame, the CIG interface reads the DRA data structures to provide the CIG with display information.

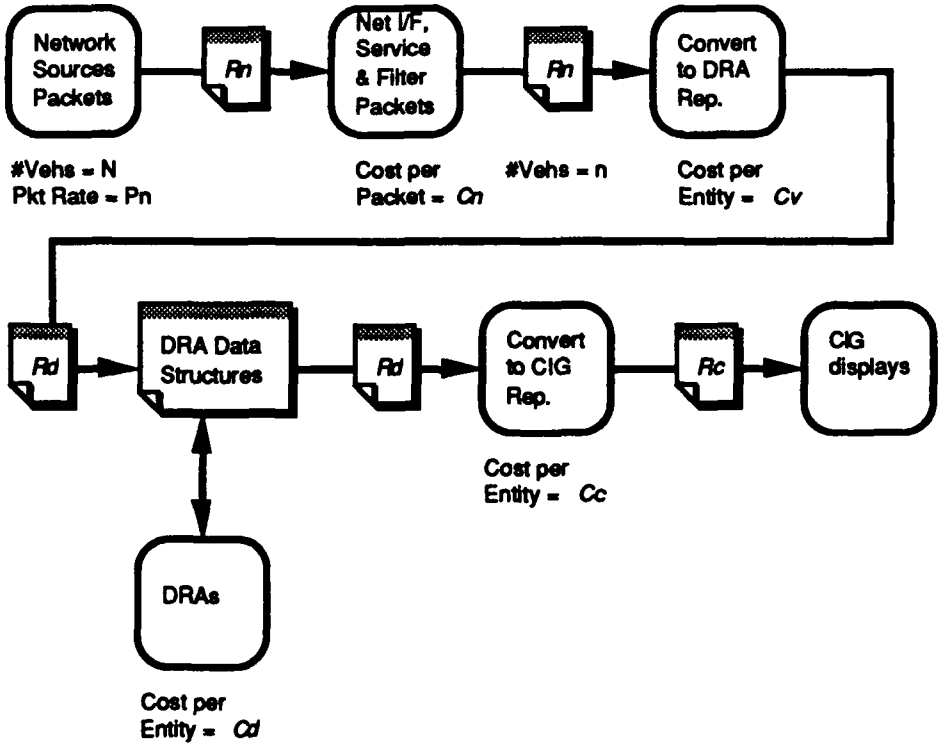


Figure 3-19: DRA Data Flow Diagram

[3] Entity state updates are also sent out periodically for several other reasons. 1, so that late-arriving observers learn of the sending entity. 2, so that errors due to missed or dropped updates are eventually corrected. Hence, the "self-healing" nature of the protocol.

Next, we quantify the effects of DRAs:

R_d=representation of entity state information in DRA data structures

C_v= Cost of conversion from entity state format to DRA format = $f3(R_n, R_d)$

C_c= Cost of conversion from DRA format to CIG format = $f4(R_d,R_c)$

C_d= Cost of DRA application = $f5(R_d)$[4]

P_n= entity state update rate per entity

Thus,

$$B = NP_nS' = NP_nf0(R_n')$$

Because R_n' now contains time-derivative state variable parameters, S' is now larger, by 32 bits for each of:

- first and possibly second order position updates
- first order rotation roll, pitch, and yaw
- first order rotation and position rates for, say, 2 articulated parts

13 * 32 bits = 416 bits. Thus, although S' has doubled in size, P_n has dropped by a factor of at least 6. Thus, total bandwidth is cut by at least one-third. What is the computation cost of such a bandwidth-savings?[5]

$$M = NP_nC_n + NP_nC_v + NPC_d + NPC_c$$

Making the same optimization we did before:

$$M = NP_nC_n' + nP_nC_v + nPC_d + nPC_c$$

$$M = NP_nf1(R_n') + nP_nf3(R_n,R_d) + nPf5(R_d) + nPf4(R_d,R_c)$$

---

[4] Includes smoothing, etc.

[5] We ignore the cost of DRA computation by the sender because it does not grow with either *N* or *n* except in CAUs or CIUs. **It is extremely important not to concentrate on *O(1)* processing instead of *O(N)* and *O(n)* processing.**

3.4.2.3 Cost is a Function of Data Type Representations

In summary, we have established relationships between network bandwidth and size of entity state update. We have established computation cost as a function of representations and algorithms.

$$B = NP_nS' = NP_nf0(R_n')$$

$$M = NP_nf1(R_n') + nP_nf3(R_n,R_d) + nPf5(R_d) + nPf4(R_d,R_c)$$

Where

- $f0(R_n)$ = Bandwidth Cost of size of network representation
- $f1(R_n)$ = Cost of rejecting R_n-formatted packets at network interface
- $f3(R_n, R_d)$ = Cost of translation from network representation to DRA internal preferred format.
- $f4(R_d,R_c)$ = Cost of translation from DRA format to CIG format.
- $f5(R_d)$ = Cost of DRA applications, including smoothing.

An example will help develop a feel for the *O(n)* and *O(N)* processing requirements:

$N = 1000$ entities

$n = 200$ entities

$P = 15Hz$

$Pn = 2Hz$

$$B = (1000)(2)f0(R_n) = 2000(f0(R_n))$$

$$M = (1000)(2)f1(R_n) + (200)(2)f3(R_n, R_d) + (200)(15)f5(R_d) + (200)(15)f4(R_d,R_c)$$

$$= 2000(f1(R_n)) + 400(f3(R_n, R_d)) + 3000(f5(R_d)) + 3000(f4(R_d,R_c))$$

Note how the large coefficients make the required compute power extremely sensitive to changes in the cost functions. Note also that system compute performance can be much more sensitive to R_d than R_n. *It is impossible to overemphasize how critical the data type representations are to system performance.*

For example, SIMNET used $R_n = R_d = R_c$, so f3 was 0, and f4 was zero.[6] f5 consisted of 3 floating point adds (first-order position DRA, no orientation DRA, limited smoothing). f1 was a simple range-check.[7] Thus, SIMNET was able to achieve a high level of performance for minimal compute power (usually a 25Mhz 68020/68881).

Even with ever-increasing performance of hardware, injudicious choices will doom the system to less-than-SIMNET performance.

Functions f1 deserves special attention because it grows *O(N)*. Function f1 is the cost of rejecting R_n-formatted packets from the network. It may contain:

- Cost of DMA of packet.
- Cost of network protocol processing/routing/addressing (*e.g.*, IP/UDP protocol processing).
- Cost of any DIS-level checksum, CRC, or ECC of packet.
- Cost of DIS-level rejection of packet (*e.g.*, range-check to weed out entities that are beyond the visible horizon, entity type-check for more complex rejection function).

Functions f3 and f4 are straightforward transformations of date type representations. However, note that even

Function f5 is the per-entity processing associated with DRAs. It is the topic of the next section.

3.4.2.4 DRAs, Smoothing

3.4.2.4.1. State Variables

DRAs require the following state variables, and zero or more time-derivatives and thresholds for each:

- Position of entity in 3-space
- 3D orientation of the entity
- 2D orientation and position of each articulated part

---

[6] 3D orientation was transmitted as a Direction Cosine Matrix. Later versions of SIMNET used BAMs for 2D rotations instead of sin/cos pairs, necessitating some f4 processing.

[7] Later version of SIMNET compensated for a small *n* by making f1 more expensive and checking for vehicle type as well as range. For example, an ADATS vehicle ignored closer-in ground vehicles in favor of tracking further-away air vehicles.

3.4.2.4.2. State Equations

The DRA state equations and thresholds are important in that they indirectly determine Pn, the frequency at which entities will broadcast updates, using up network bandwidth and simulator compute power. Examples of DRA state equations can be found in a number of DIS position papers.

3.4.2.4.3. Required Data Types

We require a network data type to represent position and its time-derivatives. There is not much debate over using *(X,Y,Z)* 3-tuples or vectors.

For 2D orientations and rates, options include:

- Direction Cosine Matrices (DCMs). Ideal for 3D rotations of vectors, overkill for 2D, and very large (9 floating point numbers).

- Sin/Cos pairs. Can do transformation in a plane with 2 multiplies. Requires 2 floating-point numbers.

- Radian angular measures. Compact (1 floating-point number), but requires *2* transcendental functions (or table lookup and interpolations) to get to Sin/Cos pair.

- Binary Angle Measures (BAMs). Typically fixed-point fractions of a circle. Requires 1 fixed-point number. Advantages include conciseness and maximal precision per bit. Same cost as radians (except arguably faster table lookup), and any extra precision is lost in the translation.

For 3D orientations and rates, options include:

- Direction Cosine Matrices (DCMs). Ideal, but large.

- Quaternions. 4-tuple defining a vector and a rotation about it. Compact 4 floating-point number representation.

- True Euler Angles. Compact 3-tuple.

- Tait-Bryan Angles. Compact 3-tuple, roll, pitch, and yaw.

3.4.2.4.4. Required Operations

- add/subtract multiply/divide
- rotate vector / transform coordinate
- scale by constant

3.4.2.5 Evaluating Different Data Type Representations

3.4.2.5.1. Candidate Representations

- position: (coordinates systems: Z up vs. Z down)
- dp/dt d2p/dt2 (world vs body coordinates)
- 2D: BAMs, radians, sin/cos
- 3D: Tait-Bryan, True Euler, Quaternions, Direction Cosine Matrices

There has been vigorous discussion of these decisions:

- Quaternions (Burchfiel, Saunders)
- Fixed vs Floating (Smith)
- BAMs

3.4.2.5.2. Cost of Operations

In order to determine the optimal representation at each point in the system, we need to quantify the cost functions. Each row of tables 3-1,3-2, and 3-3, when they are complete, can be plugged into the bandwidth and computation-cost equations above. The minimal bandwidth and minimal compute-cost representations are the optimal choices.

The tables assume that Direction Cosine Matrices and World Coordinates are the preferred input formats to most CIGs.[8]

[8] Even those CIGs that accept other representations must convert them to DCMs in order to transform points from world coordinates to screen coordinates. *IS THIS TRUE???*

Table 3-1: Cost Contribution of Various 2D Orientation Representations

Rn	Rd	f3	f4	f5
BAM	BAM	0		
BAM	Radians			
BAM	Sin/Cos pair			
Radians	BAM			
Radians	Radians	0		
Radians	Sin/Cos pair			
Sin/Cos pair	BAM			
Sin/Cos pair	Radians			
Sin/Cos pair	Sin/Cos pair	0		

Table 3-2: Cost Contribution of Various 3D Orientation Representations

Rn	Rd	f3	f4	f5
True Euler	True Euler	0		
True Euler	Tait-Bryan			
True Euler	Quaternions			
Tait-Bryan	True Euler			
Tait-Bryan	Tait-Bryan	0		
Tait-Bryan	Quaternions			
Quaternions	True Euler			
Quaternions	Tait-Bryan			
Quaternions	Quaternions	0		

Orientation representations have a "hidden" cost beyond that in the tables above if body coordinates are selected for position rate changes. The orientation representation can affect the cost of the position calculation.

Table 3-3: Cost Contribution of Various Position Rate Representations

Rn	Rd	f3	f4	f5
World Coords	World Coords	0		
World Coords	Body Coords			
Body Coords	World Coords			
Body Coords	Body Coords	0		

3.4.2.6 Dead Reckoning Fundamental Relationships

We now examine some fundamental relationships or rules of thumb that have use in evaluating Dead Reckoning requirements. For illustrative purposes, we will restrict the following analysis to the case of the single *x*-coordinate only.

3.4.2.6.1. Dead Reckoning Types

We begin by reviewing the mechanics of the two DR types: first-order DR and second-order DR.

As background, we describe the details of positional update in the entity vehicle math models. Typically, the entity positional math model computes, on a frame-by-frame basis, the *x*-forces acting on the vehicle. By means of Newton's Second Law (f=ma), *x*-acceleration for simulation frame n, a_n, is computed based on these forces. *x*-velocity v_n and *x*-position x_n for frame n are then computed by numerical (digital) integration over the time-step interval t_s. The equations are

$$v_n = v_{n-1} + (t_s)\, a_n$$

$$x_n = x_{n-1} + (t_s)\, v_n$$

where $t_s = \frac{1}{t_n - t_{n-1}}$ is the (assumed to be constant, but not necessarily so) time-step interval. This system of equations simplifies to

$$x_n = x_{n-1} + (t_s)\, [v_{n-1} + (t_s)\, a_n]$$

First-order DR is merely replication of this numerical integration process, with the the simplifying assumption that *x*-acceleration is zero. This process

becomes mere linear extrapolation of the last-received positional information according to the last-received velocity information. At time t_n, the correct x-position x_n and x-velocity v_n are received. Until the next correcting update arrives, the DR model position $\hat{x}_{n+m}$ for time t_{n+m} is computed as

$$\hat{x}_{n+m} = x_n + m\, t_s v_n \,.$$

Second-order DR is linear extrapolation of the received velocity v_n according to the received acceleration a_n. The relevant equations at time t_{n+m} are

$$\hat{v}_{n+m} = v_n + (mt_s)\, v_n$$

$$\hat{x}_{n+m} = x_n + (mt_s)\hat{v}_n + \left(\frac{m^2+m}{2}\right)(t_s)^2 a_n$$

where $\hat{v}_{n+m}$ is the DR velocity.

3.4.2.6.2. Required Update Frequency

DIS users and developers must specify the Battlefield Database parameter values that will shape the fidelity of the intended exercise. Dead Reckoning error thresholds (the position update criteria) stand as key parameters of interest in this specification process. While it is tempting to set arbitrarily tight error thresholds to amply ensure entity positional accuracy throughout the conduct of the exercise, developers must guard against the side effects of greater network traffic which will necessarily occur because of greater amount of entity state PDUs required to support the stricter error thresholds.

Let *I* represent the time interval between two successive required updates under dead reckoning. The cognizant user/developer, having specified the DR error threshold T, and having some knowledge of the the maximum acceleration capability of the combat platform, should be able to derive an estimate for the minimum (worst-case) possible value of I, and thereby estimate the maximum possible (worst-case) transient update frequency F required by the DR update model.

We now derive the worst-case estimate for *I* and *F*.

For the first-order DR case, assume time t_n represents the time of the last positional update. A new positional update will be required at time t_{n+m} for which $|\hat{x}_{n+m} - x_{n+m}| > T$ (for smallest possible integral value of m). The shortest time interval between two consecutive updates must occur when the absolute value of acceleration is at a sustained maximum, a_H. For acceleration at this sustained extreme value, the x-position at time t_{n+m} is equal to

$$x_{n+m} = x_n + mt_s v_n + \left(\frac{m^2+m}{2}\right)(t_s)^2 a_H.$$

Therefore, the positional error at time t_{n+m} is equal to $| \hat{x}_{n+m} - x_{n+m} |$ and can be computed by

$$| \hat{x}_{n+m} - x_{n+m} | = | x_n + mt_s v_n + (\frac{m^2+m}{2})(t_s)^2 a_H - x_n - mt_s v_n |$$

$$= | (\frac{m^2+m}{2})(t_s)^2 a_H |.$$

Therefore, an update is required whenever

$$| \frac{m(m+1)}{2}(t_s)^2 a_H | > T.$$

When the above equation is solved for m, the resulting answer becomes

$$m > \frac{1}{2}\sqrt{1 + \frac{8T}{(t_s)^2 a_H}} - \frac{1}{2}$$

for smallest integral value of m.When m is found, then the worst case (transient) update frequency F that can result is $F = \frac{1}{m}$.

For the second-order case, the analysis becomes a bit more difficult. For this case, the DR model position at time t_{n+m} (for the most recent update occurring at time t_n) is equal to

$$\hat{x}_{n+m} = x_n + mt_s v_n + (\frac{m^2+m}{2})(t_s)^2 a_n.$$

Therefore the positional error at time t_{n+m} will be equal to

$$| \hat{x}_{n+m} - x_{n+m} | = | (\frac{m^2+m}{2})(t_s)^2 (a_H - a_n) |.$$

Proceeding as in the first-order case, the minimum integral value of m that will cause an update to be required will be

$$m > \frac{1}{2}\sqrt{1 + \frac{8T}{(t_s)^2 (a_H - a_n)}} - \frac{1}{2}.$$

3.4.2.6.3. Worst Case Positional Error Under Latency

We now bring latency into the analysis. We ask the question: what is the worst-case positional error that can occur for dead reckoned positions when faced with latency? Again, we assume that this worst case error will occur when the subject vehicle is undergoing maximum acceleration a_H. We will again examine both types of dead reckoning.

As we saw above, under Type 1 dead reckoning, the maximum amount that the DR model position can depart from the actual position (under digital simulation) in m simulation frames is equal to

$$\left| \frac{m^2+m}{2} (t_s)^2 a_H \right|.$$

The maximum amount of departure under Type 2 DR (given the same assumptions) will be

$$\left| \frac{m^2+m}{2} (t_s)^2 (a_H - a_n) \right|.$$

Under digital simulation, latency must be measured in integral frames. Thus, m frames of latency will produce a positional displacement error of magnitude

$$m \left| \frac{m^2+m}{2} (t_s)^2 a_H \right|$$

and

$$m \left| \frac{m^2+m}{2} (t_s)^2 (a_H - a_n) \right|$$

respectively under Type 1 and Type 2 DR.

3.4.3 Smoothing

As previously discussed, distributed simulation network traffic can be reduced through the use of dead reckoning. By providing time derivatives of position (velocity, acceleration, angular velocity, etc.) in the appearance packet broadcast by an entity onto the network, remote entities can extrapolate the broadcasting entity's kinematic state into the future. When the error between an entity's internally simulated position, and the perceived position as extrapolated by remote entities, exceeds a predetermined threshold, the entity broadcasts an update with new position and time derivative information. When remote entities receive this update they have to somehow correlate their current dead reckoned position of the broadcasting entity, and the updated information just received over the network. The simplest approach is to place the entity at its new location and continue to dead reckon from there. Though this method requires no computational power, it leads to a disconcerting jump in the position of the remote entity. An observer will perceive the entity instantly blinking from one position to another. This effect jeopardizes the believablility of the simulation, distracting the user from his tasks. It also may produce negative training. Figure 3-20 shows how a "jump discontinuity" is produced.

It is clearly desirable to eliminate these "jump discontinuities" to insure that the effectiveness and believability of the simulation is not compromised.

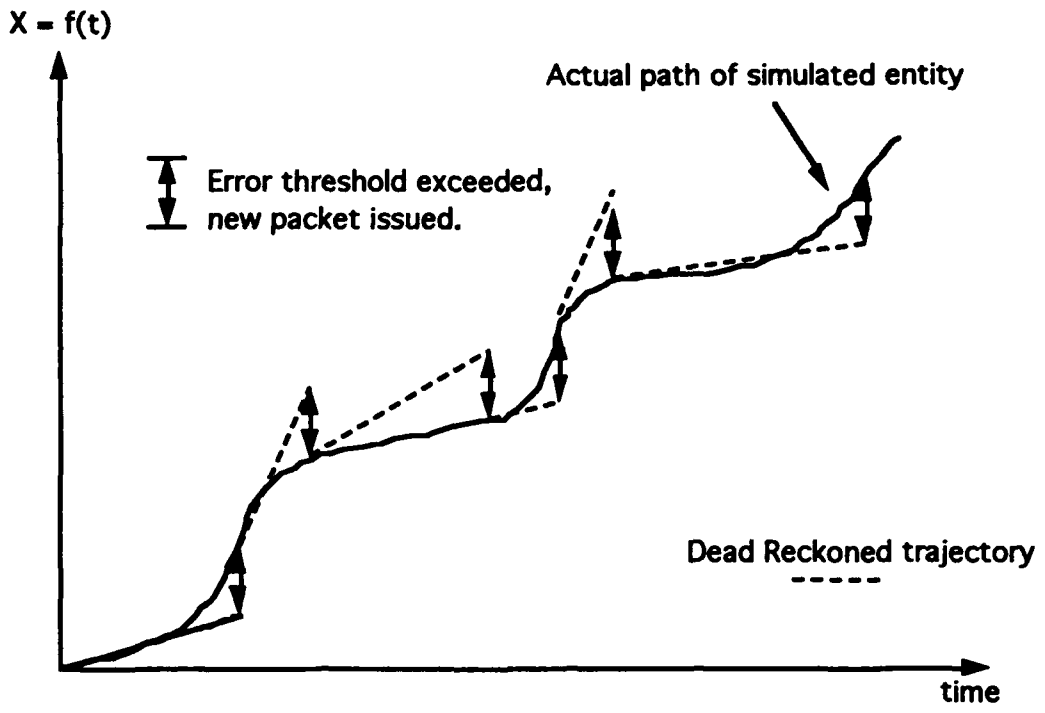


Figure 3-20. - **Jump discontinuities resulting from exceeding the dead reckoning threshold** . Each double arrow represents the time at which the dead reckoned position differs from the entity's actual position by a value greater than the threshold. A new packet is sent by the entity, and if the entity is instantly placed at its updated position, a visible position jump the size of the threshold will result. This assumes negligible network delay.

3.4.3.1 Prior Attempts at Smoothing

An early attempt to provide a so called "smoothing" function in distributed simulation was embarked upon in the SIMNET Stealth Vehicle. The Stealth has the unique capability to "tether" on to another entity, providing the Stealth operator an easy way to observe the battlefield with respect to the tethered entity. Since the position and orientation of the Stealth is dependent on the position and orientation of the tethered entity, the effects of jump discontinuities were greatly exaggerated. When a tethered entity jumped in position, instead of just observing a disconcerting position change in the remote entity, the entire Stealth platform underwent the same jump. The Stealth operator saw his whole world lurch in an extremely disorienting way. For this reason it became necessary to "smooth" out the effects of these position jumps between network updates.

The first smoothing algorithm developed for the Stealth Vehicle is graphically depicted in Figure 3-21. Upon receipt of a network update from a remote entity, the receiving entity selects a time in the future, t_{sm}, at which the remote entity position will coincide with the kinematic state dead reckoned from the newly received appearance packet. This time period is the time over which the

smoothing algorithm operates, hence is called the smooth time. By extrapolating the newly received kinematic information over the smooth time, the position in space at the end of the smooth time, which would have resulted if the entity was dead reckoned with the new dead reckoning parameters, is computed. For SIMNET first order dead reckoning, this intercept position, in one dimension, would be:

$$X_{int} = X_{new} + V_{new} * t_{sm} \quad (1)$$

Where:

X_{int} = Intercept position at desired time t_{sm}.

X_{new} = Updated position of remote entity from packet.

V_{new} = Updated velocity of remote entity from packet.

Using the last dead reckoned position of the remote entity, and knowing the target position in space, X_{int}, at which the two kinematic states will coincide, and knowing the time, t_{sm}, over which that distance has to be traversed , a smoothing velocity is computed:

$$V_{sm} = \frac{X_{int} - X_{old}}{t_{sm}} \quad (2)$$

Where:

V_{sm} = Velocity during smoothing.

X_{old} = Last dead reckoned position of remote entity.

This velocity, V_{sm}, is used to dead reckon (1st order) the entity over the smooth time, from the position X_{old}, to the position X_{int}, where the entity intercepts the trajectory from the packet. Upon interception, normal dead reckoning using the parameters from the packet takes over from the smoothing algorithm. This smoothing algorithm is used on orientation as well as position.

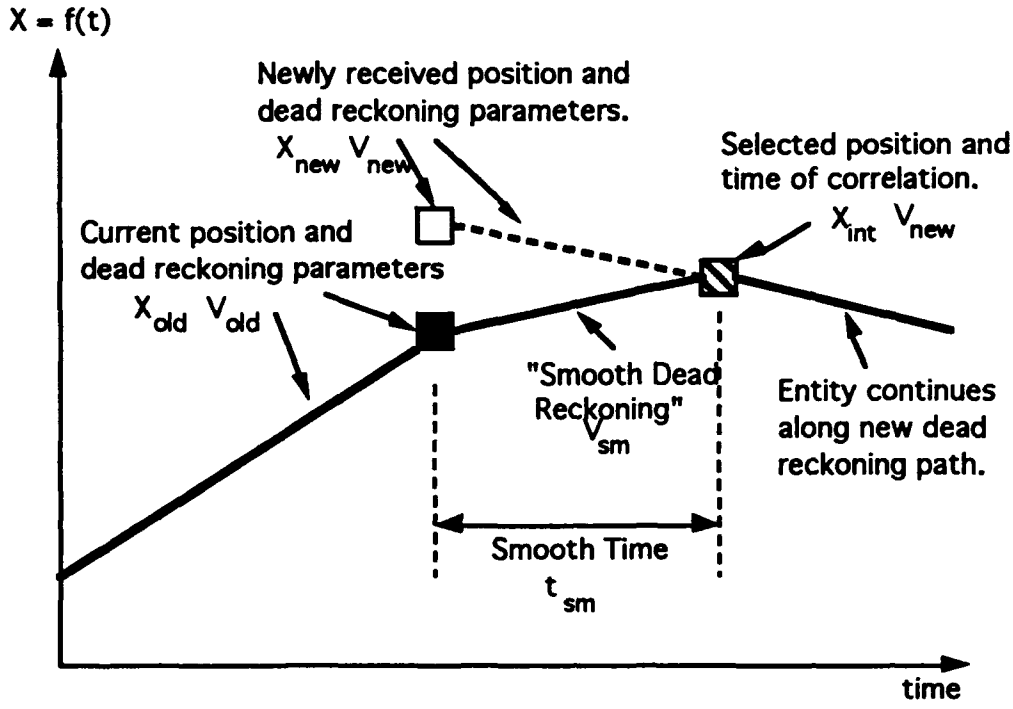


Figure 3-21 - First "smoothing" algorithm in SIMNET . A smooth time, *tsm,* is selected, and the newly received position and dead reckoning parameters are exptrapolated into the future to the "intercept" position, *Xint*. "Smooth" dead reckoning velocity is computed and substituted in the entity's dead reckoning equations. When the entity "intercepts" its new course, the new dead reckoning parameters take over from the smoothing parameters.

In the first iteration of this algorithm, a constant smooth time, t_{sm}, of seven frames, or about half a second was selected. Upon analysis, the constant smooth time appeared to cause two undesirable side effects. Firstly, if appearance packets are typically received before the smoothing is finished, a steady state position error will accumulate. Though the entity will look smooth, it could build up a positional error much greater than the discrepancy threshold. Secondly, if new appearance packets are not required until long after the smoothing is finished, such as in straight line, non-accelerating ground traversal, or straight-and-level flight, the termination of the smoothing algorithm can be visibly discerned because of its relatively large deviation from the otherwise uniform trajectory of the entity. To solve these problems the smooth time was made variable, and was set equal to the time delay between the last two received appearance packets. It was found in SIMNET that the difference in packet rate with respect to time was fairly small, meaning that one could approximately predict when the next packet would arrive based upon the delay between the two most recently received packets. This improvement solved the aforementioned problems and is currently the algorithm in use in SIMNET today. Smoothing of turrets on tanks was also implemented using this same basic algorithm.

Though the SIMNET smoothing algorithm works very well in most cases, it does still produce some visible anomalies. Figure 3-22 shows the smoothing trajectory from Figure 3-21, as well as the time derivatives of that trajectory. The figure shows that the velocity of the entity changes as a step function while the acceleration consists of two impulses. An impulse in acceleration implies that an infinite force was applied to the entity for an infinitessimally small time, the product of the two being the magnitude of the impulse. Infinite forces don't occur very often in nature. When observing the SIMNET smoothing algorithm in action, the instantaneous step change in velocity is actually noticeable, and distracting on occasion.

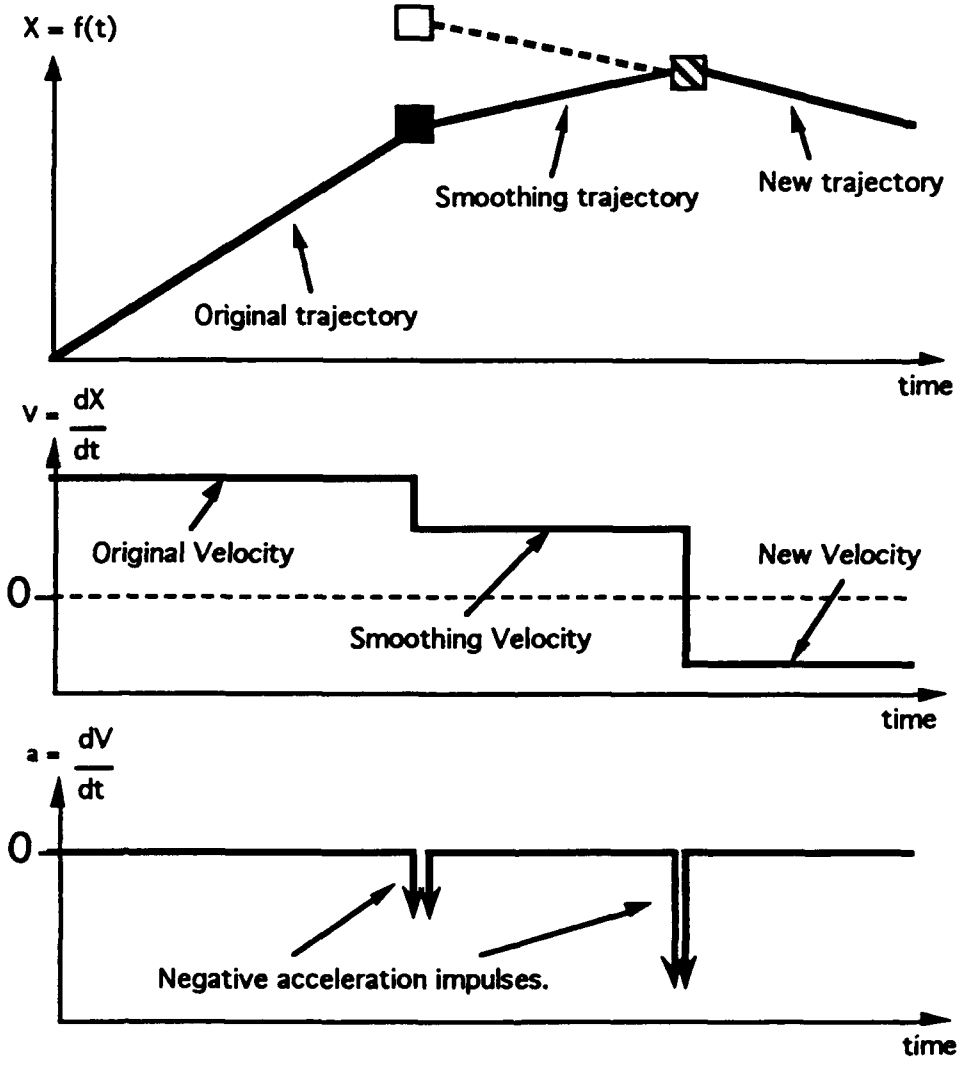


Figure 3-22. - **Time Derivatives of SIMNET Smoothing Algorithm.** The top graph shows the change in position due to smoothing (from Figure 3-21). The second graph shows the first derivative of position (velocity), and the third graph shows acceleration. The acceleration profile indicates that two impulses are generated by the smoothing algorithm. This is an unrealistic dynamical event for entities and can be visually disconcerting.

3.4.3.2 Optimized Smoothing Algorithms

In the quest for an "optimal" smoothing algorithm, there are three main requirements which may be traded off against each other:

- The smoothing should look realistic, should not be visually noticeable, and should not distract the user.
- The position error between the smoothed entity and the straight dead reckoned entity should be minimized.
- The computational load of the smoothing algorithm should be minimized.

Section 5.0 discusses in detail the issue of overload management as it relates to computational burdens in DIS simulators. Clearly, if an entity is too far away to benefit from smoothing, or dead reckoning for that matter, the simulator should switch to a less computationally intensive algorithm. For the purpose of this discussion we will assume that an overload management policy will degrade our "optimal" smoothing algorithm to another algorithm when convenient. The first two requirements, visual realism and minimized position error, then become our primary concerns.

3.4.3.2.1. Visual Realism in Smoothing

Visual realism in smoothing is achieved when a dead reckoned entity switches from one kinematic state to an updated one in a natural enough way such that the user is not alerted to the transition. In order to design an algorithm to accomplish this task we must first analyze human perception as it pertains to visual understanding of dynamic scenes. There are three key visual mechanisms developed over millions of years of evolution to cope with the tracking of objects in motion: saccade, smooth pursuit, and optokinetic nystagmus. There is also the intellectual process of determining that the visual scene is a plausible reality.

The saccade is a rapid eye movement which corresponds to jumping from one object to another over a relatively large distance. When the eye performs a saccade, the acuity threshold of the visual system drops an order of magnitude so as not to overload the brain with too much information. This is called saccadic suppression. When the eye comes to rest in the area of interest, visual acuity returns to normal after a small delay. This is usually followed by a corrective saccade to account for any error in targeting. Saccades are both voluntary and involuntary. The saccade, in and of itself, does not participate in tracking of objects in motion, but is used by smooth pursuit and optokinetic nystagmus.

Smooth pursuit is the motion of the eyes when tracking a moving object. Pursuit is initiated when an object of interest begins to move out of the fovea region (highest acuity) of the eye. The brain, detecting the error in position of the object, initiates a saccade which jumps the eye just ahead of the moving object. The eye then smoothly tracks the object, remaining just ahead of it, during its

travel. If the object begins to accelerate, the brain compensates for the change in velocity such that the object remains in the desired view spot. If the object accelerates too quickly such that it successfully departs the foveal region, the brain initiates another saccade to correct for the displacement, then compensates for the new velocity as best it can. These corrective saccades are very noticeable, such as when a person is trying to track a rapidly moving fly and can't "maintain lock" on the target..

Optokinetic nystagmus is a phenomena which occurs when a person watches a rapidly moving pattern. Since the brain cannot process all the information associated with a rapidly moving pattern within its visual field, it forces the eye to track a fixed point on the moving object, then performs a saccade to return to the original position in space, where it acquired the original fixed point, to acquire another fixed point. This repetitive motion resembles that of a mechanical typewriter carriage return. Examples of optokinetic nystagmus are: watching a picket fence while driving in a car, watching the turbine of a jet engine spin up, or watching a merry-go-round.

Intellectual verification of scene plausibility is the comparison of a viewed scene with memories of similar scenes and behaviors, and authenticating or dismissing the comparison. Many optical illusions can be created to illustrate this point. Figure 3-23 is a commonly known optical illusion that violates spatial relations. Other violations of scene plausibility occur when objects in motion violate the laws of Newtonian physics, such as an object moving from one place to another in an unreasonably short period of time, or a cartoon character running off a cliff but not falling until he looks down.

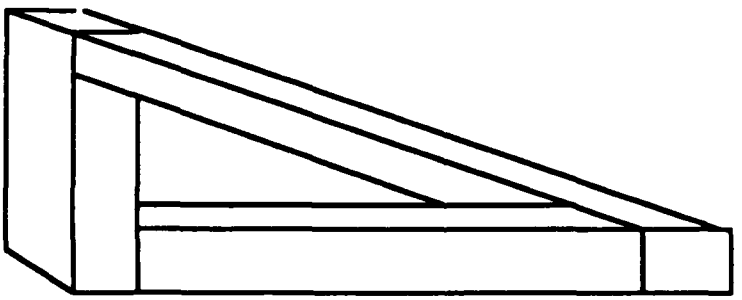


Figure 3-23 - Optical Illusion . The human brain can reason that this object cannot exist based on past experience in spatial relations.

These three visual mechanisms, coupled with reasoning ability, dictate some basic requirements that an acceptable smoothing algorithm must address:

- Entities must not instantaneously change position. This comes from the need to avoid saccadic corrections during smooth pursuit, and avoid violating scene plausibility.

- Entities must not instantaneously change velocity. Instant velocity changes can induce saccades in smooth pursuit if the velocity change is large. Instantaneous velocity changes don t usually occur in nature, making such changes violate scene plausibility. Magnitudes of accelerations should be kept to a minimum to avoid saccades during pursuit.

- To accommodate scene plausibility, accelerations on entities should correspond to realistic forces.

The above requirements indicate that visual realism of a transition from one position/velocity kinematic state to another is primarily dependent on the acceleration profile. Figure 3-24 shows the same smoothing situation of Figure 3-22. Instead of computing a linear smoothing velocity to get from the last dead reckoned entity location to the intercept point, we form an acceleration function as shown. This acceleration function consists of a positive acceleration step of magnitude a, time duration t_1, and a negative step of magnitude $-a$, time duration $t_{sm} - t_1$. A profile of two steps in acceleration corresponds to the onset of a constant force in one direction, followed by an equal force in the opposite direction. The magnitude a, and the variable time duration of the two steps relative to each other permit the entity to intercept its new trajectory in both position and velocity. For a given smooth time, t_{sm}, this function yields the lowest possible acceleration magnitude that can still solve both position and velocity interception.

This algorithm works even better for second order dead reckoning models than for first order. A second order model would have non-zero accelerations on either side of the step profile in Figure 3-24. This would look very natural to the eye because the velocity would be a sawtooth form instead of a constant velocity profile interrupted by an individual tooth.

To solve for the smoothing step time t_1, and the acceleration magnitude a, we use the basic equations of motion:

$$X = X_0 + (V * t) + (\frac{1}{2} a * t^2) \qquad (3)$$

and,

$$V = V_0 + (a * t) \qquad (4)$$

Using the same kinematic parameters from the SIMNET smoothing example, with t_{sm} having been chosen and X_{int} having been computed, our acceleration profile yields:

$$X_{int} = X_{old} + (V_{old} * t_1) + (\frac{1}{2} a * t_1^2) + [(a * t_1) + V_{old}] * [t_{sm} - t_1]$$

$$- \{\frac{1}{2} a^*[t_{sm} - t_1]^2\} \quad (5)$$

and,

$$V_{new} = V_{old} + \{a^*t_1\} - \{a^*[t_{sm} - t_1]\} \quad (6)$$

For convenience we define:

$$\Delta X = X_{int} - X_{old}$$

$$\Delta V = V_{new} - V_{old}$$

$$M = \Delta X - \{V_{new}^*t_{sm}\}$$

Solving for a in equation (6):

$$a = \frac{\Delta V}{\{2^*t_1\} - t_{sm}} \quad (7)$$

and substituting for it in (5), we can solve for t_1 :

$$t_1 = \frac{M +/- \sqrt{M^2 - \Delta V[\{V_{new} + V_{old}\}^*\frac{t_{sm}^2}{2} - \Delta X^*t_{sm}}}{\Delta V} \quad (8)$$

Once t_1 is known, it can be substituted back into (7) to solve for a.

Equation (8) has a singularity at $\Delta V = 0$. This is not a problem for two reasons. First, as ΔV approaches 0 in equation(8), t_1 approaches $t_{sm}/2$. Secondly, $\Delta V = 0$ implies that the updated velocity is the same as the old velocity. We can immediately assume that $t_1 = t_{sm}./2$ because the the positive acceleration step must have the same duration as the negative acceleration step if the end velocity is to be equal to the start velocity. This simplification allows us to rewrite equation (5) as:

$$X_{int} = X_{old} + \{V_{old}^*t_{sm}\} + \{a^*\frac{t_{sm}^2}{4}\} \quad (9)$$

We can then solve directly for a:

$$a = \frac{4*[\Delta X-(V_{old}*t_1)]}{t_{sm}^2} \qquad (10)$$

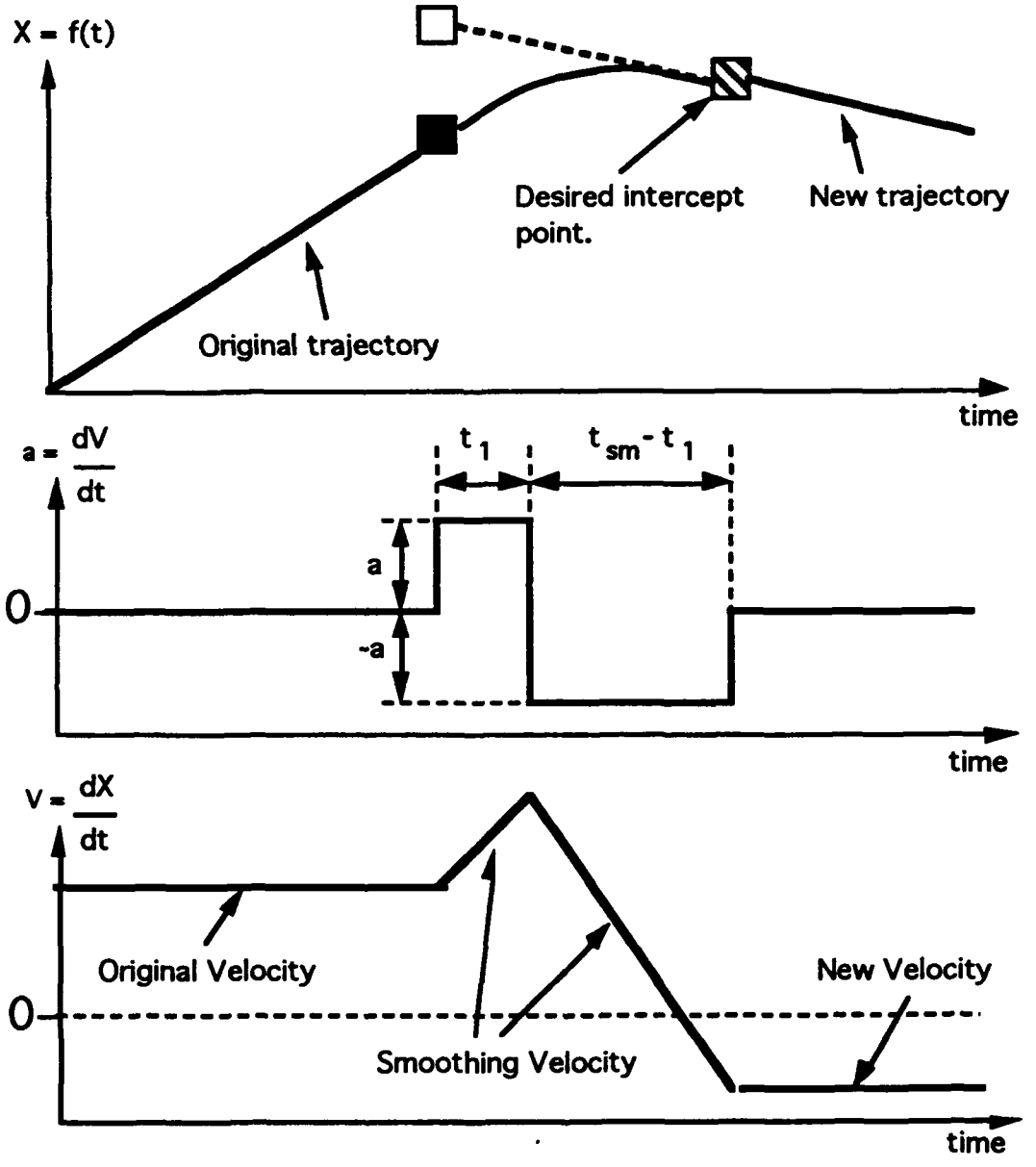


Figure 3-24 - **An Optimized Smoothing Algorithm** . By designing an acceleration profile as shown, both position and velocity can be intercepted. The eye will not see any instant position changes, or instant velocity changes. Step changes in acceleration correspond to the onset of a constant force, something the eye and brain are accustomed to. Keeping the acceleration magnitude the same for both positive and negative steps, yields the lowest possible acceleration magnitude for a given smooth time.

3.4.3.2.2. Minimizing Position Error While Smoothing

The original SIMNET smoothing algorithm, and the newly proposed optimized smoothing algorithm, both reduce the error between smoothed position and dead reckoned position to zero over time. Figure 3-25 shows the SIMNET smoothing position trajectory and the error in position during smoothing. Assuming no network delay, the maximum error is the threshold at which a packet is issued, and the minimum error is zero.

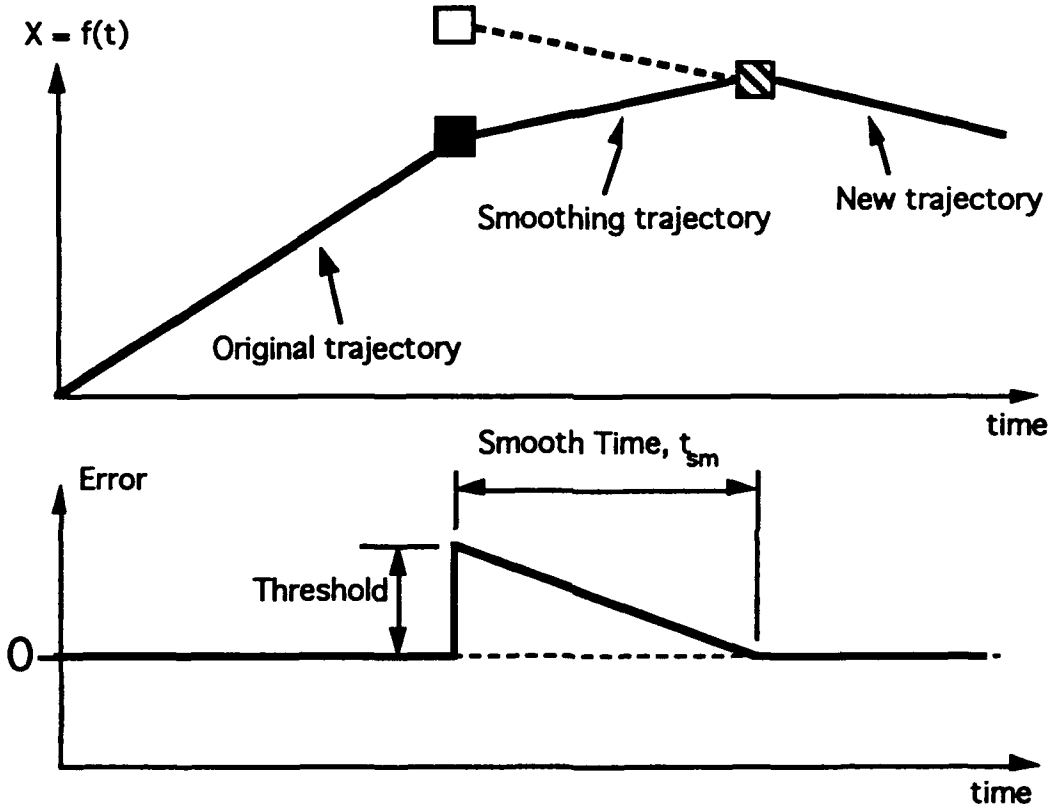


Figure 3-25 - **Error Between Smoothed Trajectory and Dead Reckoned Trajectory for SIMNET Smoothing.** Because the smoothing velocity is linear, and the dead reckoned velocity is linear, the distance between the two trajectories converges linearly over *tsm*. After the smoothing is finished, the error is zero until the next appearance packet is received.

As alluded to earlier, this situation is aggravated by appearance packets arriving before the smoothing algorithm is completed. Figure 3-26 shows the effect of packets arriving earlier than expected for the SIMNET smoothing algorithm. When the first packet is received, the smooth time, t_{sm}, is chosen, and the vehicle traverses the first smoothing trajectory. The second packet arrives before the first smoothing trajectory intercepts the dead reckoning path. From its current position in the middle of the first smoothing trajectory, the vehicle must intercept the second packet's dead reckoning trajectory. Since the error between

the first dead reckoning trajectory and the second packet position is, by definition, the threshold, there is an additional position error, X, which now separates the vehicle from the first dead reckoning trajectory. The total error is then *Threshold* + *X*. The second graph in Figure 3-26 shows position error as a function of time.

To guarantee that position error never exceeds the threshold, t_{sm} must be selected such that the smoothing algorithm always finishes before the next packet arrives. Since SIMNET uses a first order dead reckoning algorithm, the packet rate was a function of the acceleration of the vehicle. It was found that the time between two packets did not vary very much from the time between the two most recently received packets. A second order dead reckoning algorithm will be somewhat harder to predict since the difference in acceleration with respect to time (jerk) may not vary as smoothly as acceleration.

There is a tradeoff between selecting a very short t_{sm} and the perceived "smoothness" of the trajectory. Figure 3-26 shows the "optimized" smoothing acceleration profile, presented in the previous section, for two different values of t_{sm}. As t_{sm} decreases, acceleration increases (refer to equation (7)). At a certain value of t_{sm}, acceleration will exceed a level which is believable by a human observer. This maximum level should not be exceeded, since it will result in an unreal looking dynamic situation. It is better to suffer some additional position error than to present an unreal picture to the user. Since the remote simulation is unable to exceed the performance envelope of the vehicle, maintaining a cap on the smoothing acceleration will not result in accumulated position error after several packets.

Any smoothing algorithm will have the tradeoff of smoothing time versus vehicle acceleration. Intuitively, if the vehicle gets there sooner it must move there faster. On a case-by-case basis, acceleration must be balanced with requirements for guaranteed maximum position error.

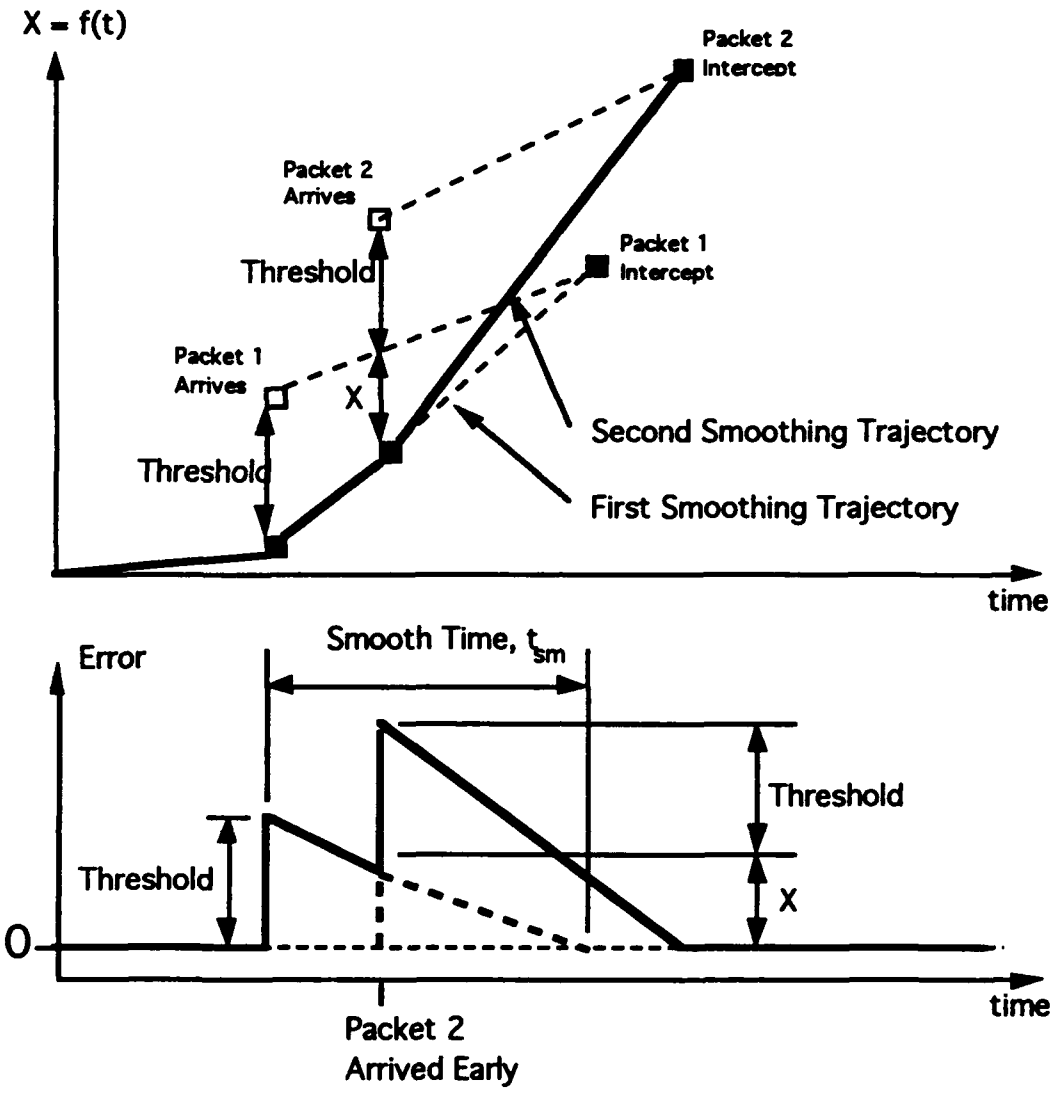


Figure 3-26 - **Error Between Smoothed Trajectory and Dead Reckoned Trajectory for SIMNET Smoothing When Packet Arrives Early.** Packet 2 arrived before the smoothing trajectory could reach the intercept point. The position error is *Threshold + X* . If this happens repeatedly, the error will continue to grow. For this reason it is very important to choose *tsm* sufficiently small such that the smoothing has a chance to finish before the arrival of the next packet.

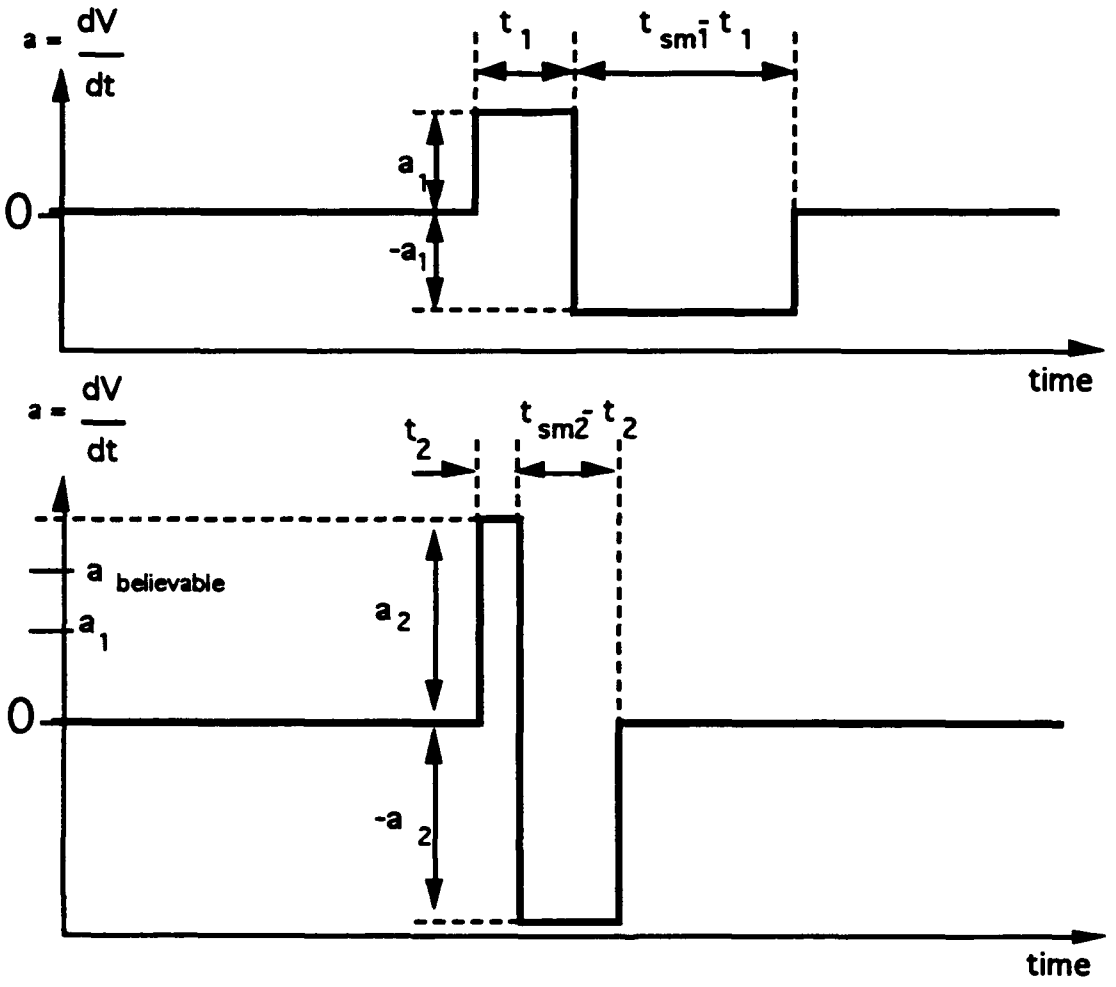


Figure 3-27 - **Tradeoff between acceleration and Smooth Time** . To insure that position error does not accumulate due to prolonged smooth time, it is desireable to shorten the smooth time such that smoothing is finished before the next packet arrives. Equation (7) shows that if *tsm* is reduced, acceleration will increase. Acceleration can increase until it reaches a believability threshold, which is exceeded in the second graph.

3.4.3.3 Conclusions and Recommendations

Smoothing is a mechanism by which dead reckoned entities transition from a previous position,velocity, and acceleration, to an updated set, eliminating error between the two states over time. There are many algorithms which can reduce error in systems. Typically these algorithms, such as optimal feedback control systems, are designed to reduce error as quickly and accurately as possible. In Distributed Simulation systems a more overriding concern is the believability of the system and its effect on the goals of the simulation. In light of this observation, the main requirements for smoothing algorithms are:

- The smoothing should look realistic, should not be visually noticeable, and should not distract the user.
- The position error between the smoothed entity and the straight dead reckoned entity should be minimized.

Guidelines for evaluating potential smoothing algorithms are:

- Accelerations on an entity should be minimized during smoothing.
- Unpredictable packet arrival time should not induce cumulative position error.

With these guidelines, smoothing algorithms may be quantitatively, as well as qualitatively evaluated.

3.4.4 Timebase Correction

This algorithm corrects the visual anomalies caused by different quantizations of time, due to different base frame rates at sender and receiver. See figure 3-28.

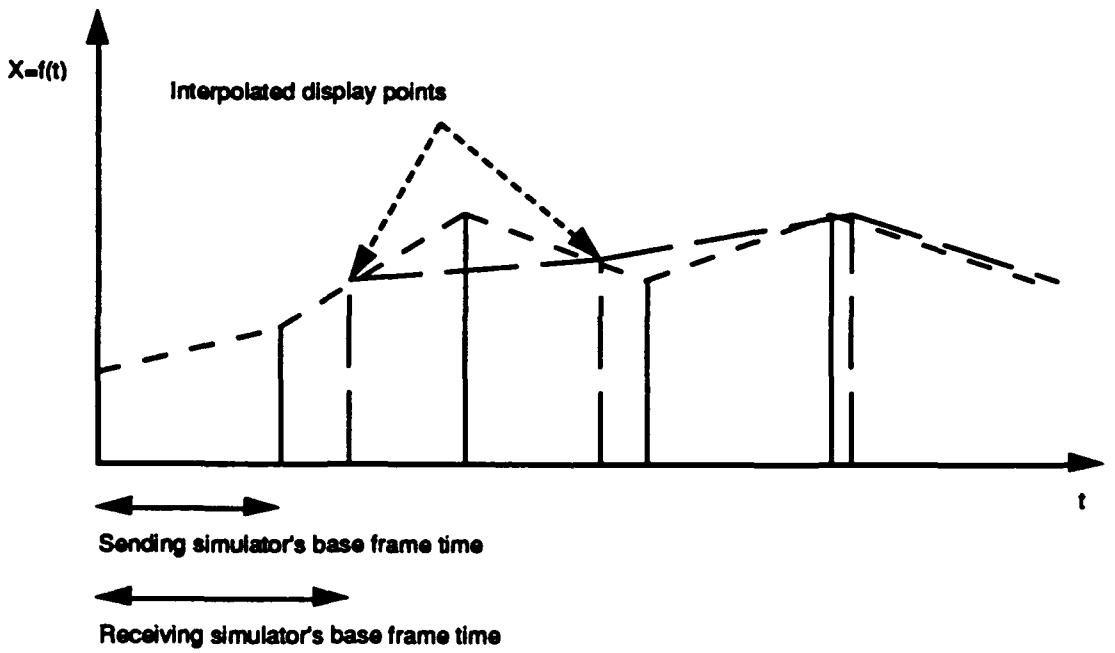


Figure 3-28: Timestamping

Relative timestamping, defined in later versions of SIMNET and in DIS, is sufficient to define the track as a function of time, and thus allow the receiving simulation to determine where to display the entity.

3.4.5 Forward Reckoning Algorithms

As stated earlier in this section, latencies associated with network delay and internal processing, cause events to be displayed to the user which are outdated by some small period of time. Some simulator makers have proposed a concept known as "forward reckoning" to compensate for this delay. Entity states are extrapolated into the future, and display devices are primed with this predicted image, such that the events are presented to the user at approximately the same time as they occur. Figure 3-29 shows the problem that forward reckoning tries to solve, and Figure 3-30 shows the forward reckoning algorithm. At first glance this seems to be a clever way to compensate for network and processing latency. There are some artifacts created by this paradigm, however, which make it less appealing.

A good way to analyze the effects of forward reckoning is to take a sample case, and compare the actual trajectory of an entity to a normal dead reckoned display and to a forward reckoned display. By plotting position as a function of time for each of these cases, the time and position errors can be compared. For this example we will examine a missile, initially at rest, fired at a stationary tank 50 meters away. The missile undergoes a constant acceleration of 100 m/s^2 for a period of 1 second before it hits the target. The dead reckoning model will be second order, and the threshold at which a new packet is sent is 1 meter. Figure 3-31 shows the actual trajectory of the missile as simulated in its host computer. We will assume a total latency, t_3, of 250 milliseconds. We will also assume, for this analysis, that the simulation frame time is small compared to the latency, so that effects of a discrete frame will not be a factor.

The first task in the analysis is to determine at what times appearance packets are issued by the missile simulator. At time 0, the missile begins to accelerate. The time at which the threshold is exceeded can be computed with the equation:

$$s = \frac{1}{2} a t^2 \qquad (1)$$

Where s is the threshold (1 meter), a is the acceleration (100 m/s^2), and t is the time at which the threshold is reached, computed to be 0.14142 seconds. The first packet, which is transmitted at time t, contains:

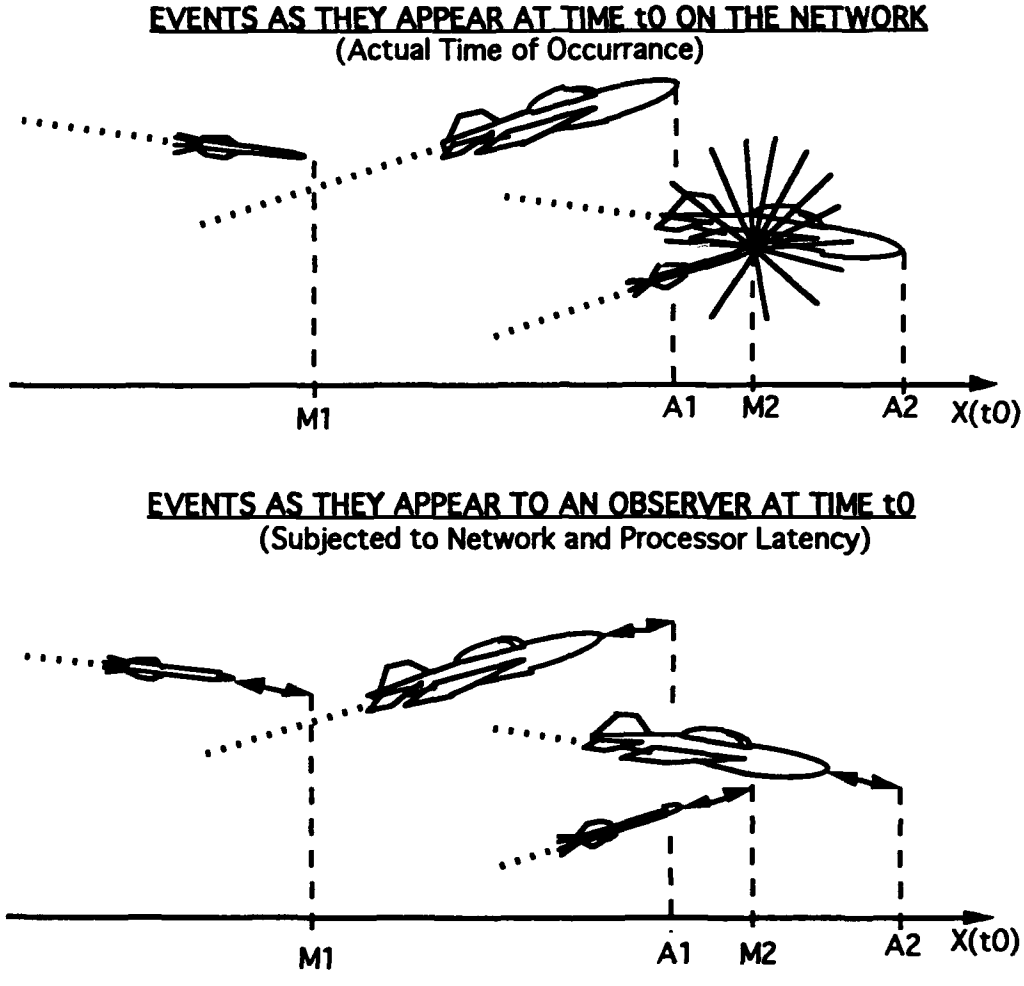


Figure 3-29 - **Effect of Network and Processing Latency on Correlation of Events in Time** . The top figure shows events as they occur, or rather as the events are reported on the network by transmitting entities at time t0. The bottom figure shows the scene as an observer would see it from an observing entity. The information in the observed scene is not the information currently on the network, but the information that appeared on the network some time ago that finally propagated its way through the system. The double arrows indicate the position error between the displayed scene and the actual position of the entities at time t0. Note that the explosion event has not yet been displayed by the observer.

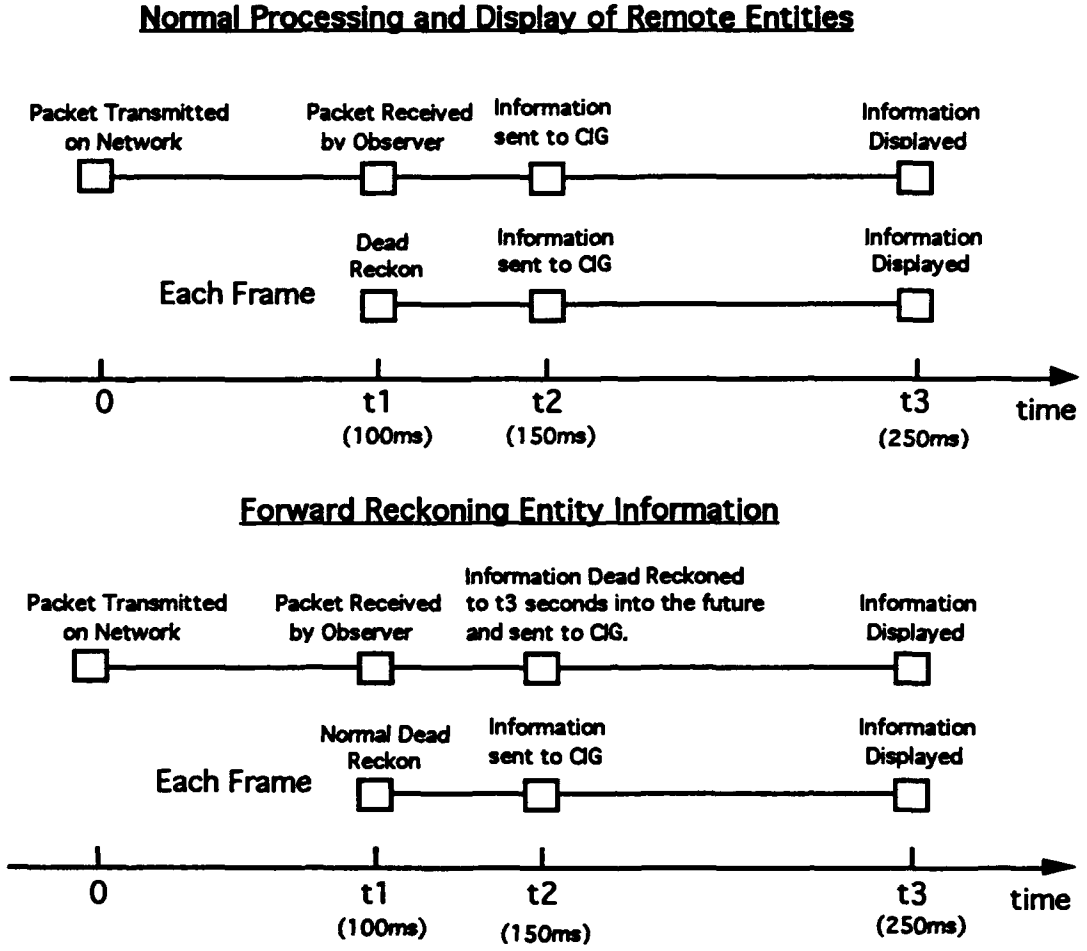


Figure 3-30 - **Forward Reckoning.** The top timeline shows that by the time entity position information is displayed to the user, it has been delayed by t3 seconds from the time it was put out on the network. The bottom timeline shows the forward reckoning paradigm. Upon receipt of a packet, the entity is dead reckoned t3 seconds into the future before being sent to the CIG. When the entity is displayed it should be closer in space to its actual position. The entity is dead reckoned in the normal way from that point onward.

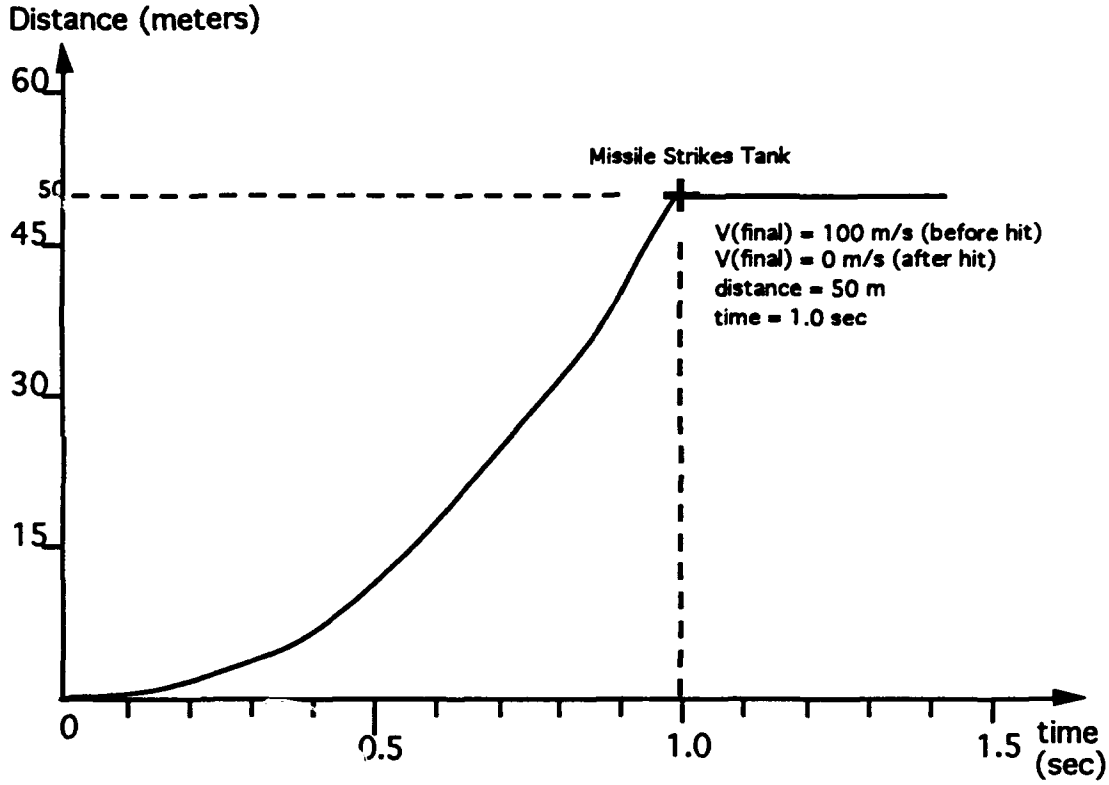


Figure 3-31 - **Actual Trajectory of Test Case Missile.** This is the trajectory of a missile, as it is perceived by the computer simulating it. The missile, initially at rest, accelerates at 100 m/s/s, for 1 second, when it hits a stationary tank. Its velocity goes to 0 after the hit.

position = 1 meter

velocity = 14.142 m/s

acceleration = $100 m/s^2$

Since the acceleration is constant, the second order dead reckoning model will match the actual trajectory exactly, up until the time the tank is hit. When the tank is hit, the missile state is:

position = 50 meters

velocity = 100 m/s

acceleration = $100 m/s^2$

The missile velocity goes to 0 after the hit, so the second packet will be transmitted when the dead reckoned model puts the missile at the 51 meter mark. Using equation (1) with a value of 51 for the distance, the time is computed to be 1.00995 seconds. The parameters in the second packet will be:

position = 50 meters

velocity = 0 m/s

acceleration = 0 m/s^2

Knowing what the network packets contain, and knowing when they are transmitted, the dead reckoned and forward reckoned trajectories can be constructed. Figure 3-32 shows the actual trajectory of the vehicle alongside the computed position at the receiver's host, and the final displayed trajectory, for normal dead reckoning. The first packet is transmitted at 0.1414 seconds and is received at *t1* + 0.1414. This causes a 1 meter jump in the position of the missile. The computed missile trajectory then follows the actual trajectory with a lag of *t1*. The displayed trajectory is exactly the same as the computed trajectory, with an additional lag of *t3* - *t1*, or 150ms. The total effect of latency and dead reckoning is a time lag of 250 ms and two 1 meter jump discontinuities in the.displayed trajectory. This solution represents the worst case with respect to time lag, but the best case as far as position accuracy. The largest position error that will occur is the threshold.

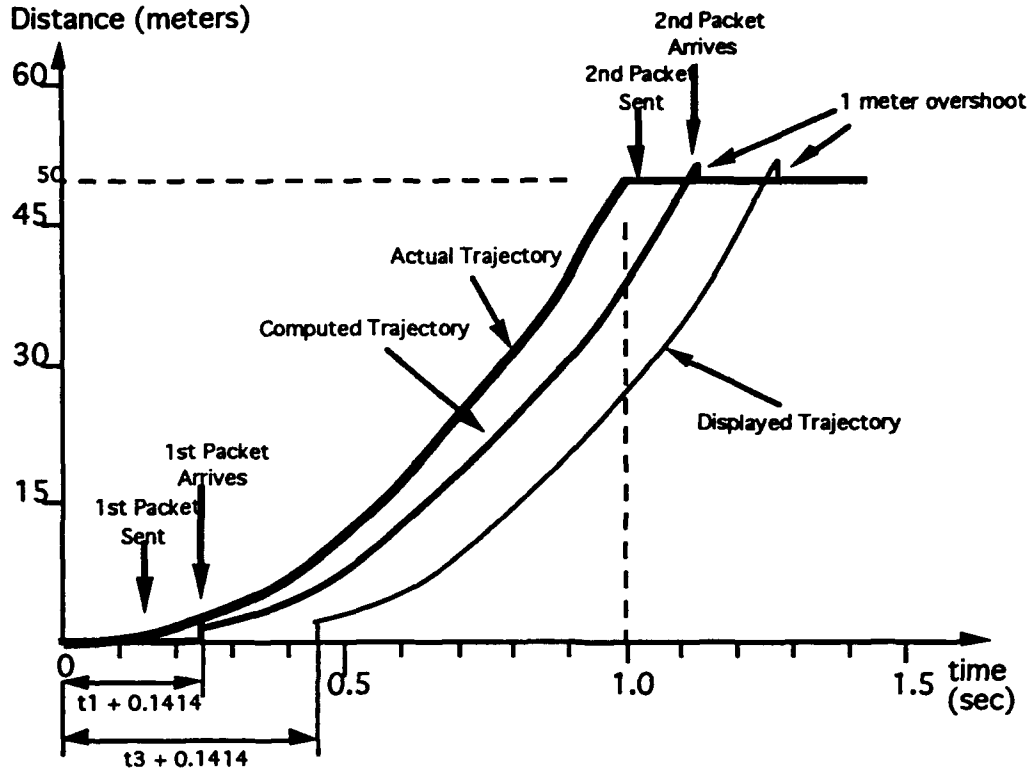


Figure 3-32 - **Displayed Position of Missile as Compared to Actual Trajectory for Normal Dead Reckoning** . The computed trajectory has a latency of *t1*, which is the network latency, with respect to the actual trajectory. The first packet is sent at 0.1414 seconds, so it does not arrive until *t1* + 0.1414, causing the sharp 1 meter correction in the computed trajectory. The displayed trajectory is exactly the same as the computed trajectory, with a time lag of *t3* - *t1* , or 150ms in this case. The total time lag from the actual trajectory to the displayed trajectory is *t3*, or 250 ms. The positional error, however, is never more than the threshold, which is 1 meter.

Figure 3-33 shows the displayed forward reckoning trajectory next to actual and computed trajectories. When the first packet arrives at *t1* + 0.1414 (241.14ms), the simulation host attempts to compensate for the network and processing latency of 250ms by extrapolating the kinematic information in the packet. The packet information is 100ms old due to the network, and there is another 150ms delay before the image is displayed. By using the equations:

$$s = s_0 + V\,t + \frac{1}{2}a\,t^2 \qquad (2)$$

$$V = a\,t \qquad (3)$$

and the packet parameters:

$$s_0 = 1 \text{ meter}$$

$$V = 14.142 \text{ m/s}$$

$$a = 100 \text{ m/s}^2$$

the predicted position and velocity, computed for display at 391ms (241 + 150), are:

$$s = 7.66 \text{ meters}$$

$$V = 39.14 \text{ m/s}$$

Instead of a 1 meter threshold jump discontinuity as experienced in normal dead reckoning, the forward reckoning has exaggerated it to 7.66 meters. From the 391ms mark to the 1 second point the forward reckoning trajectory faithfully follows the actual trajectory of the missile in both space and time. When the 1 second point is reached, however, the actual model diverges from the forward reckoned model. The forward reckoning algorithm will not hear about the divergence until the 1 meter threshold is exceed, and after a 100ms network delay. This total delay is 109ms after the impact, at the 1109ms mark. By this time, the image generator has been primed with the state of the missile as it was predicted for the 1250ms mark. Using equation (1), with a time of 1250ms, we find that the image generator was primed with a distance of:

$$s(1250\text{ms}) = 78.125 \text{ meters}$$

This is an error of over 28m, or 28 times the dead reckoning threshold.

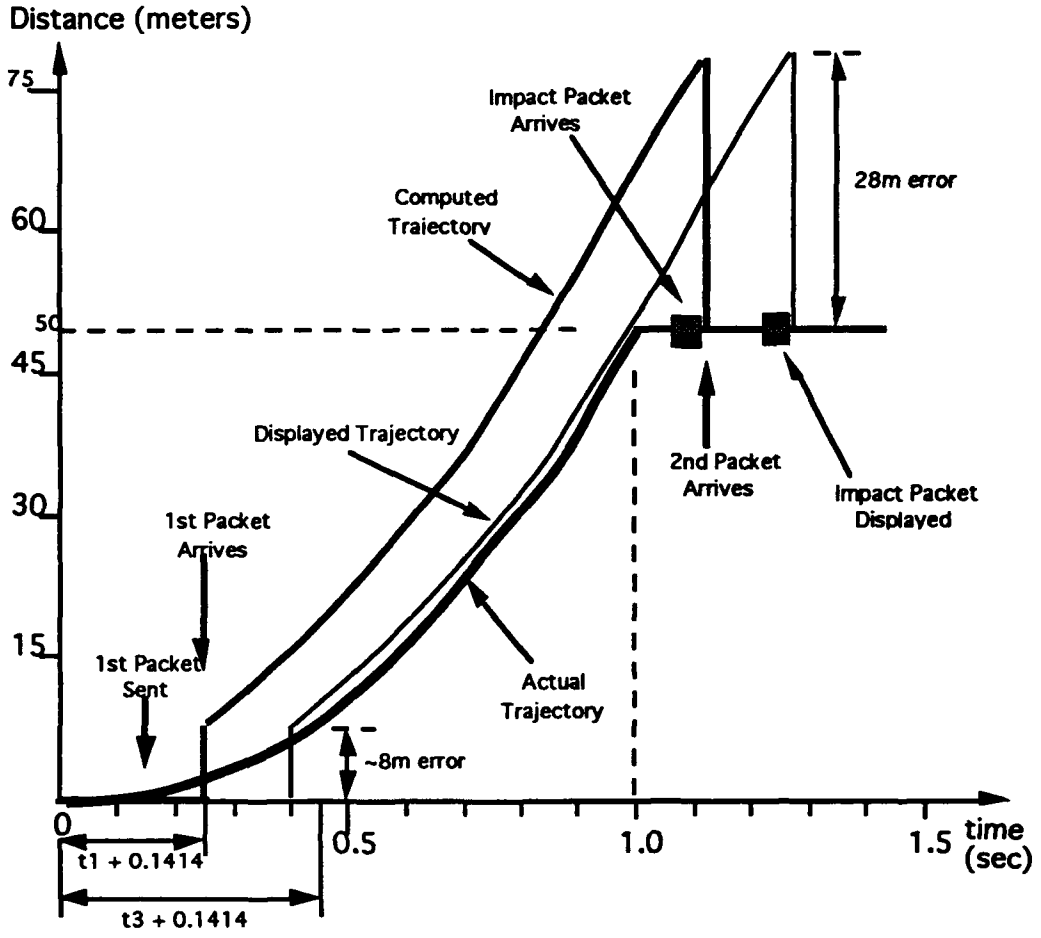


Figure 3-33 - **Position and Time Error in Forward Reckoning.** When the first packet arrives, the receiving host computer extrapolates the kinematic information 250ms into the future to compensate for network delay and processing delay. For our missile, the initial position jump is 7.66m. The bulk of the displayed trajectory does coincide with the actual trajectory in both time and space. By the time the second packet arrives, however, the CIG has been primed with a very inaccurate position. The missile will be displayed 28m beyond the impact point.

Another artifact of forward reckoning is the discorrelation of discrete events with forward reckoned entities. Figure 3-33 shows that just before the second missile appearance packet is issued, an impact packet is issued. Being that impact packets are discrete events, they cannot be predicted in advance, nor can they be easily attached to the entity that caused it. The explosion will be painted at the 50 meter impact point just before the missile is painted at the 78 meter point. Figure 3-34 shows what the simulated picture might look like from above if both the missile and tank were in motion and the observer was forward reckoning both entities.

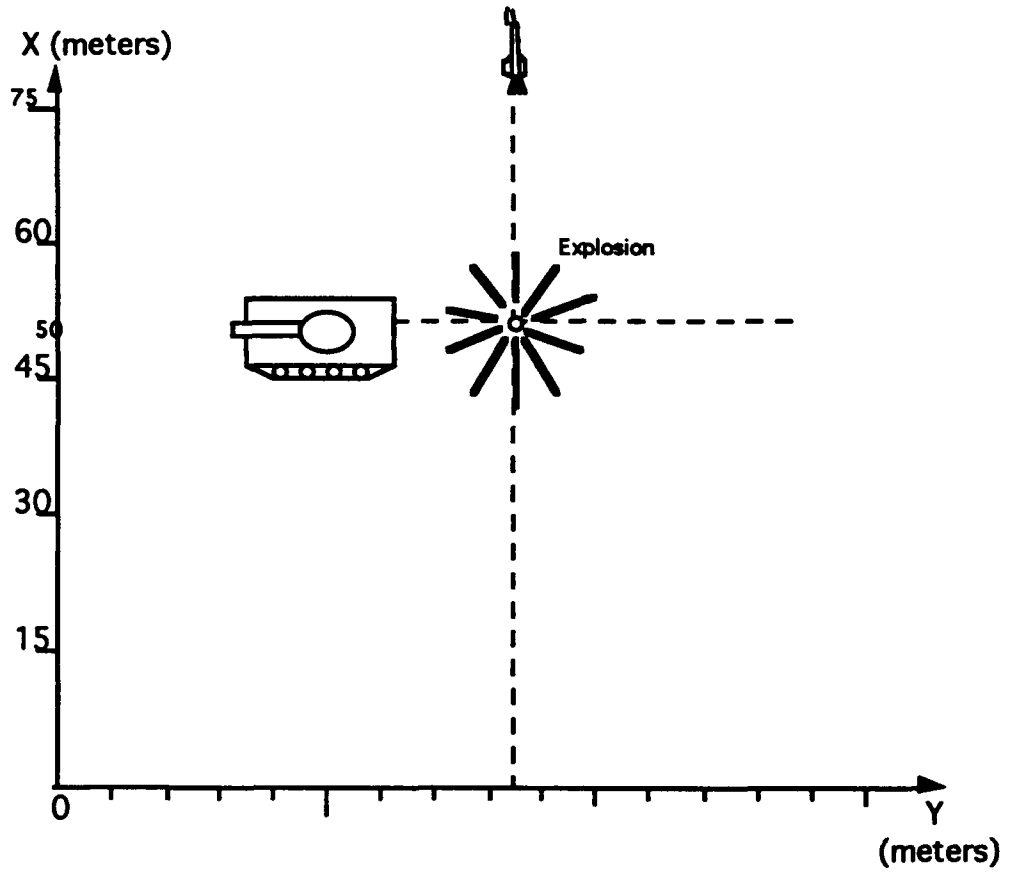


Figure 3-34 - **Discorrelation of Events and Entities Due to Forward Reckoning** . An observer forward reckoning the missile and the tank will see the missile pass through the tank at the impact point, fly an additional 28 meters, then see the explosion at the point in space where the two entities collided. The tank and missile will then have to be pulled back to their collision point.

The missile example is an extreme case of undesirable behavior due to forward reckoning. In general, the error between a forward reckoned position and a straight dead reckoned position, as measured when a new packet arrives, is given by:

$$\Delta s = s_0 + \Delta V \, t_{fr} + \frac{1}{2} \Delta a \, t_{fr}^2 \qquad (4)$$

Where s_0 is the dead reckoning threshold, ΔV is the velocity difference between the standard dead reckoned velocity at the time of packet receipt, and new velocity from the packet, Δa is the acceleration difference between previous acceleration and new acceleration from the packet, and t_{fr} is forward reckoning time (250ms in this case). Note that when $t_{fr} = 0$ (no forward reckoning), Δs becomes the threshold.

Forward reckoning provides a mechanism for displaying events to a user in a remote distributed simulation node that more closely correlate to their actual occurrence in time. This prediction mechanism, however, causes two artifacts which degrade the simulation. Forward reckoned entities can deviate from their true trajectory by a distance much greater than their threshold, and forward reckoned entities become discorrelated in space and time with discrete events and other forward reckoned entities. These two artifacts jeopardize the believability of the simulation and degrade the effectiveness of the system.

3.4.6 Object Handover

Network delays and processing delays create latencies between the time when events actually occur, and the time when they are displayed to a user in a remote simulation node. The importance of the correlation of these displayed events with the time and position of actual occurrence is a function of the simulation task. For Distributed Simulation, we assert that it is more important, in general, for a user to see a cohesive, plausible view of the world, than it is for displayed events to correlate in time with their occurrence. In other words, we'd rather the user see a well coordinated picture of what happened a fraction of a second ago, than for him to see an uncoordinated, disjoint view of what's happening at this very moment.

There are times, however, when an activity must be presented to the user in a more timely fashion for accomplishment of the simulation task. One example of this situation is the terminal guidance of weapons.

Figure 3-35 shows the problematic situation of terminal guidance for a missile being simulated by a launching node, and a target simulated by a remote node. From the standpoint of the airplane node, the missile is seen at a distance which is not current. The most recent information from the missile node has been delayed by the network and processing, so it appears at a position it occupied sometime in the recent past. The airplane, however, sees itself at its current location. The apparent distance between the two is larger than it actually is. The missile has the same problem. It sees itself at its current correct position, but the airplane is occupying a position in its recent past. The net effect is that the missile will decide on its impact with the plane long before the plane needs to make its final maneuver to escape. This is unfair to the pilot and jeopardizes the simulation's credibility.

A solution to this problem is to "migrate" the missile from the firing node to the airplane node. This entails packaging up the missile's state, and sending it to the airplane so the airplane node can simulate it with much greater accuracy. This operation is graphically depicted in Figure 3-36.

There are many other reasons to transfer ownership of objects from one node to another. A general paradigm for the transfer of objects in Distributed Simulation is warranted.

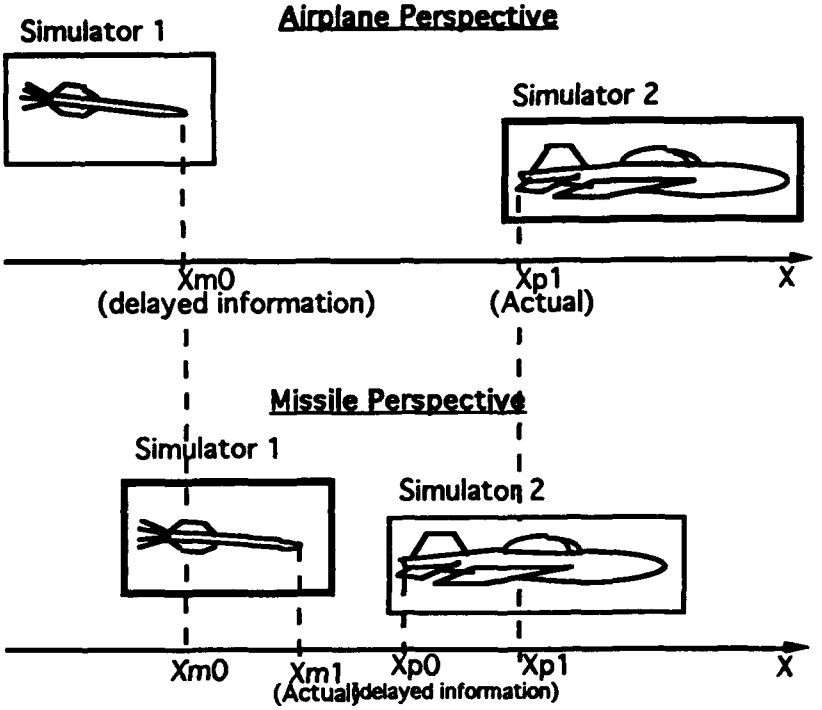


Figure 3-35 - **Latency Problem in Terminal Guidance.** The top picture shows the impression of the state of both entities from the airplane's point of view. The airplane sees itself in its actual position, while the missile is rendered further away due to network and processing delays. The second picture, however, shows that the missile sees itself in the correct position but sees an old version of the airplane position. The missile will decide it has hit the airplane before the airplane has a chance to make a succeful final avoidance maneuver.

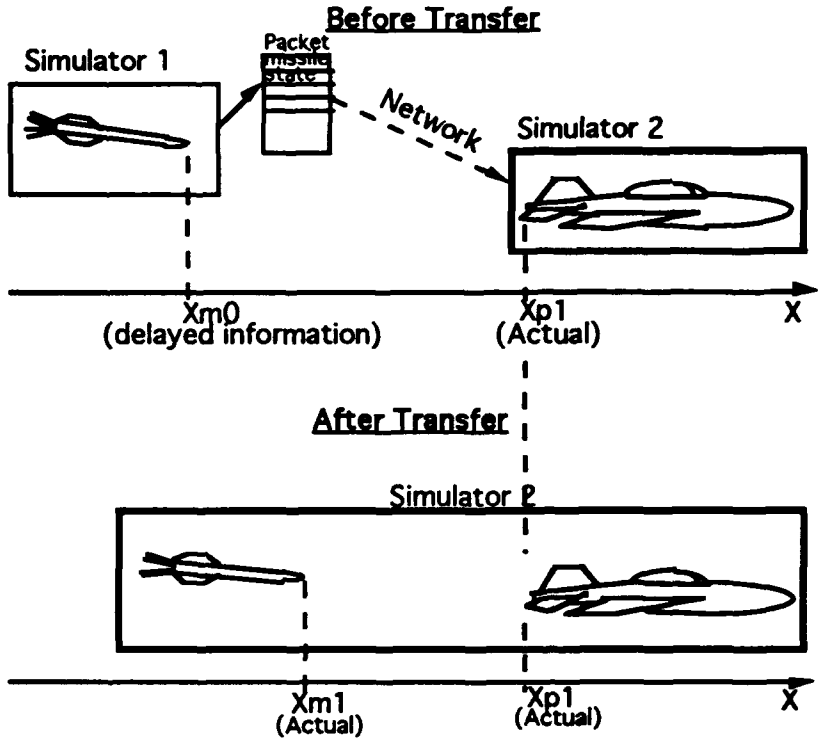


Figure 3-36 - **Object Migration Solves Terminal Guidance Problem.** By packaging up the missile state and sending it to the airplane node across the network, the airplane node can simulate the airplane as well as the missile. All problems associated with network and processing latency disappear when both entities are simulated on the same node. This is a general solution for many problems which require a tighter coupling between two remote entities.

4. Visual/Sensor Correlation

The ability to correlate visual and sensor simulations is a primary factor in determining the value of the simulation for its intended application. For example, in training aircrews, high fidelity Weapon System Trainers (WSTs) are required to correlate the Visual scene with Radar displays, the Electronic Warfare (EW) environment, Electro-Optical (EO) sensors, and auditory cues. Each of these sensory inputs is typically provided by an independent simulation operating on a unique local database. Figure 4-1 is a top level diagram illustrating the major visual/sensor components in a typical WST and the corresponding data base generation processing that is performed for the WST.

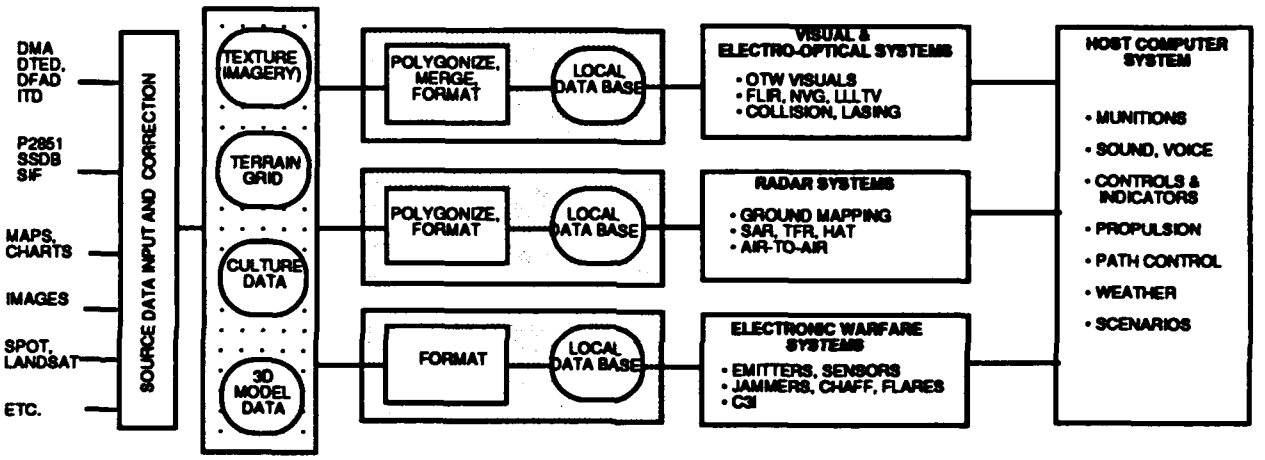


Figure 4-1: Typical WST Configuration and DBGS Processing

In the case of WSTs, correlation is usually considered in the specification and design of the overall system. Thus the individual simulations are designed from the outset to provide a high degree of correlation with each other and are tested to insure that the necessary degree of correlation exists.

For the general DIS application, however, that will not be the case. DIS is required to address issues associated with determining the level of correlation between systems that are specified and built independently of each other. Further, the systems will be built with incomplete knowledge of the capabilities of the other systems potentially involved in large-scale simulation exercises. In fact DIS exercises may involve multiple implementations for the same type of cue. For example, different visual systems using different databases and display systems may be involved. Thus DIS will not only have to address correlation across different visual and sensor systems, but also correlation between different implementations of the same simulation system.

As suggested by the referenced figure, sensor data correlation is a function of the correlation of the local databases and the sensory processing/rendering

systems. The databases are dependent on the source data, modeling tools, modeling fidelity and accuracy, transformation algorithms, and transformation parameters. The sensory processing/rendering systems are dependent on the processing/rendering models or algorithms implemented in the various simulation devices. Note that this fits the correlation model introduced in paragraph 2.5.

The following sections discuss some of the problems caused by visual/sensor inconsistencies. A major focus is given to the terrain aspects of the problem, both from a data base standpoint and a processing standpoint, since terrain is central to visual and sensor simulation systems. We then describe a correlation measurement methodology using the terrain example. Finally, we show the relationship of the visual/sensor correlation problem to other correlation issues in the context of the proposed DIS Data Base Standard that is described in Section 4 of Volume I of this document.

4.1 Effects of Visual/Sensor Inconsistencies

Lack of correlation between visual and sensor simulations can cause several effects that can be observed by the exercise participants and may affect the outcome of the exercise. The following sections address several of the anomalies that can occur as the result of uncorrelated visual/sensor cues. We discuss the problems caused by terrain representation differences, visual scene processing/rendering differences, high resolution sensor simulation, electronic warfare simulation, and environmental effect simulation.

4.1.1 Terrain

Terrain representation and rendering has been one of the more intractable challenges to visual simulation system designers. The problem, simply stated, is the reduction of terrain detail with increasing range without the introduction of artifacts in the displayed scene. Early solutions bypassed the problem entirely by requiring fixed terrain data bases with no changes allowed in real-time (the SIMNET solution). Later solutions introduced the concept of multiple, discrete terrain Levels of Detail (LOD) controlled by real-time load management algorithms. The problems these latter approaches introduced were sometimes less than desirable - e.g., terrain transition zones where two LOD's of the terrain would co-exist, creating an ambiguity in the representation of the world, and other visual artifacts. However, load managed terrain rendering was (and is) considered essential in the world of finite and very expensive IG resources.

In recent years, with the advent of the real-time depth buffer and lower cost visual systems, it has become feasible to develop methods to continuously vary the terrain representation in real time. The continuous adaptation of the terrain results in no terrain ambiguities (no more overlap) and, if implemented correctly, virtually no visual artifacts. However, even with this level of control over the terrain surface we are left with the problem of a dynamically changing

environment, which is difficult enough to cope with in a single simulator, let alone a network of heterogeneous simulators.

Some of the more significant problems caused by terrain differences (both static and dynamic) between linked simulators are intervisibility, ground vehicles detached from the terrain surface, collision and weapon impact, as discussed below. In the discussion that follows terrain refers to the mathematical representation of the Earth's surface, including the ground and the fixed objects that reside on the Earth's surface, such as roads, trees and buildings.

Intervisibility Different representations of the terrain can result in the ability of player one to see player two when player two can't see player one, and vice versa. For example, a trench or ravine may exist in player one's data base where flat ground appears in player two's data base. Player one believes he is obscured from view, but player two sees him fully exposed.

Ground Vehicles Detached from Terrain Player one moves across the terrain surface correctly in his data base, but in player two's data base player one occasionally detaches from the terrain surface, and appears to float above and/or sink below the terrain surface. Of course, this can be avoided if player two overrides player one's vertical position data and forces it to follow his terrain surface, but it does not solve the intervisibility problem.

Collision Player one collides with a fixed structure in his data base, but in player two's data base the structure is slightly misplaced and therefore player one appears to not collide with the structure from player two's viewpoint.

Weapon Impact Player one fires his weapon at player two and the round appears to impact player two's vehicle because he has an unobstructed view. Player two observes the muzzle blast of player one, but because he has positioned himself behind a large boulder in his data base, player two believes that he has not been hit.

4.1.2 Visuals

Differences in the performance of visual systems that may be interacting in a DIS environment can compound the terrain data base inconsistencies described earlier. Examples of error sources induced by different visual systems are feature position errors, scene management induced errors, and weather/atmospheric effects.

Feature Positioning The positioning of features (buildings, trees) on the Earth's surface is a function of the precision of the calculations, the Earth reference model, and the strategy used to block the simulator data base into manageable pieces, among other things. Double precision floating point calculations are needed to achieve sub-foot positioning accuracy on the surface of a geocentric Earth. A spherical Earth model will induce significant positioning

errors compared to a WGS-84 ellipsoidal Earth model.[4] A data base blocking strategy that approximates the curved Earth with contiguous flat "plates" will induce significant feature positional errors proportional to the distance from the center of each plate.

Scene Management Scene management controls are used by visual simulators to manage the processing load such that channel polygon loading obeys prescribed min and max tolerances, to emphasize one feature type over another (e.g., terrain fidelity may be more important than cultural feature fidelity in a given situation), and in general to maximize the utility of a finite set of resources for a given application. Level of detail controls, model/polygon blending, and range limits are some of the mechanisms that are employed.

The algorithms and control strategies used to manage scene loading can have a significant effect on the relative appearance of two otherwise identical data bases - and for that matter two otherwise identical simulators. Consider the case where player one's simulator switches terrain detail on a range basis, such that terrain polygon density reduces with increasing range. Player two's simulator, on the other hand, may force terrain density to be constant at all ranges, thus sacrificing cultural feature density and/or terrain range for a given polygon budget. The rendered scene in these two cases would be quite different, even though the simulators and the data bases were identical.

Atmospheric Effects The algorithms used to simulate weather and other atmospheric effects such as fog, rain and battlefield smoke can vary from simulator to simulator. This can result in the ability to observe a given target in the weather in one simulator but not the other. See paragraph 4.1.5.

4.1.3 High Resolution Sensors

A common problem in simulation today with stand-alone WST's is the correlation of high resolution sensors with OTW Visuals. For example, Synthetic Aperture Radar (SAR) may have a range well beyond visual range, and yet exhibit a ground resolution measured in feet. Similarly, narrow field-of-view EO sensors such as FLIR's and LLLTV's may be able to see well beyond visual range under poor OTW visibility conditions (smoke, darkness) with very high resolution. Often these types of simulations are implemented by extending the OTW Visual system with sensor simulation hardware. This often mitigates the correlation problem, but it does not completely eliminate it because of the data base update rates involved with high resolution sensors, which is often exacerbated by fast sensor slew rates.

The typical work-around with today's simulators is the restriction of high resolution data to small patches of high detail. Also, significant features are tagged for inclusion in all views of the simulation system. These strategems have the undesirable effect of cueing the participant to "look for the high resolution patches". As technology advances it will be possible to increase the size of these

patches in a gradual fashion, and reduce false cueing. The off-line and real-time management of these data sets of varying size and resolution presents a significant correlation challenge.

4.1.4 Electronic Warfare

The outcome of modern battlefield engagements is becoming increasingly reliant on effective use of the electromagnetic spectrum. For this reason extensive use of ECM and ECCM techniques have become an integral part of any comprehensive battle plan. Consequently the EW (Electronic Warfare) portion of simulation efforts has also increased in scope and in priority. Four aspects of the EW simulation problem are mentioned below. Additional information is provided in section 7.2.

First, in considering electronic warfare systems, it must be recognized that faint signals must be considered and that a very significant amount of data on the signal structure and the spatial/temporal characteristics of the beam are required. Hence antenna patterns and signatures must cover a sizeable fraction of 4 pi space. At the same time, many radiations exist, including numerous ones from highly complex emitters (jammers, advanced radars, etc.).

Second, regardless of the update rate on information between linked simulators, there are always processes in EW which are significantly faster than any update rate (up to 10 KHz rates would be required if a brute force approach were used). At the same time there are long, complex actions which can adequately be described by defining action, initiation time, and electronic dead reckoning.

Further, EW simulation does not permit an approach in which a sensor communicates only with its target entity. Every entity with an intercept receiver is a potential receiver of every signal. The path from every emitter to every appropriate receive entity must be considered to determine whether the received signals are above the thresholds of the equipment when considering weather and such terrain effects as ducting and diffraction (if these are part of the simulator).

Still a third part of the problem lies in the effect of weather and natural environment effects on the electromagnetic elements. Advanced simulators will contain a complex weather model, varying in three dimensions and time. This weather produces an appreciable effect on electromagnetic propagation which can be unique between each pair of entities even if they are at the same range (so that the r-squared losses are identical).

This imposes a coherence problem between classes of simulators similar to that between simulators with different terrain fidelities. In the terrain case, visibility (radar or optic) is different for the two entities in different simulators because one simulator recognizes a level of detail which includes an occulting feature. In the weather effects case, the problem is that of differential visibility in

the RF, IR, or EO area due to intervening weather which is considered in only one of the simulators.

Finally the fourth part of the electromagnetic problem lies in the coherence problem to be found in linking high fidelity simulators with electromagnetic effects such as ducting and diffraction with simulators which either do not consider such effects, or which use a different level of fidelity in their terrain simulations. In either case, a similar occulting/obscuration problem exists.

4.1.5 Environment

Environment effects include weather, changes to the terrain (dynamic terrain), and weapon effects, such as smoke, dust, chaff, and flares. Table 4-1 tabulates some of the major environmental effects on visual and sensor simulation systems.

ENVIRONMENTAL EFFECT	OTW VISUALS	EO (FLIR, LLLTV, NVG)	RF (Radar, Elint, Jammers, Chaff)	INFRARED (Sensors, Jammers, Flares)
TEMPERATURE, HUMIDITY, SALINITY	✔	✔	✔ (ducting)	✔
PRECIPITATION (Reflection and Attenuation)	✔	✔	✔	✔
HAZE, FOG, CLOUDS, DUST, SMOKE, ETC.	✔	✔	✔	✔
SUN, AMBIENT LIGHT	✔	✔		✔
RF JAMMING & CHAFF	✔ (chaff flash)	✔ (chaff flash)	✔	
IR JAMMING & FLARES	✔	✔		✔
WEATHER CELLS (4 dimensional)	✔	✔	✔	✔
BLAST/FRAG	✔	✔		✔
TERRAIN	✔ (intervisibility)	✔	✔ (diffraction)	✔

Table 4-1 Environmental Effects on Simulation

Weather Complex weather is a reality in modern simulators and has a significant impact on DIS. The basic problem resides with the data base. The problem is significant since most gaming areas are large and a complex weather model is four dimensional. Weather not only varies from point to point in three dimensions but will have weather cells which move in or across the gaming area.

A secondary problem lies in retaining temporal and spatial coherence, particularly for violent, moving weather cells which can have major impacts on IR, EO, and RF.

Weapon Effects Weapon effects are similar to weather effects, the primary difference being that they are man-made. Weapon effects include dust, smoke, flares, chaff, flame, explosion, tracers, rotor wash, sea state, and dynamic shadows. They exhibit many of the same properties as weather effects. The simulation of these effects can vary widely from simulator to simulator.

Dynamic Terrain When a weapon detonates on the ground it changes the terrain - craters are formed, buildings are destroyed, and dams burst. Berms may be erected or trenches dug for defensive purposes. The occurrence of these transient changes in the environment during an exercise is referred to as dynamic terrain. Within the DIS environment we will be faced with the task of determining how differing approaches to implementing dynamic terrain can be integrated into the same exercise. This topic is discussed further in section 7.1.

4.2 Terrain Data Processing

Terrain data processing from off-line data preparation to real-time processing and rendering is now discussed, primarily in the context of visual systems. This will lead to a description of a proposed correlation measurement methodology.

Figure 4-2 illustrates the processing flow experienced by terrain data from the acquisition of source data in grid form to the intermediate polygonal mesh form of the Local Data Base and finally to the rendered result as viewed by the warfighter on the simulator display. The gridded terrain data is similar to Project 2851 SSDB terrain data; in DIS terminology it is referred to as the BATTLEFIELD Data Base. The processing that is performed on this gridded terrain data by the Off-line Data Base Processing and Real-time Simulator Processing devices is controlled by standardized specifications, called SIMWORLD data, for the off-line and real-time systems. The intermediate form of the terrain data, the polygon mesh in IG specific format, is referred to as the simulator Local Data Base.

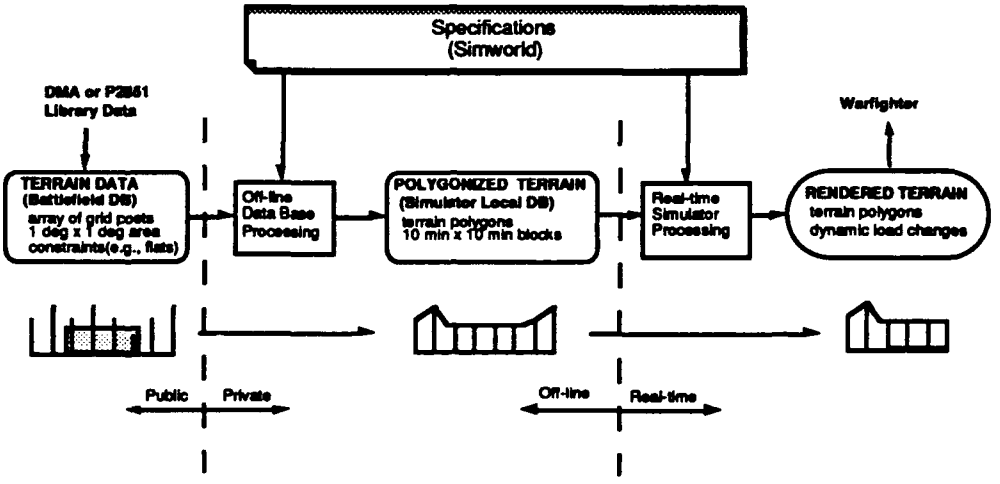


Figure 4-2: Terrain Data Processing

4.2.1 Off-line Data Base Processing

The processing that occurs during the off-line data base generation process is illustrated in figure 4-3. The terrain data flow is a horizontal slice through the overall data base generation process; this is shown as a shaded area in figure 4-3. A terrain grid constrained by topographic features such as water bodies or ridge lines is proposed as the public, standardized form of the terrain data. The terrain grid is then processed, resulting in a polygon mesh that approximates the grid constrained by topographic features.

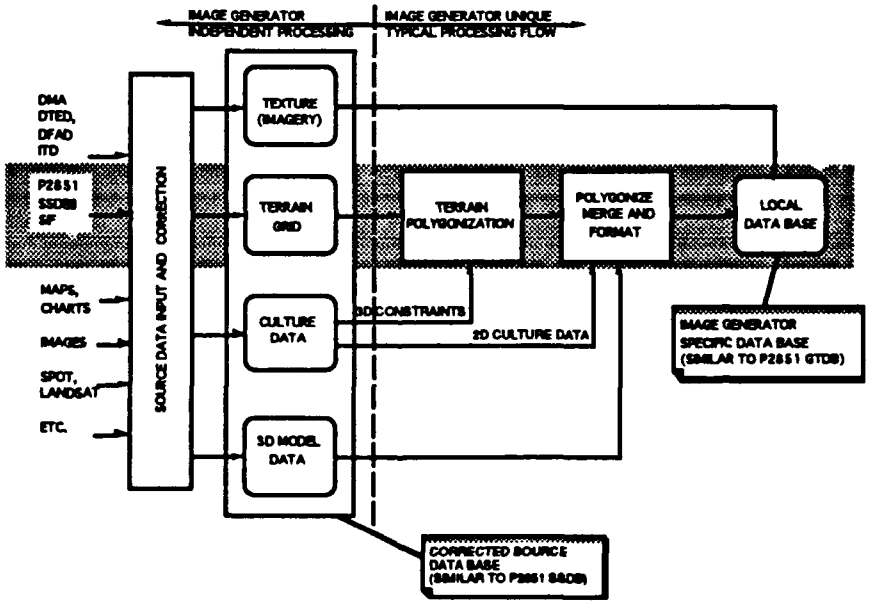


Figure 4-3: Typical Data Base Generation Processing Flow

4.2.2 Real-time Simulator Processing

Figure 4-4 illustrates a typical real-time visual system processing flow. Terrain polygons are culled based on field of view and range, discarded based on projected size, orientation, or occlusion by another feature, and transitioned from higher to lower detail representations by Level Of Detail (LOD) and range controls. The net result is that the terrain surface dynamically changes in real-time; the number of terrain polygons that survive to the display continuously varies under the control of the real-time load management controls. Therefore, the load management algorithm will bear heavily on the ultimate interoperability of a given visual system.

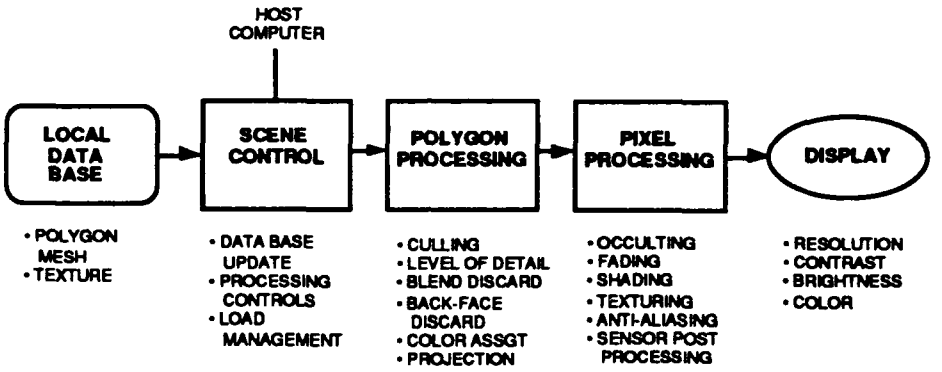


Figure 4-4: Typical Real-time Visual System Processing Flow

4.3 Correlation Metrics and Interoperability

The overall framework for developing and testing terrain correlation metrics and for determining interoperability is now presented; see figure 4-5. Data is captured at three points: source data in grid form, local data base data in polygon form, and the end result of the rendering process in pixel form. Correlation metrics are required to compare the local data base with the source data, and to compare the processing results with the local data base. These metrics are then used to determine interoperability by comparing them with "exercise validity" criteria which are derived from consideration of the application and the types of simulation devices that are planned for the exercise. It is expected that exercise validity criteria will need to be developed empirically, and then tested and refined based on subsequent warfighter-in-the-loop experiments.

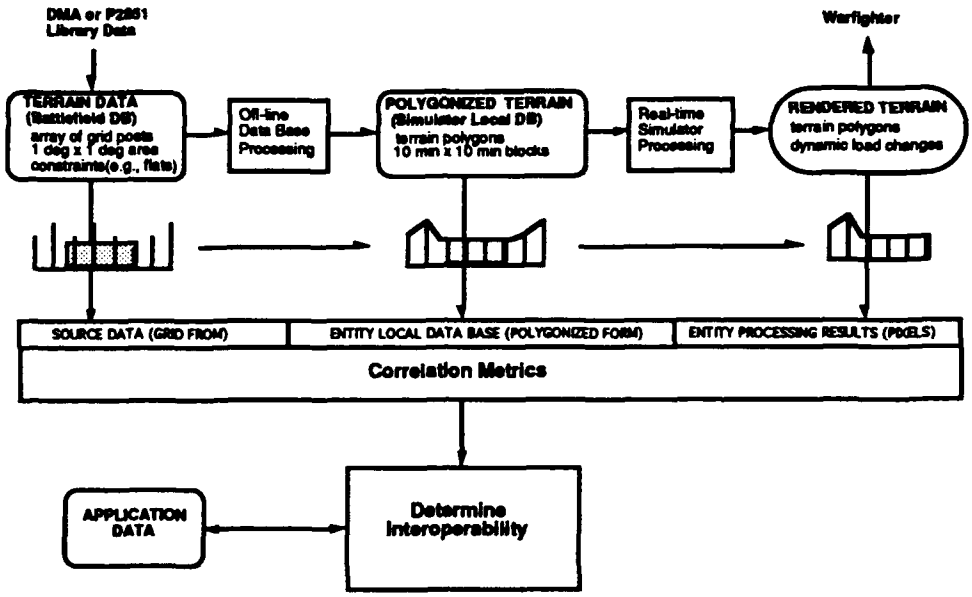


Figure 4-5: Correlation And Interoperability Framework

In this section we have been focusing on terrain/visual correlation. It is important to keep in mind that there are many dimensions to the correlation problem when viewed in the context of a Distributed Interactive Simulation environment. A strawman organization for a data base structure was presented in Volume I of this document, and is repeated here as table 4-2. This data base, consisting of SIMWORLD, BATTLEFIELD and SESSION components, is intended to characterize all of the data and entities that comprise the electronic battlefield. The task before us is to develop correlation metrics and ultimately interoperability criteria for all of the data types and entities listed for each of the intended applications - e.g., ground, air-to-ground, air-to-air, low-level flight, air defense, and so on.

DATA BASE ELEMENT	TYPE/CHARACTERISTIC
Cartographic Data	**Static Data**
Terrain	Gridded terrain data
Culture	Points, lineals, areals
Models	Geometry, attributes
Texture	Imagery
Platform Data	**Entities**
Vehicles	Geometry, appearance, dynamics, articulation, kinematics
Lifeforms	
Sites (relocatable)	
Munition Data	**Entities**
Guided	Geometry, appearance, dynamics, kinematics
Non-Guided	
Environment Data	**Entities**
Weather	Fog, lighting, TPH, wind
Atmospheric Effects	Smoke, dust, chaff, flares
Dynamic Terrain	Craters, berms, buildings
Electromagnetic Data	**Components**
Visuals	Rendering, load management
Electro-Optical	FLIR, NVG, LLLTV
Radar	Ground mapping, SAR, TFR
Electronic Warfare	Elint, jammers, C3I
Radio Nets	Digital Voice Communications
Session Data	**Control Data**
Network Initialization	Topology
Entity Initialization	Position, attitude, stores, etc.

Table 4-2: Strawman DIS Data Base Organization

5. Overload Management

5.1 Network Overload Management

5.1.1 DIS Systems Are Queueing Systems

DIS is a network. Like any network, it can be thought of as a system of queues. Both communications links (*e.g.*, T-1, 56Kbit) and communications processors (*e.g.*, routers, CIUs/CAUs, simulation nodes) can be modeled as queues. Each queue has a maximum number of elements (a length), and a service time for each element.

This Section examines what queues can be overloaded in a DIS system, what anomalies are manifest to the warfighters as a result, and what techniques can be used to minimize these anomalies.

5.1.2 What Queues Can be Overloaded?

Two types of queues can become overloaded. Communications link queues have limited bandwidth with which to transmit packets, and communications processor queues have limited compute power with which to route or otherwise process packets.

In order to better separate the effects of these overloads, we first examine the effects of limited bandwidth while ignoring any effects of limited compute power. Next, we examine the effects of limited compute power while ignoring limited bandwidth.

5.1.2.1 Communications Link Queues

DIS traffic is routed over an Internet of WANs and LANs. If the packet arrival rate at a given communications link queue exceeds its capacity for any length of time, the size of the queue will tend towards infinity. This means that packets may sit on the queue for different lengths of time, thus experiencing varying latencies. Further, since the queue is necessarily of finite size, packets will be dropped.

Such will happen, however temporarily, when the underlying network is not sized for the maximum burst rate of traffic. This may be likely if either dynamic multicasting or dynamic CIU-to-CIU routing is used, because then the traffic pattern over any given link in the network may be very hard to estimate *a-priori*.

Varying latency is manifest to the warfighter as discontinuous battlefield entity trajectories on the out-the-window display (See Section 3.) Smoothing may make the trajectory appear more continuous, but it may be computationally

expensive than, and will not ensure that the trajectory is physically realistic to the warfighter. (See Section 3.)

Dropped packets may cause even more discontinuous trajectories, because the "correcting" update does not arrive at all. Further, events such as direct fire impacts may be dropped, causing an anomaly to the shooter. For instance, a 105mm HEAT impact on an M2 should result in a catastrophic kill, and the shooter's tank will indeed paint the impact, but if the detonation packet is dropped, the M2 will never run its damage model and hence will not explode. This is sure to be disconcerting to the tank's crew.

5.1.2.2 Communications Processor Queues

The effects of communications processor queue overload are the same as those for communications links, but some of the causes are different.

The maximum per-packet arrival rate the communications processor can support is inversely proportional to its per-packet service time. This service time is a function of the processing required for each packet. Different processors perform different types of processing,

For example, network routing can be arbitrarily complicated, and its execution time may vary (*e.g.*, the number of multicast groups in use, the number of source/destination network pairs).[9] For simulation nodes, the DIS per-packet processing time may be a function of the number of entities in the exercise[10], and a function of how much link-layer hardware support is available. For CIUs/CAUs, the per-packet processing time may be a function of the content-based compression that is in use.

5.1.3 Techniques for Overload Management

Several techniques promise to help work around bottlenecks and eliminate their visual effects. However, each technique exacerbates another bottleneck or introduces more potential anomalies.

5.1.3.1 Conserve Bandwidth With Content-based Compression

CIUs (and CAUs) can minimize WAN bandwidth by compressing DIS packets based on their content. For instance, the DIS Entity State Update arguably constitutes the bulk of DIS traffic, and a significant fraction of update contents does not change from update to update (3 padding fields, markings, entity type, etc.). CIUs could conserve WAN bandwidth by defining a CIU-to-CIU protocol which contains multiple, highly compressed Entity State Update packets, and

9 Multicast Extensions to OSPF. J. Moy, Proteon, Inc. November, 1991.

10 The entity-identifier lookup time may be a function of the number of entities.

which transmits only changes. Such a scheme trades off network bandwidth at the cost of increased computation at the CIUs, and some additional latency. This cost increases linearly with the number of entities.

CIUs could arguably conserve WAN and LAN bandwidths by performing DIS Protocol-level routing, forwarding only those packets that "need to be" forwarded. This probably necessitates a CIU-to-CIU protocol in order to perform the time-varying routing.

Again, this scheme trades off bandwidth for computation which increases linearly with the number of entities.

5.1.3.2 Conserve CPU Power With Geographical Multicast Addressing

Geographically-based multicast groups have been proposed as a technique to minimize the CPU power needed to filter packets received at simulation nodes. In such a scheme, network layer multicast identifiers (MCIDs) would be assigned to regions of the terrain (*e.g.*, grid squares). Individual simulations would then subscribe to the MCIDs of the region that they occupy, and also surrounding regions. This would assure that they received only "interesting" entity state updates.

The assumption underlying this scheme is that filtering on MCIDs is computationally cheaper than filtering on DIS contents. This may not be true. Secondly, the much larger number of MCIDs may add to the cost of routing and receiving packets, because the list of MCIDs may be much longer than without such a scheme.

5.1.3.3 Apply Parallel Processing

This is the brute-force approach to coping with increased computation requirements of the communications processors. Per-packet routing and processing is an inherently parallelizable.

CIUs/CAUs and local simulation nodes are candidates for this approach. Unfortunately, it results in increased recurring cost.

5.1.3.4 Prioritize Traffic

It is important to note that there really are no packets that are unnecessary. Entity State Updates, the most obviously expendable packets, are generated only when absolutely necessary to maintain space-time coherence (see Section 3).

However, in the event that traffic exceeds either bandwidth or CPU capacity, prioritized traffic allows a graceful degradation. Packets that are lower priority

can be dropped. Again, entity state updates, which constitute the bulk of traffic, can be sent at a lower priority than combat events.

A drawback is that if combat events are sent at a higher priority than entity states, and if the network in fact delivers higher priority traffic more quickly than lower priority traffic, then the events can become dislocated in time with respect to the state updates.

Again, any dropped packets can disrupt space-time coherence.

5.1.4 Architecture Support for Network Overload Management

The two-tier network architecture provides separation and localization of DIS Cells to enable content based compression and filtering in the CIUs. Geographical multicast addressing can be supported in the top tier network, alleviating CIU/CAU loading. Parallel processing can be implemented in the CIU/CAU or in individual nodes, as can prioritization schemes.

5.2 CPU

Distributed simulation host CPUs are typically sized to meet their own needs as far as vehicle simulation, plus, they maintain a certain computational budget for dead reckoning and processing remote entities. There are two main mechanisms for managing computational overload on the host CPU; filtration, and dynamic algorithm switching.

When a packet arrives at a simulation node, several levels of filtration can be applied to the packet to determine whether it should be rejected or not. The very last rejection tests are the most intelligent. These might correspond to rejecting entities which are out of range, or not of primary interest. The remaining packets are placed on target priority lists as described in section 5.2.2.1. The priority lists define a pecking order for the importance of remote entities to the host entity. The primary use of priority lists is to prioritize the entities sent to the CIG for rendering. Most CIGs have a limited number of moving models that they can paint each frame. Each entity individually decides which remote entities are most important for it to see. This process of filtration and prioritization results in less CPU power being spent on remote entities.

Though filtration reduces CPU computational load by eliminating uninteresting entities, it is still possible for the CPU to be overloaded by the remaining entities. Because entities are sorted into priority lists, a simulation host can selectively downgrade the processing requirements for individual entities based on their priority and other aspects of their state. Table 5.3.1 shows some of the downgrades that a CPU might make if its computational budget were exceeded.

Table 5.3.1
Dynamic Algorithm Switching
(In order of degradation)

Algorithm	When to Switch	Rationale
Rotational Smoothing (Computational load mostly occurs upon receipt of packet)	Turn off, beginning with low priority vehicles, up until rotations become visible.	Rotational smoothing can only be discerned very close up. Position jumps can be seen at a much greater distance.
Rotational Dead Reckoning (Heavy computational load each frame)	Turn off after rotational smoothing is eliminated, low priority vehicles first, up until noticeable range.	At great distances, rotation is still less perceptable than position jumps. Positional smoothing still takes priority.
Positional Smoothing (Computational load mostly occurs upon receipt of packet)	Switch from high to low fidelity algorithm, low priority vehicles first, up until noticeable range.	A cheap algorithm such as SIMNET's would eliminate noticeable position jumps at long ranges, but be less compute intensive.
Visual Discontinuities May Occur After This Point		
Positional Smoothing	Turn off, low priority vehicles first, up until noticeable range.	It's better to see the closest vehicles with the best d.r. and smoothing, and for the further vehicles to degrade.
Rotational Smoothing	Turn off for vehicles within noticeable range, low to high priority.	Position will still be continuous, but rotation may noticeably jink.
Rotational Dead Reckoning	Turn off for vehicles within noticeable range, low to high priority.	Position will still be continuous, but rotation will jink with each new packet.
Positional Smoothing	Turn off for vehicles within noticeable range, low to high priority.	Position will jump the threshold distance with each new packet.

The table indicates one possible ordering of algorithm degradation to provide a continuous reduction of computational load. For a discussion of the metrics of computational loading due to dead reckoning and smoothing see section 3.3.1. The driving requirement for degradation ordering is the desire for the simulation task to be minimally impacted by the degradation.

6. Seamless Simulation

Abstract

This section provides an overview of how the proposed DIS Architecture supports seamless integration of heterogeneous systems. Seamless Simulation can be defined as the linking of dissimilar heterogeneous functionality systems on a simulation network. Current DIS efforts are aimed at homogeneous functionality objects (platform simulations and their associated environments) implemented in a heterogeneous manner (by different manufacturers). This has lead to the current situation in which several hundred vehicles are networked at the battalion level of team tactical combat. The presence on the currently implemented electronic battlefield of Semi-Automated Forces (SAFOR) is the first indication of interacting heterogeneous functionality systems. In SAFOR, the human controller inputs his orders and receives his information as though he were operating at the company (or arguably battalion) level of aggregation, with the SAFOR system interacting with the rest of the DIS battlefield at the platform level. This can be viewed as a first step towards integrating unit level wargames or simulations with the platform level DIS. Seamless Simulation is the attempt to extend the DIS to incorporate a wide range of functionality systems while maintaining the operational objectives and implementation principles of DIS. The critical DIS principle is that to be eligible for seamless integration with DIS a system must have a human in the loop. The human in the loop principle can be interpreted as meaning that human decision making is required during the simulation, not just to set up the initial state of the system that has been linked to DIS prior to run time. Each of five classes of system are examined with respect to their linkage to DIS, the mechanisms for supporting their linkage, and the effects of seamless simulation on the DIS standards are discussed. Much of the introductory material to this report (§§7.1 and §§7.2) is contained in [Downes-Martin, 1991] previously prepared for the Institute for Defense Analyses.

6.1 Introduction

SIMNET was DARPA's distributed simulation program from 1983 to 1990, consisting of distributed combined arms tactical team trainer prototypes. It forms the basis of the current Distributed Interactive Simulation (DIS) technology. The DIS technology supports a network of hundreds of vehicles at the battalion tactical level (see Figure 7-1) [Downes-Martin and Saffi, 1987], distributed over the continental USA, consisting of a mix of fully manned simulators, analysis tools (see Figure 7-1) [Garvey and Monday, 1989; GTRI, 1990], and Semi-Automated Forces (see Figure 7-2) [Brooks et al 1989; Downes-Martin, 1989b]. Systems such as computer generated forces interact at the vehicle level via a network protocol (see Figure 7-2) [Downes-Martin, 1989b], and current efforts to produce a DIS network standard [IST, 1989, 1990, 1991] are aimed at homogeneous functionality objects (vehicles) which are implemented in a heterogeneous manner (by different manufacturers).

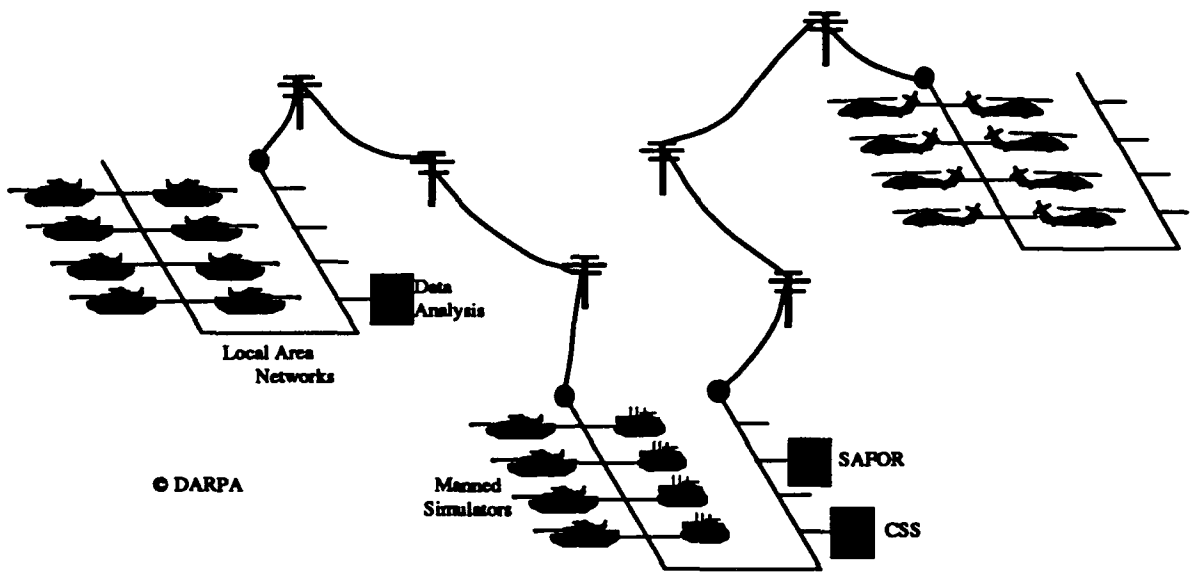


Figure 7-1: SIMNET Distributed Interactive Simulation Architecture. The baseline DIS architecture is provided by the DARPA SIMNET (SIMulator NETworking) project [Garvey and Monday, 1989; GTRI, 1990]. This links manned vehicle simulators, data analysis tools, Combat Service Support simulators, and Semi-Automated Forces by local and long-haul networks. All interaction is at the vehicle level via a set of vehicle level Protocol Data Units. Different vehicle types by the same manufacturer (Bolt Beranek and Newman) are networked, thus we have a homogeneous functionality (vehicles) homogeneous implementation (same manufacturer) system.

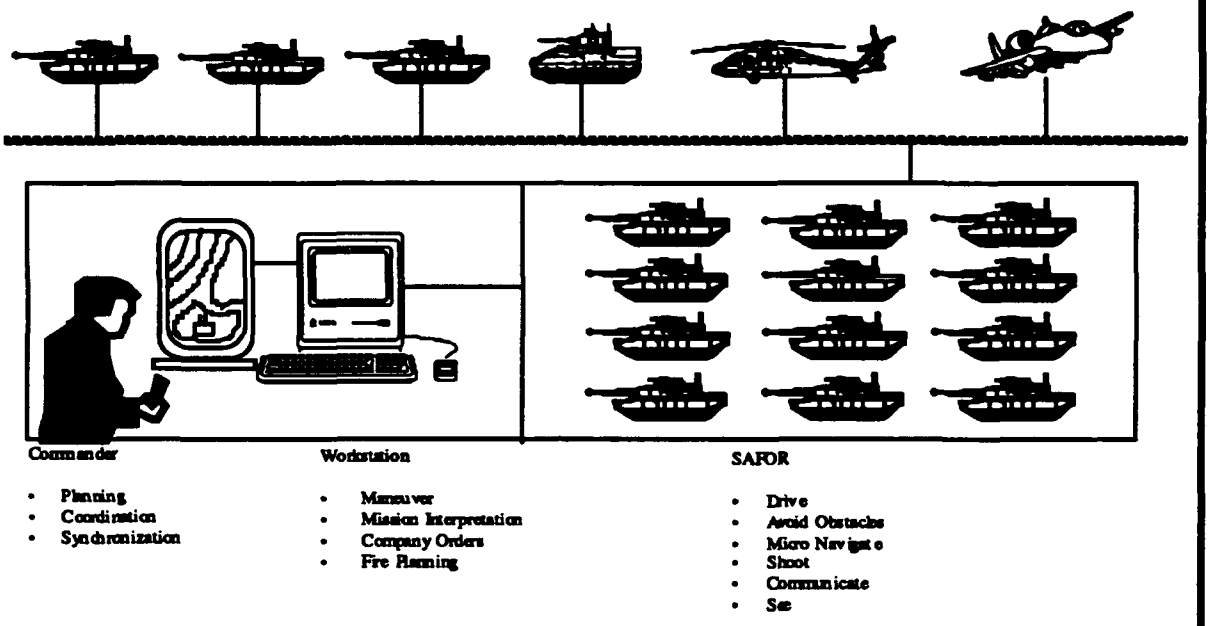


Figure 7-2: Semi-Automated Forces Architecture. Although the user-interface to the Semi-Automated Forces is at the Unit level, the SAFOR interact with SIMNET at the vehicle level via the same vehicle Protocol Data Units as do manned vehicle simulators [Downes-Martin, 1989b]. What we see here is unit level simulation at vehicle level resolution interacting with manned vehicle simulators, a first step towards Seamless Simulation.

Both DARPA and PMTRADE have recently announced major initiatives in distributed simulation applied to team training. These initiatives will be in addition to those already seen as extensions and expansions of the SIMNET distributed simulation technology [Pasha, 1991a, 1991b]. PMTRADE will extend the state of the art in DIS under their Battlefield Distributed Simulation -- Developmental (BDS-D) program, while DARPA continues its interest in the area under the umbrella initiative "Advanced Warfighting Simulation" (AWS). These two umbrella initiatives overlap each other considerably, although it is PMTRADE who is responsible for producing training products based on distributed simulation (such as CCTT).

For these reasons, DARPA and PMTRADE are interested, along with the services, in expanding the current DIS technology along a number of dimensions (see Figure 7-3) [McBride, 1990]. These dimensions include: vertical expansion from the current tactical battalion sized battles of hundreds of vehicles to Joint/Theater with tens of thousands of vehicles; horizontal expansion to include

all battlefield functional areas (such as C3I, IEW, Planning, et cetera); and application-oriented expansion in which the previous two expansions are implemented by networking vehicle simulators, wargames, computer-generated forces, and operational equipment. It is interesting to note that the DARPA conference on 73 Easting [DARPA, 1991b] last year has resulted in the requirement to integrate historical data tracks with interactive (and possibly stochastic) simulations.

This expansion requires the integration on a network of objects which are not only heterogeneous in implementation, but are also heterogeneous in military function. For example, aggregated unit level objects produced by a wargame must be able to interact with manned vehicle simulators (see Figure 7-4) [McBride, 1990]. All simulation objects produced by the heterogeneous functionality systems on the net must be able to interact with each other in a consistent and realistic manner. This requires that each object is presented with an environment that is supported by the computer system that generates that object. Seamless Simulation is not a goal in its own right, it is a proposed implementation for a set of interacting goals:

- **Vertical Expansion.** The current distributed simulation technology supports battles at the several hundred vehicle level, Regiment+ versus Battalion+, with a mix of manned simulators and semi-automated forces. The goal is to provide a simulated theater/joint command level battlefield at the vehicle level of resolution, in which tens of thousands of platforms are simulated by a mix of manned simulators and semi-automated forces. It is expected that the majority of these vehicles will be semi-automated. Vertical expansion assumes joint service participation, and international participation.

- **Horizontal Expansion.** Along with the vertical expansion, the full range of battlefield functional areas must be included. The current SIMNET technology is essentially direct fire and maneuver for ground and aviation, with crude artillery, logistics, and maintenance. When theater level warfare is considered, the full range of battlefield functional areas must be included. In fact, as the simulation expands vertically, functions other than fire and maneuver dominate, such as C3I, while at the battalion level the reverse is true

- **Five-Figure Vehicle Battlefield.** As the level of command rises, the potential number of platforms that must be simulated rises into the tens of thousands. Both DARPA and PMTRADE require that the simulated battlefield be observable at the vehicle level of resolution at arbitrary times and places. This must be done without prohibitive hardware or personnel costs.

- **Computer-Generated Forces.** On a simulated battlefield of tens of thousands of vehicles, most of those vehicles are going to be computer generated. In addition, the C2 distance between the senior level of

command and the vehicle level of combat execution increases beyond the normal "look down two, command down one". The CGF commander cannot effectively supervise his subordinates below two levels down without becoming overloaded, and thus the CGF must become autonomous and smart at the lower levels of command in order to be "indistinguishable from manned simulators".

- **Live Ranges and Operational Equipment.** As weaponry increases in complexity, lethality and range, it becomes increasingly harder to provide a complete battlefield environment on live ranges. Integration of live range platforms and distributed simulation is considered a possible solution to this problem.

The basic technical challenge here is to provide an interface between simulations of heterogeneous granularity such that each simulation or piece of equipment sees a rational world according to its expectations, and that the various worlds seen by all the participants are consistent with each other both in time and space. Aggregation and de-aggregation of simulation entities must be carried out dynamically, and the distributed simulation network protocol must be extended to take account of the new non-vehicle types of object on the net.

The Industry/DoD Standards for Interoperability of Defense Simulations Workshop, funded by DARPA and PMTRADE, is administrated by the Institute for Simulation and Training at the University of Central Florida. The goal is to develop standards that will allow vendors to internet their own vehicle simulators to those from any other vendor without knowledge of the internals of their competitors machines. The effort includes working groups to handle human-machine interfaces, network protocols, and terrain databases. This effort has so far restricted itself to interconnecting vehicles only, but will shortly be considering interconnecting wargames with vehicles. A number of draft standards and position papers have been published [IST, 1989, 1990, 1991; Pope 1989].

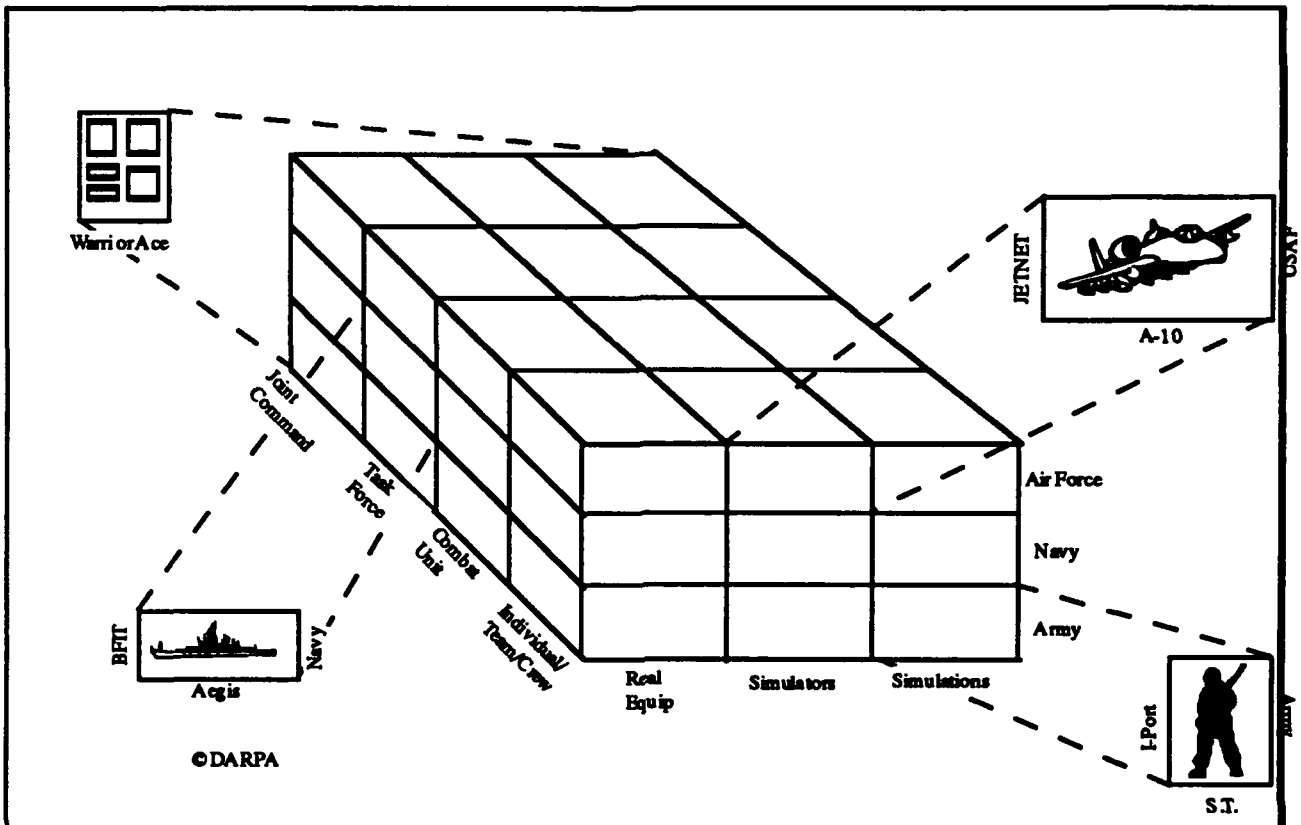


Figure 7-3: DARPA/PMTRADE Seamless Simulation. DARPA and PMTRADE wish to extend the Distributed Interactive Simulation (as exemplified by SIMNET) along a number of interacting dimensions [McBride, 1990]. These include vertical expansion to a full Joint/Theater sized simulated battlefield and the corresponding horizontal expansion to include all battlefield functional areas (such as electronic warfare and intelligence for example). To support this expansion DARPA and PMTRADE propose that extant and future systems of all functionality be linked into a single global synthetic environment. These systems include manned simulators, computer-driven Semi-Automated Forces, unit level wargames, and real operational equipment.

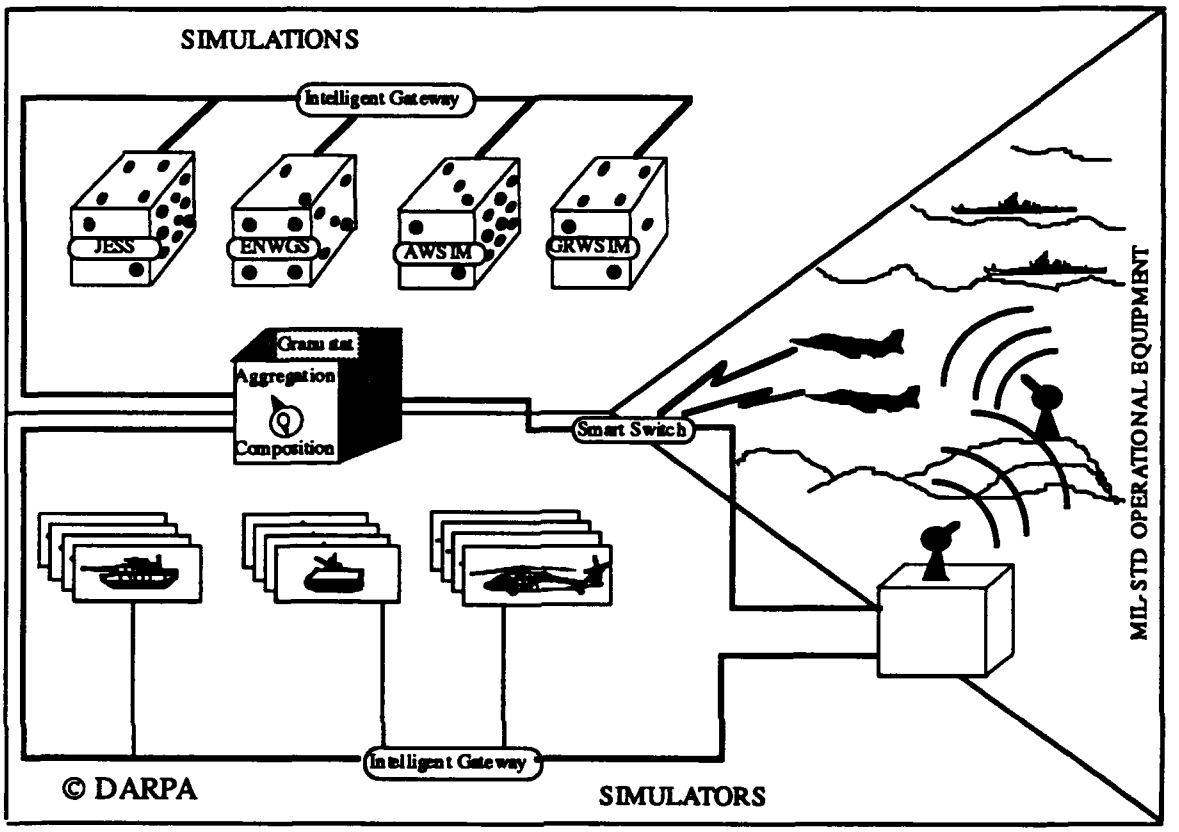


Figure 7-4: DARPA Heterogeneous Functionality Integration. Seamless Simulation implies the integration of heterogeneous functionality systems, systems that represent different kinds of object or application [McBride, 1990]. Peculiar problems arise when attempting to integrate high update rate deterministic manned vehicle simulators with low update rate stochastic wargames, or extremely high resolution operational equipment with medium to low fidelity simulation systems.

6.2 Overview of Seamless Simulation Issues

Seamless Simulation involves the integration into a single synthetic environment of many distributed objects with heterogeneous functionality and heterogeneous implementation. Some of the issues are common to those involved in integrating vehicle level simulations, such as terrain data bases, communications architectures, and interfaces. However, the current DARPA/PMTRADE sponsored workshop on DoD/Industry Standards for the Interoperability of Defense Simulations is already handling these in great detail. This detail is firmly based on the assumption of homogeneous functionality

systems, i.e. interaction of vehicles at the vehicle level of resolution. The overview of Seamless Simulation issues provided here is an attempt to review from literature a general set of issues that cover heterogeneous functionality systems. These issues should subsume those for the vehicle level. It should be noted however, that the issues discussed here are drawn from a variety of sources, and do not necessarily comprise a systematic theory of Seamless Simulation. The overview provided here is for the purpose of providing a background for the literature review. An attempt is made in this report to place these issues into a broad framework, however a proper theory of Seamless Simulation will be attempted in a later report.

One goal of the DIS Architecture is to support Seamless Simulation, the linking of heterogeneous functionality heterogeneous implementation systems into a consistent synthetic environment. This goal breaks down into two subgoals, both of which must be considered:

- **Synthetic Environment Construction.** The DIS will make use of a large number of heterogeneous systems to create a synthetic environment. Selection must be made before run time of which applications, processes, databases, and user interfaces (and which versions) are going to be used. A large number of general architecture issues are relevant, including the construction of new applications before run time by integrating components from multiple applications across the network.

- **Synthetic Environment Dynamics.** The DIS during run time must maintain a global internal consistency and satisfy user criteria of military reality. Application interactivity at the conceptual and dynamic level are the fundamental issues applicable to this subgoal.

These subgoals in turn generate three classes of requirements:

- **Simulation Truth.** There exists a simulated ground truth, that is the same no matter which system is used to interface to the DIS. Each interlinked system may or may not deal with uncertainty, intelligence, or the fog of war by suitably filtering this ground truth in some way.

- **Conceptual Consistency.** The DIS must be globally consistent in conceptual terms. Each interlinked system must be able to interact with the DIS using its own conceptual structures. For example, a vehicle simulation interacting with a unit level simulation must be able to interact with the units in terms of vehicles. Conversely, the unit simulation must be able to interact with the vehicle simulation in terms of units. This means that computer driven units or vehicles must interact with humans in a credibly realistic fashion.

- **Temporal Consistency.** The DIS must be globally consistent in temporal terms [Weatherley et al, 1991]. Each interlinked system must interact

with the DIS in a timely and causal fashion, maintaining temporal logic. For example, a fast update-rate vehicle simulation interacting with a slow update-rate unit simulation must perceive the DIS as containing fast update-rate vehicles, and those (unit generated) vehicles must respond to the vehicle simulation at the same update rate.

These requirements in turn generate four major technical dimensions to the problem of generating a DIS architecture that supports Seamless Simulation:

- **System Architecture.** System architecture determines the mechanics of how the distributed system is put together, including the communications between each interlinked system. Hardware and software issues are dealt with here. [Weatherley et al, 1991].

- **Data Management.** "Data Management is responsible for the rules by which data can be changed, and the interpretation of data values." Conceptual consistency issues are dealt with here, including aggregation and deaggregation of units, and terrain consistency. [Weatherley et al, 1991].

- **Human Machine Interaction.** Human Machine Interaction covers issues such as how simulation agents are to be built that simulate human performance to an acceptable degree of realism, the knowledge bases required to support the simulation of human behavior, and the cognitive models. [Downes-Martin, 1989a], [Deutsch, 1989], [Abrett et al, 1990a, 1990b].

- **Time Management.** Time management deals with the temporal consistency issues, timeliness and temporal logic, or causality, of the Seamless Simulation. This includes the different time rates at which say a unit simulation runs compared with a vehicle simulation. [Weatherley et al, 1991].

These dimensions in turn break down into component issues, and a certain amount of flexibility occurs in determining to which dimension each component issue belongs. It should be noted that the System Architecture and Data Management dimensions are similar to those raised by the Object management Group [Soley, 1990] in their architecture for integrating heterogeneous business applications across networks. Time Management arises in Seamless Simulation due to the competitive nature of combat and the resultant requirement for simultaneity and temporal logic.

Seamless Simulation generalizes interconnected vehicle simulations to interconnected defense simulations. The goal is to provide a simulated battlefield at theater level in which aggregate wargames (both current and future), vehicle simulators, actual field equipment (both combat vehicles and C3I assets), individual soldier "virtual reality" ports, and computer generated forces all interact in such a way that the seams between the technologies are hidden from

the participants. This includes interconnecting real platforms on live ranges with manned platform simulators, and integrating the individual soldier on the simulated battlefield as though he were on foot and operating battlefield equipment. The types of system that are eligible for seamless integration fall into five broad categories:

- **Manned platform simulators.** These interact directly with other platforms with the humans in the loop making all decisions. These systems provide user interfaces to platform level simulations.

- **Computer Generated Forces (CGF).** These are commander interfaces to simulations of C3I hierarchies, with code that drives platform level simulations. All interactions are evaluated at the platform level. Any aggregation is strictly at the level of simulating the aggregation that takes place in the real world intelligence process (the perception of the world versus ground truth). The human in the loop is making unit level planning and execution decisions, and the system is interpreting these in order to drive many unmanned platform level simulations. The system must now be able to carry out decision making and decision interpretation (decomposition of mission into subordinate missions) as well as handle platform level simulation. The CGF is a unit level simulation at the platform level of resolution and interaction.

- **Unit level simulations and wargames.** These are commander interfaces to simulations of C3I hierarchies, with code that drives unit level simulations. These systems do not continuously maintain platform levels of resolution at all times. Interaction between battlefield entities is at some unit level of aggregation above platform. Those systems that dynamically provide selective units at platform levels of interaction are carrying out deaggregation (and aggregation if they later recombine the platforms into units). In many cases, because of the aggregated level of interaction, these systems use stochastic processes to replace the causal relationships that have been hidden by the aggregation. Furthermore, they often run at faster than real time. Thus these systems differ from DIS in two critical areas, they run at different speeds and at different levels of granularity.

- **Operational platforms.** Four requirements are generated by the integration of operational platforms to DIS. First, the physical system must be represented by software within the DIS battlefield so that it can be perceived by the other participants, and if appropriate its battlefield vulnerability be simulated. Second, the actual movement of the operational platform in the real world (particularly if we are dealing with an air asset) must be correlated to its movement in the DIS world. The major point here is the correlation of real world terrain to the DIS terrain database. Third, the informational interaction of the operational platform (especially if it plays a major C3I role) must be simulated. The

DIS world must have the assets and be able to respond in ways expected by the operational platform. Finally and most critical, the operational platform must be able to have a DIS interface package of hardware and software placed on board without danger to the crew or platform, and without major impact on the cost of the platform.

- **Live ranges.** For the purposes of seamless simulation, live ranges are considered to be already instrumented with a Range Central Command and Control (RCC) system. Integration with DIS means the linking of the Range Central Command and Control system with DIS. In addition to the problems of integrating operational platforms to DIS, the characteristics of the RCC must be dealt with. The more critical characteristics are those that differ significantly from DIS, ie low bandwidth on the RCC net and low broadcast rate and reliability of transmission by the range platforms.

The DIS architecture must provide support for seamless integration of these classes of system into the DIS network. This implies a linking technology that handles system to DIS rather than system to system pairwise connection. The classes of problem that have to be dealt with are:

- **Temporal Consistency.** The Seamless Simulation must be globally consistent in temporal terms. Each interlinked system must interact within the DIS in a timely and causal fashion, maintaining temporal logic. For example, a fast update-rate vehicle simulation interacting with a slow update-rate unit simulation must perceive the Seamless Simulation as containing fast update-rate vehicles, and those (unit generated) vehicles must respond to the vehicle simulation at the same update rate.

- **Spatial Consistency.** The Seamless Simulation must maintain spatial consistency in the face of different spatial resolutions. Unit level simulations may not only operate on different time aggregations, but also with different spatial aggregations.

- **Functional Consistency.** Simulation objects representing the same real world objects may be represented in different ways and with different functions and behaviors in different systems. When these systems are linked, these differences must be dealt with. Polymorphism is one way of handling this if and only if the differences are acceptable in operational terms for the purposes of the simulation.

- **Organizational Aggregation.** Different systems may aggregate the battlefield objects into different levels of interaction, for example the platform level of execution of DIS and unit levels of execution of higher order models. Seamless integration must provide all systems with what appears to be a battlefield at the level of interaction expected by that system.

The proposed DIS Architecture must support the linking of each of the five classes of system to DIS in a seamless fashion. The architecture must indicate where the solutions to the linking problems should be embedded, and provide a mechanism for open development of the linking technologies.

6.3 Manned Platform Simulators

The seamless simulation issues of manned simulators are dealt with elsewhere in this document under Time and Space Coherency.

6.4 Computer Generated Forces

Computer Generate Forces generate platform levels of execution, and are by design DIS compliant. They are dealt with in detail elsewhere in this document. To summarize however, the existence of the CGF will require two changes to the DIS message protocol, both dealing with simulated communications. First simulated tactical communications by CGF means that the radio message packet must be able to carry data structures representing CGF generated radio messages. Second, if the CGF software command posts are distributed, then simulated face to face communications between simulated staff functions must be carried by the inter-cell tier of the network.

6.5 Unit Level Simulations and Wargames

6.5.1 Basic Interoperability Issues

Some of the major operational objectives in integrating unit level simulations (higher order models) with DIS are:

- provide a C3I interface to large units on the DIS battlefield, inspectable at the platform level of resolution.
- make use of extant and accepted wargaming algorithms.
- examine and assess wargaming algorithms within a platform level of resolution environment.
- expand the scale of the DIS battlefield with less than linear rise in hardware and personnel costs.

The unique technical problem that has to be solved for integrating unit level simulations to DIS deals with the level of interaction. Time and space coherency are dealt with elsewhere in this document. DIS currently assumes a common level of interaction at the platform level of resolution. Any extant system which is to be integrated on DIS must have its internals preserved as much as possible, and thus be presented with information which is consistent with its internal data structures and internal assumptions. Otherwise, it would be more rational to

simply rewrite the unit level simulation. The DIS architecture supports the integration of unit level simulations by placing them into non-standard DIS cells, and places to translation and filtering between DIS and the unit level simulation into the cell adapter unit (see Figure 7-5). Note that the Unit Levels Simulation's own user interface should probably be extended with data from the intercell database, since the DIS system will contain objects unknown to the Unit Level Simulation but which the user will want to interact with. The Cell Adapter Unit is responsible for handling these additional objects.

In this architecture, the distributed Cell Adapter Units across the DIS system will contain knowledge of all objects interacting on the DIS. The intercell databases will eventually be the repository of most knowledge about the internals of the non-standard DIS cells, and eventually it will become easier to modify and extend the Cell Adapter Units than the original contents of the non-standard cells. A common set of tools and techniques must be developed for the CAUs if seamless integration of extant unit level models is to be practical.

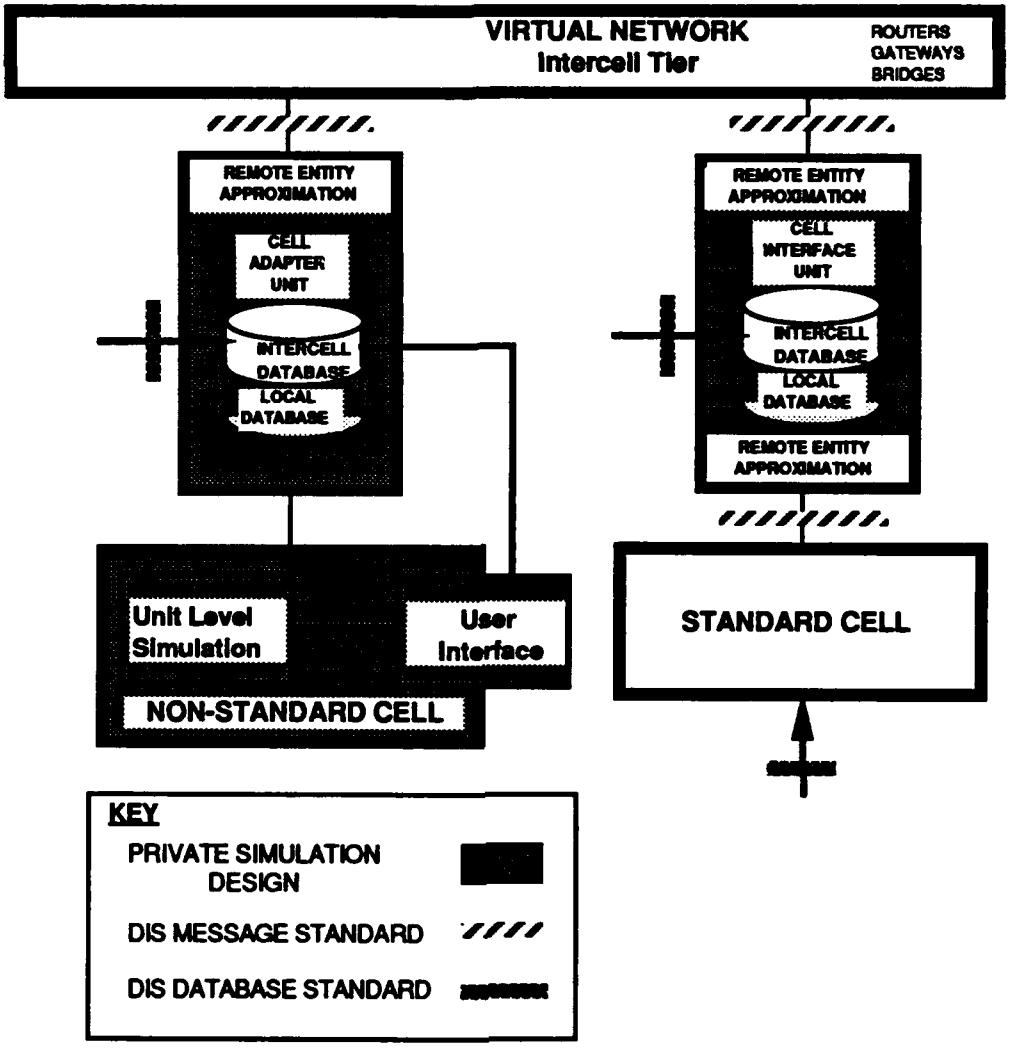


Figure 7-5 A Unit Level Simulation in DIS is placed into a non-Standard DIS cell. The Cell Adapter Unit is responsible for handling translation between the DIS and Unit Level Simulation.

6.5.2 Unit /Platform Translation

Interaction between units and platforms will be handled by the Cell Adapter Unit and must abide by two conditions. First the CAU must translate the DIS environment into one of units for the Unit Level Simulation and translate the Unit Level Simulation environment into one of platforms for the DIS. Second, the

interaction between the objects must follow the DIS paradigm of each object maintaining its own state and broadcasting changes to the net.

The CAU associated with the unit level simulation is responsible for deaggregating the units into platforms and maintaining the platform level of simulation in its own processors, using its own databases (see Figure 7-6). This is because the own-CAU is the only place that has the necessary information required for deaggregation into platforms. It therefore requires considerable knowledge of the internals of the unit level simulation to be placed in the intercell and local databases. It is arbitrary whether or not the CAU continuously maintains all platforms, or only generates platforms which are interacting with other objects on the DIS network (see Figure 7-7). A trade-off is required between the processing power required to deaggregate versus maintain all platforms. This latter is often called variable granularity simulation, and has been proposed as a technique for generating the appearance of massive numbers of platforms at lower processing costs.

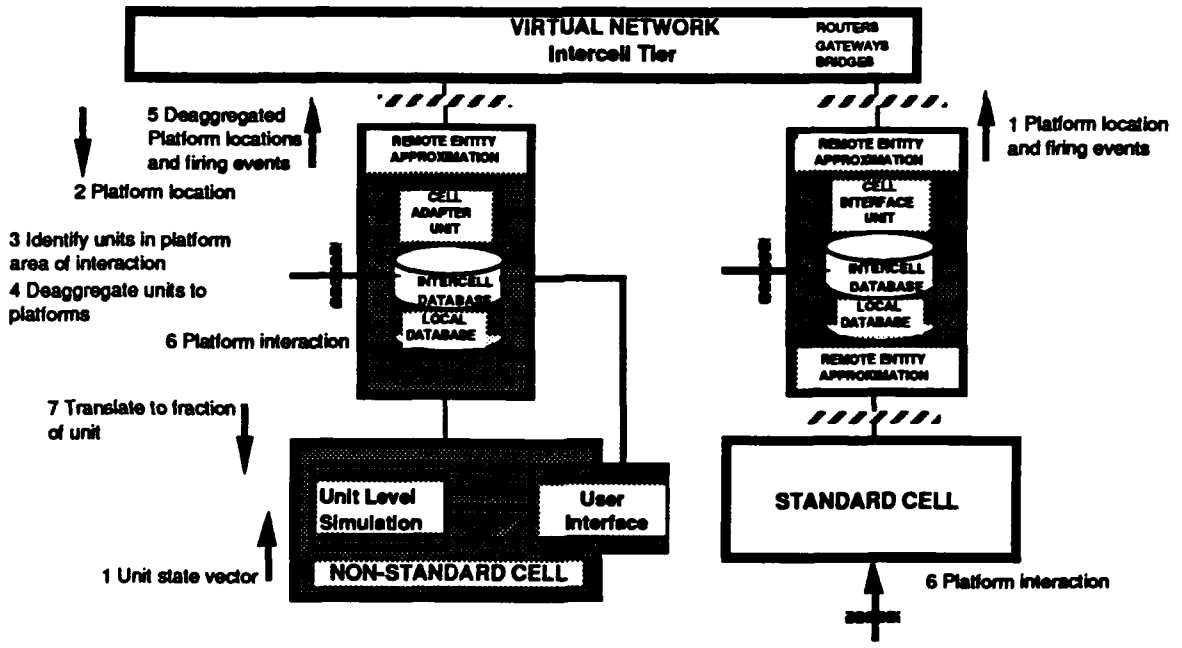


Figure 7-6 Deaggregation of Units occurs in the CAU since this is the only place where the information required to deaggregate in a rational manner exists.

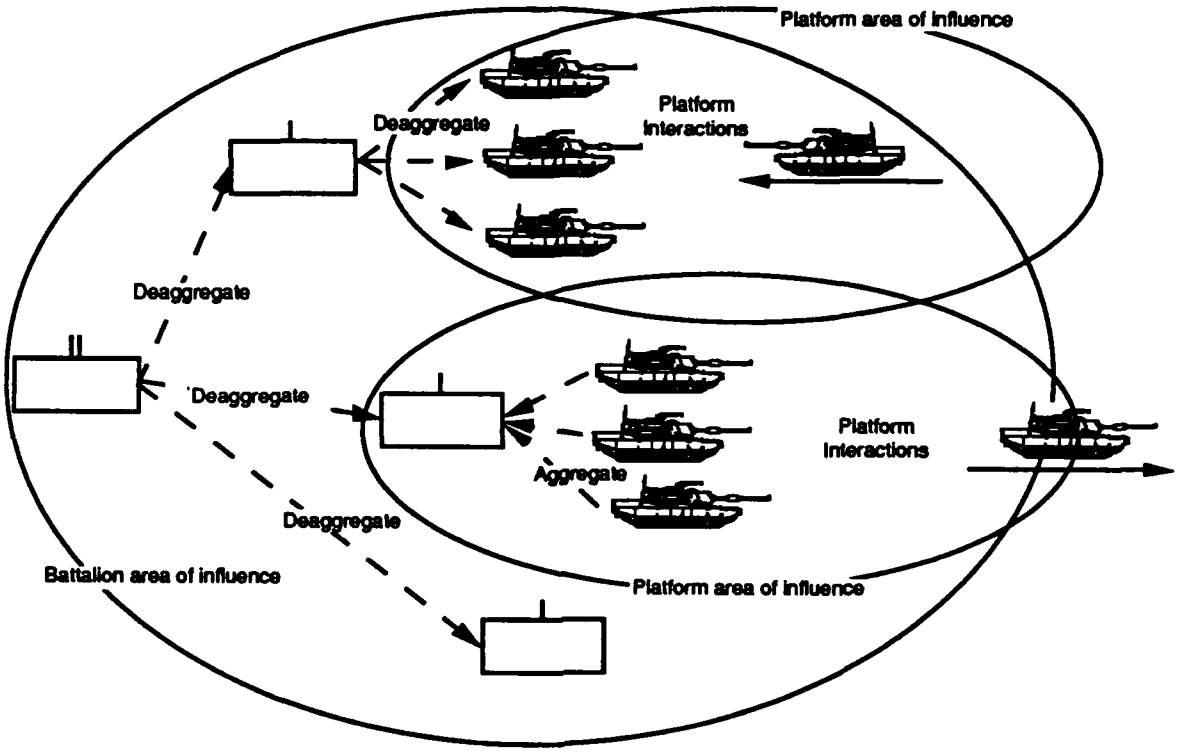


Figure 7-7 Variable Resolution Simulation occurs when the CAU only maintains platform representations of the units in the face of interaction with platforms from other cells.

Note that interaction at some intermediate level of unit aggregation (ie as might be assumed if two unit level simulations at different levels of aggregation were interacting with no platform level simulations involved) is currently assumed not to take place in DIS, although the architecture does not forbid this. The DIS Architecture will support unit levels of interaction (e.g. a simulation at battalion aggregation interacting with one at company aggregation via platoons) if the DIS message standard is extended to handle control messages informing the network what levels of aggregation are expected by the other CAUs since only the own-CAU may deaggregate the unit (see Figure 7-8).

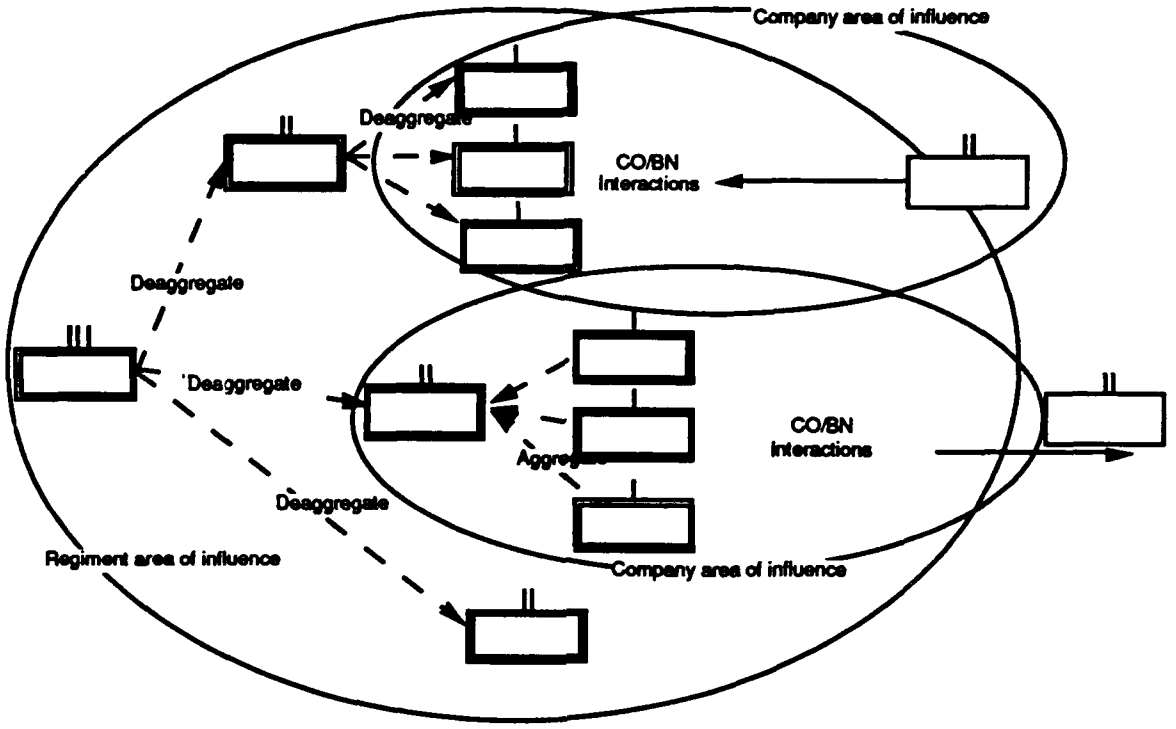


Figure 7-8 Unit Level Interaction is supported (but not mandated) by the DIS Architecture if the DIS message standard is extended to include control messages informing remote CAUs what level of interaction is required.

6.5.3 DeAggregation

One mechanism for handling deaggregation into platforms within the CAU is to embed in the CAU computer generated forces software. The role of the CGF is to provide platform level execution of unit level missions input by human operators. If the C3I structure of the unit level simulation can be used to feed CGF code, then the CGF can be used as a CAU (see Figure 7-9).

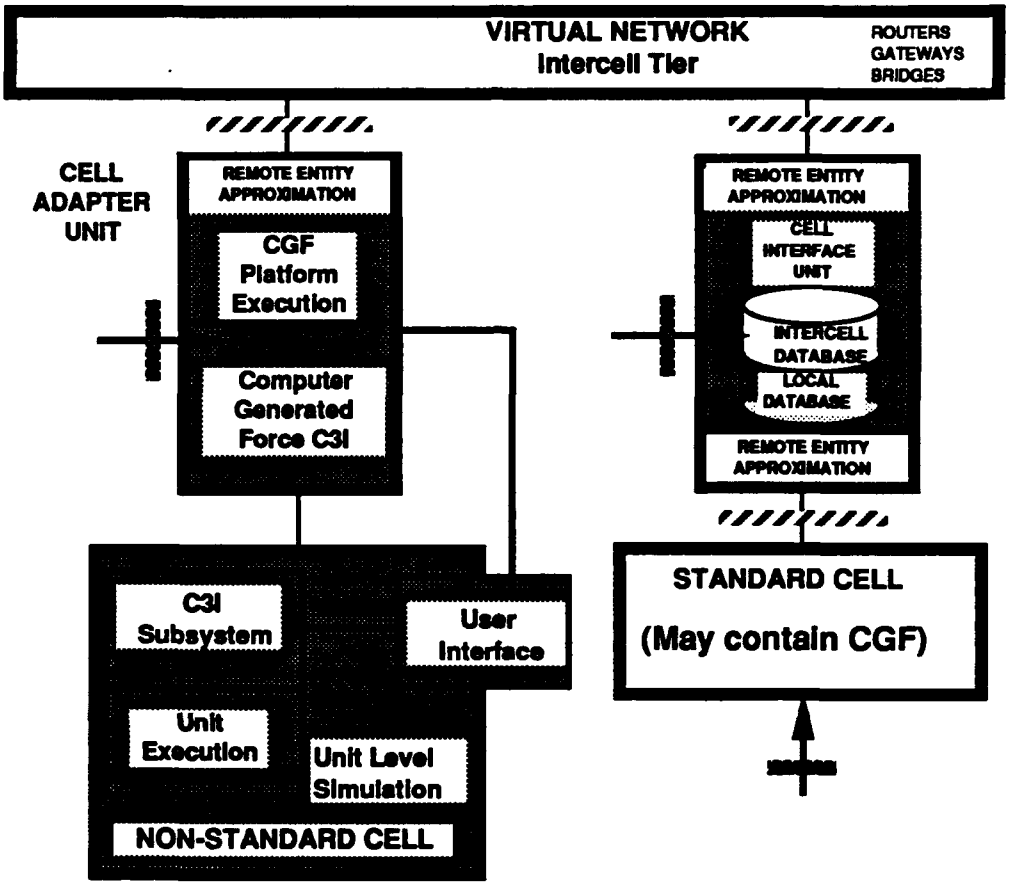


Figure 7-9 Computer Generated Forces is a CAU if it is used to generate and control platform simulations based on the command and control of a unit level simulation or wargame.

6.5.4 Impact on the DIS Message Protocol

Integrating unit level simulations will require an extension to the DIS message standard to take account of control messages between CAUs. These messages are those that deal with informing the network of the required level of deaggregation (for unit levels of interaction). If platform levels of interaction are the only levels permitted, then these messages are not necessary.

6.6 Live Ranges and Operational Platforms

6.6.1 Basic Interoperability Issues

This section discusses some of the critical issues involved in the interoperability of live ranges with simulation devices from the Distributed Interactive Simulation (DIS) community. Example scenarios are used to focus the discussion. Seamless simulation with DIS may provide a solution to four operational objectives of live ranges:

- Use simulation assets to provide range participants with the infrastructure of a larger battlefield than that possible using field equipment at the range.
- Use simulation assets to provide range participants with nuclear strike calculations and effects (similarly for chemical or any area weapon).
- Use simulation assets to provide range participants with indirect fire calculations and effects.
- Use simulation assets to provide range participants direct interaction with specialized platforms not available at the range.

In order to provide seamless simulation of live ranges with DIS, six assumptions are made concerning the operation of the range:

- Range platforms are networked in the range system to a Range Control Center (RCC).
- Range platforms transmit location and velocity information whenever sudden changes in those parameters occur.
- The platforms transmit own platform firing events whenever they occur.
- Latency from man machine interface to man machine interface within the range or to man machine interface within the DIS is within 300ms.
- The RCC handles hit calculations and informs the platforms accordingly.
- An accurate terrain database of the range is available to the simulation assets.

These assumptions mean that the DIS simulation assets can use DIS Dead Reckoning Algorithms (DRA) to model the range platform movement smoothly within their own image generation systems. An accurate terrain database of the

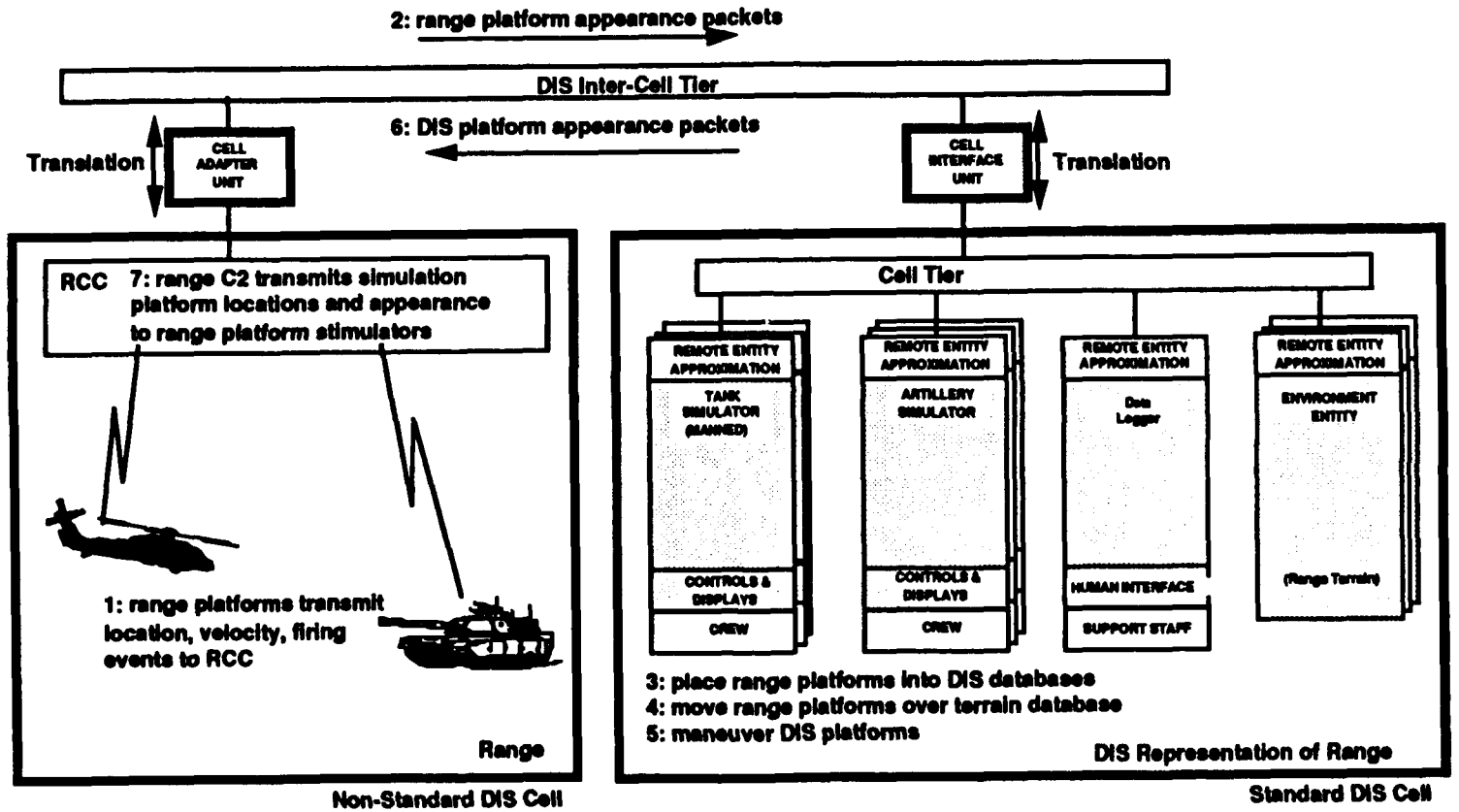
range is necessary to provide an adequate degree of correlation between the simulation entity and its representation in the range RCC database.

The DIS standard itself is based on a number of assumptions. Any networked system that proposes to use this standard must examine the DIS assumptions to determine if they are valid for that system.

- DIS is not bandwidth constrained.
- DIS objects are constrained in their I/O.
- The DIS network is extremely reliable, message loss is virtually zero.
- DIS platforms use a dead reckoning model of themselves to determine when to transmit an appearance packet. Transmission rates vary from one every five seconds for a static uneventful platform to a maximum of 15Hz for moving platforms (current SIMNET, may increase to 60Hz for fast movers). Transmission is reliable.
- Latency from one platform user interface to another platform user interface is less than 300ms

If a networked system is bandwidth constrained, or if the objects are unable to transmit appearance data at a rate that permits credible dead reckoning, then another message standard is advised. Since bandwidth between RCCs and range platforms is constrained, and the message data rates low and unreliable, the DIS protocol standard is not optimum and probably inappropriate for platform/RCC communications within ranges..

Range platforms transmit their location, velocity and firing events to the RCC (see Figure 7-10). The RCC transmits range platform appearance packets to the DIS Intercell Tier. The DIS asset reads these packets and creates shadow simulation platforms corresponding to the range platforms, and places them on the simulation terrain database. The DIS asset uses DIS dead reckoning algorithms to move the shadow platforms on the terrain database, and uses new updates from the range to correct locations. The DIS asset handles its own platforms, and transmits their appearance packets over the Intercell Tier. The RCC is responsible for transmitting to the range platforms the locations and appearances of the simulation platforms, for inserting these images into the sensors of the range platforms, and for informing the range platforms of damage caused to them by the simulation platforms. The Cell Adapter Unit (CAU) is responsible for translation between the range internal data and database representations and the DIS standard data protocol and databases.


2: range platform appearance packets
DIS Inter-Cell Tier
6: DIS platform appearance packets
Translation
CELL ADAPTER UNIT
CELL INTERFACE UNIT
Translation
RCC 7: range C2 transmits simulation platform locations and appearance to range platform stimulators
1: range platforms transmit location, velocity, firing events to RCC
Range
Non-Standard DIS Cell
Cell Tier
REMOTE ENTITY APPROXIMATION
TANK SIMULATOR (MANNED)
CONTROLS & DISPLAYS
CREW
REMOTE ENTITY APPROXIMATION
ARTILLERY SIMULATOR
CONTROLS & DISPLAYS
CREW
REMOTE ENTITY APPROXIMATION
Data Logger
HUMAN INTERFACE
SUPPORT STAFF
REMOTE ENTITY APPROXIMATION
ENVIRONMENT ENTITY
(Range Terrain)
3: place range platforms into DIS databases
4: move range platforms over terrain database
5: maneuver DIS platforms
DIS Representation of Range
Standard DIS Cell

6.6.2 Large Battlefield Infrastructure

The DIS simulation can support the RCC by embedding the range units in a larger battlefield consisting of simulated man in the loop units. For example (see Figure 7-11), the range unit requests support in the form of assets from higher command not physically available at range. The RCC play the command and control (since they are in charge of training), and decide which assets will be made available and when. These assets are simulated in the DIS system, and must fight their way across the larger simulated battlefield in order to support the range unit. They are attrited en route (by computer generated forces or by other manned simulator platforms in DIS), and arrive with realistic and validatable lateness and reduced force.

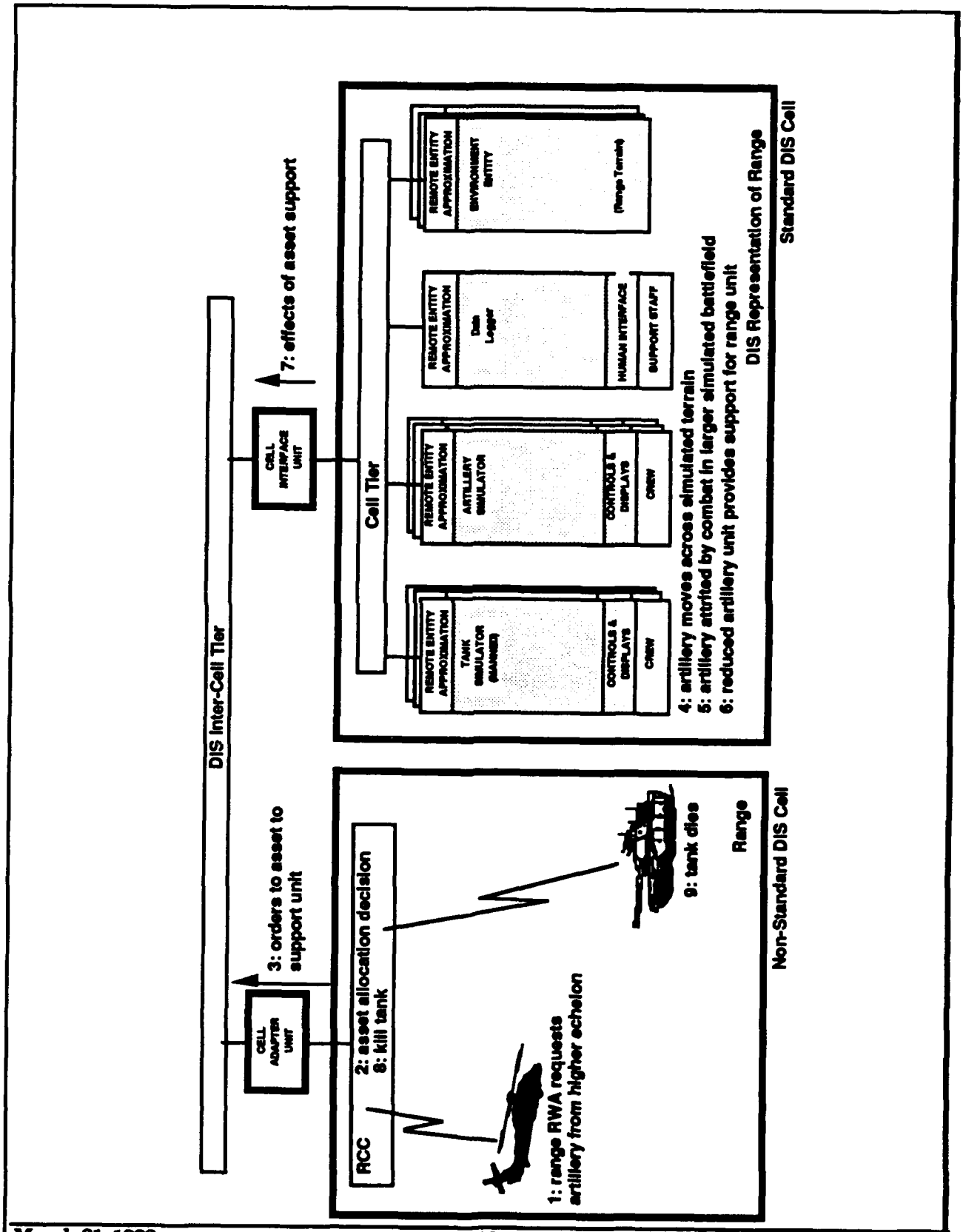


Figure 7-11 Embed the Range Unit in a Larger Simulated Battlefield.

6.6.3 Area Weapon Interaction

An example of area weapon interaction is shown in Figure 7-12. A decision is made on the range to request a nuclear strike, and this decision passed back to the RCC to simulate the processing of that request. If a strike decision is made, then the parameters of the strike are transmitted over the DIS Intercell Tier to the DIS asset. The DIS environment entity computes the effects of the blast on all platforms, and the effects of radiation (modified by atmosphere and terrain) on all platforms over time. These effects on range platforms are transmitted over the DIS Intercell Tier to the range as they occur (since the effects will vary over time as the platforms move in and out of the moving radiation cloud for example). The RCC system transmits kill or partial damage information to the range platforms.

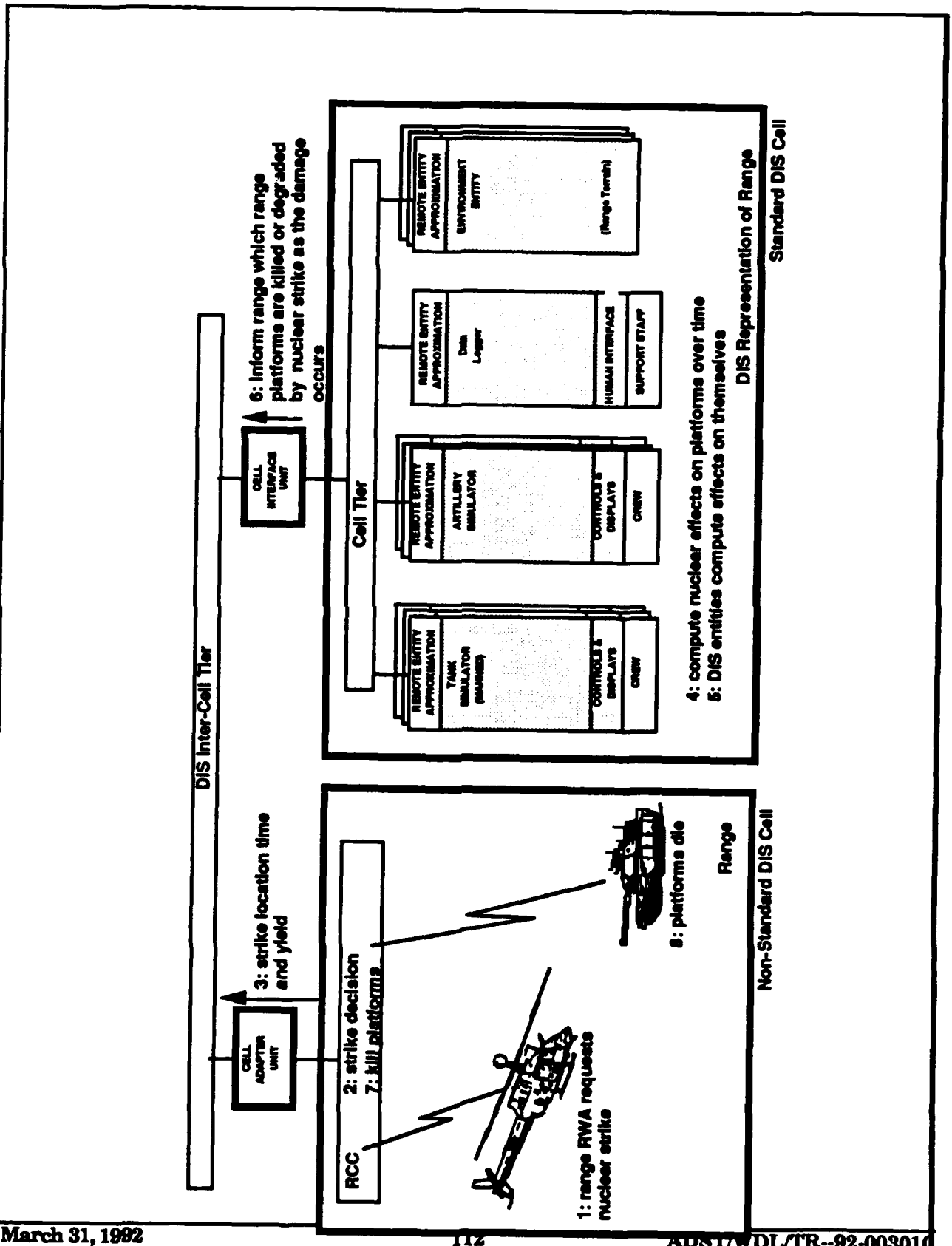


Figure 7-12 Area Weapon Interaction.

6.6.4 Indirect Fire Effects

Indirect fire (artillery) is handled similarly to area weapon effects, with the difference that indirect fire effects are immediate and localized.

6.6.5 Specialized Platforms

If the DIS system is to provide the appearance on the range battlefield of platform or event types not available to the range, then the RCC and the DIS databases must agree on the specifications and the physical effects of the simulated systems.

6.6.6 Major Technical Challenges

Two major technical challenges are of interest in dealing with interoperability of ranges and DIS:

- Superimposing images in the range sensor systems. The DIS system platforms must have their appearances and behavior superimposed on the fields of view of the relevant range sensor systems. Direct view visual interaction between actual equipment on the range and DIS objects is not feasible with available technology. While it is theoretically possible to superimpose computer generated images on the viewports of actual vehicles, such image generators with sufficient levels of realism are currently too complex, bulky, power-hungry, and expensive to operate on field equipment.

- Database correlation. In particular, it is critical that the terrain database available to the DIS system be closely correlated to the actual range terrain. This is required so that the images of the DIS platforms may be clamped to the real terrain surface (if appropriate). Otherwise platforms will appear to float above or burrow below the terrain surface in each others system.

Increasingly, modern war is fought at night, at long ranges, and using sensors. Under these circumstances terrain is not viewed or detected with the extreme detail available to the human eye at close ranges. Also, air platforms do not need to have their images clamped to the terrain surface and often engage at beyond visual ranges. If properly designed, it will be possible to inject images into sensor systems with credible realism. With these restrictions in mind, the following scenario indicates the kind of interactions possible using current technology.

6.6.7 An Example Scenario

This example suggests some of the possible operational interactions between DIS and range. It assumes that many of the technical challenges described above have been solved, or at least handled in such a way that the operational objectives of the range and DIS participants are met.

A Mechanized CO is operating on range at dusk. They call for a Aviation Scout mission to attempt to locate enemy defensive positions. Via DIS, a scout helicopter mission is defined and dispatched at the BDS-D site at Ft. Rucker. The mission flies over the simulated range coordinates (by this time it is dark and the helicopters use thermal and radar for navigation), and locates the enemy locations (which appear on the DIS database via DIS messages from the Range Control Center. The helicopters could only have been seen by the range enemy actors if the helicopter images are injected into the thermal viewers available to the range enemy actors.). The scout mission reports the position to the Mech Company, who calls for an artillery strike which is handled by a Fire Control Center located at Ft. Sill. The DIS artillery simulation determines the times and locations of the artillery rounds hitting the ground, and the damage to the enemy locations. The damage and time of damage is sent via the network to the Range Control Center which then "kills" the relevant enemy locations. The Mech CO then carries out a night attack on the damaged enemy locations.

While direct visual interaction between live range platforms and DIS system platforms is currently out of the question, a great deal of valuable interaction can occur via sensor viewing. This requires the injection and manipulation of images into the sensor systems, which is an advanced research topic for LORAL. Warfare is increasingly carried out using sensors, and this will make the problem of range and DIS interaction both easier and more valuable.

6.6.8 Operational Platforms

Operational platforms are a degenerate case of live ranges, in which the range contains only a single platform (the operational platform under consideration). The operational platform is considered to be a non-standard DIS cell. The Cell Adapter Unit contains the RCC, which may or may not be loaded on the platform (see Figure 7-13).

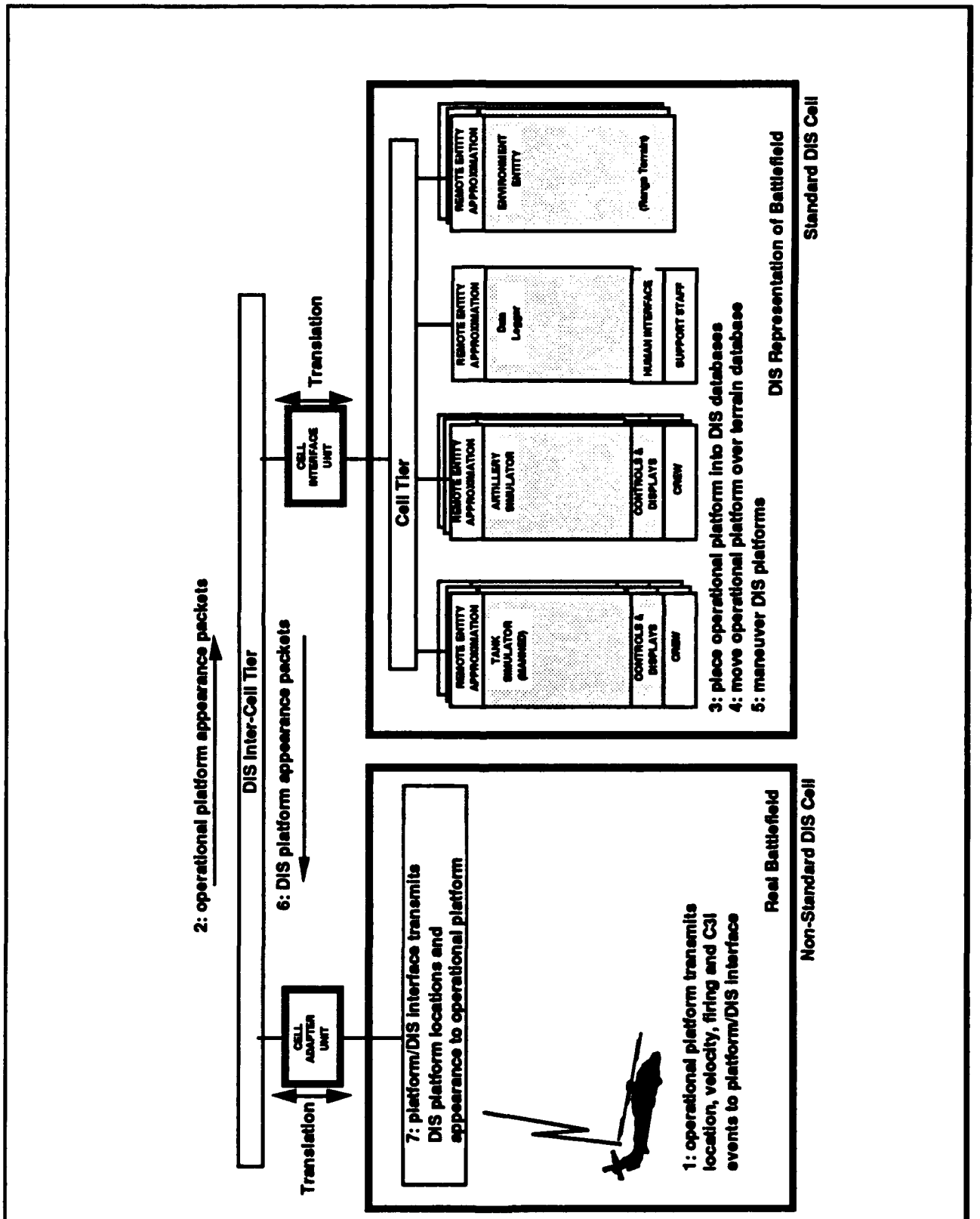


Figure 7-13 Operational Platforms, whether combat or C3I equipment, may be considered to be degenerate ranges containing a single platform with the Range Control Center becoming part of the Cell Adapter Unit.

6.6.9 Impact on the DIS Standards

Internal range message protocols must be developed that provide sufficient data, data rates and data reliability to interact with DIS. If the range system fails to track a platform for more than a few seconds then its appearance in the DIS system will be anomalous. Arbitrarily large jerks in movement may be observed depending on the reliability of the range tracking system.

Trade-offs between types of interaction, required data rates and reliabilities, and required range equipments must be identified and categorized.

The cell adapter unit for the range must be built to handle translation between the DIS PDU standard and the internal range protocol.

The range and DIS must share databases that describe the entities in both the range and DIS system, and their possible modes of interaction.

The range system must be able to track which simulated platform is being fired at by which range platform.

Insertion of simulated DIS events and platforms into the range platform sensors must be possible.

Real range terrain must be correlated with the DIS terrain databases of the range.

6.7 Conclusions

Seamless Simulation of arbitrary heterogeneous systems into DIS is a requirement if prior investments in system building , VV&A and analysis are to be exploited. The role of the DIS architecture is to support the seamless integration of these systems, providing architectural constructs for the location of seamless integration techniques and solutions. The Cell Adapter Unit, with its cell and local databases, and the encapsulation of heterogeneous systems into cells, provide a mechanism for integration. The paradigm of each object maintaining its own state and broadcasting changes can be maintained under this architecture. However, the DIS message standard will need to be extended to take account of control messages between CAUs coordinating the seamless interactions, to take account of simulated face to face and tactical communications between software representations of staff and commanders, and to take account of different families of network (such as those contained in ranges).

7. Special Problems

7.1 Dynamic Terrain

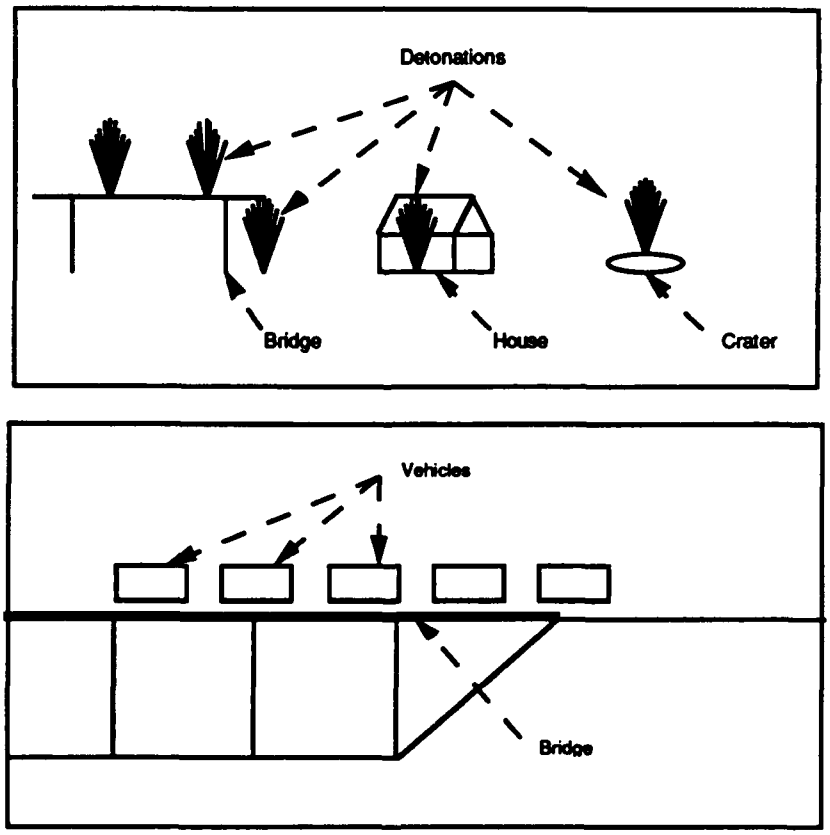
7.1.1 What is Dynamic Terrain?

In the short history of Distributed Interactive Simulation, terrain databases have been static. This precludes combat engineering operations and fire damage on database objects (e.g., craters, damaged buildings, and blown bridges).

"Dynamic Terrain" would remove those limitations, and provide a more realistic training environment.

7.1.2 Scope

Figures 7-1 and 7-2 depict examples we will use to highlight the technical problems associated with dynamic terrain. The first figure shows detonations, and some objects they can damage. The second figure shows a bridge that will collapse when enough weight is applied to it.



Figures 7-1,2: Dynamic Terrain Effects

In this section, we'll explain the need for dynamic terrain, the types of dynamic terrain, and the relationship of dynamic terrain to space-time coherence and interoperability.

The problem of dynamic terrain can be broken down into three subproblems.

1. How do we specify those portions of the terrain that can be affected by the evolution of the simulated battle? Further, how do we communicate this specification to all scenario nodes so that all Virtual Battlefield entities interact with the dynamic terrain in a globally-consistent manner?

2. How do we describe the changes that occur to dynamic terrain components during the course of the battlefield evolution?

3. How do we communicate the component changes to all Virtual Battlefield entities so that all entities share a coherent view of battlefield terrain?

An important benefit results from the object-oriented design of the DIS architecture; since terrain, with all its aspects (geographical, geometrical, cultural, emissivity) is an Actual Battlefield object, it can correspondingly be rendered in the architecture as a Virtual Battlefield object (entity). This is entirely consistent with the way warfighters view terrain--it possesses behaviors and state that must be reckoned with. Accounting for terrain in this fashion puts it on an equal footing with all other battlefield entities, and allows DIS entities to interact with terrain via the architectural communication paradigm (PDUs). This is a roundabout way of saying that the only obvious solution for coherent maintenance of dynamic terrain, the terrain server, is clearly and fully supported by the DIS architecture. The terrain server will inherit the characteristics of a battlefield entity.

7.1.2.1 Specification

The details of dynamic terrain specification are yet to be reached. Where those specifications will reside however, has been identified by the architecture. Those specifications, once developed, will reside in the Battlefield database, and as such, will be available for distribution to all participating simulation nodes, and the battlefield entities they represent. This will ensure terrain correlation and hence space coherence of the exercise.

Description

Change description involves two aspects. The first is identification and classification of the ways that a terrain object can change--i.e. enumeration of all the forms it can assume. For a cultural feature such as a building, it can be whole (undamaged), or can be damaged in a number of ways: blown-up, run-over, on fire, burned, etc. Following this enumeration of possible object states,

comes the specification of how those states appear, and how they differ from each other.

An interoperability problem that crops up at this point is that some CIGs may not be able to depict all possible enumerated forms, or the depictions may vary from CIG to CIG to such an extent that the differences affect the Virtual Battlefield and hence simulation outcome.

Communication

One must define PDUs which are analogous to the entity state packet and which can be used by the dynamic terrain entity to register is changes on the Virtual Battlefield. On the surface, the terrain entity will be very similar in operation to vehicle entities. Both types will receive PDUs, compute the effects which bear on themselves, and issue PDUs in response that relate their current status. However, a fundamental difference will exist between the two classes. The terrain entity will mark its evolution through time by discrete events while vehicle evolution (movement) will be marked by a continuously changing progression.

The problem here is missed packet updates. If entity A fails to receive an entity state PDU from vehicle entity B, the damage is somewhat contained in that entity A: has been aware of the existence of B; has been dead reckoning entity B's position and therefore knows B's position to within some (now possibly large) error tolerance; has the chance to receive the next entity state PDU from B when the next broadcast becomes necessary. This containment has been previously referred to as the "self-healing" nature of the protocol. With the terrain object however, no such self-healing exists. Dynamic terrain entity changes are discrete events triggered by dramatic battlefield occurrences: collapsed bridge, blown-up building, explosion-caused crater. If other entities fail to recognize these occurrences, then their perception of the Virtual Battlefield will dramatically differ from the perception of the other, recognizing entities.

One solution is to adopt a reliable transmission protocol for dynamic terrain messages. This solution, however, imposes added burden on the communication network and stands as a glaring exception to the message-transmission paradigm adopted for communication between other entities.

Another solution is to impose a minimum broadcast horizon time on the dynamic terrain entity. Such a mechanism already exists for vehicle entities. A time interval is selected which is appropriate for the vehicle. If the interval expires, and there has been no entity state PDUs issued by the vehicle during that time (either because no significant movement has occurred during the interval, or because the DR model position is still valid), then an updating broadcast is forced. This mechanism ensures that active entities maintain a presence on the network. However, some problems exist in implementing this solution. To begin with, an entity state PDU contains all the necessary information to correctly render the vehicle in both appearance and position. Only one broadcast is

required to repair any breach in time/space coherence. The dynamic terrain PDU must be so self-contained. However, the terrain entity may have a lot of information to report. Essentially the terrain PDU should describe the current changed state of all terrain components which have been dynamically affected on the Virtual Battlefield and which differ from their static state as described in the Battlefield database.

So this solution brings us to face another problem, inventing a concise descriptive notation for the dynamic terrain, and packing this notation into a PDU.

However, this solution finds support in the DIS architecture. Much work needs to be accomplished to implement this solution, but the details fit within the architectural scheme.

7.1.3 How Dynamic Terrain Relates to Space-Time Coherence and Interoperability

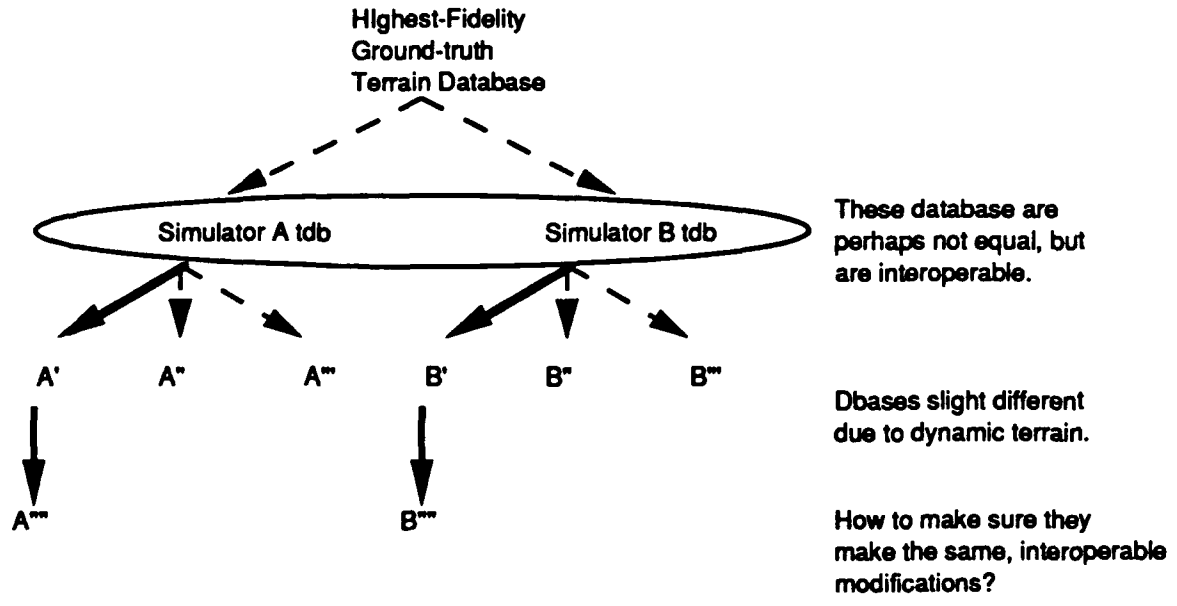


Figure 7-3: Heterogeneous Simulators

Dynamic terrain involves successive changes to a given set of databases occurring over the course of an exercise/session. Figure 7-3 shows two heterogeneous but interoperable simulators with different but interoperable terrain databases. Dynamic terrain is an incremental change to the databases. In order for interoperability to be maintained during the exercise/session, each simulator will have to make incremental changes which preserve interoperability (e.g., A goes to A', and B goes to B', with A' and B' being "equal" enough to be interoperable).

7.1.4 Technical Challenges

In this section, we'll examine some of the technical challenges presented by dynamic terrain, with emphasis on those of space-time coherence.

7.1.4.1 Balancing Antagonistic Real-time, Concurrency, and Serializability

We would like to guarantee that all observers see equivalent dynamic terrain changes. We would also like them to see them as soon as they happen. This section explains why these goals are antagonistic.

Real-time

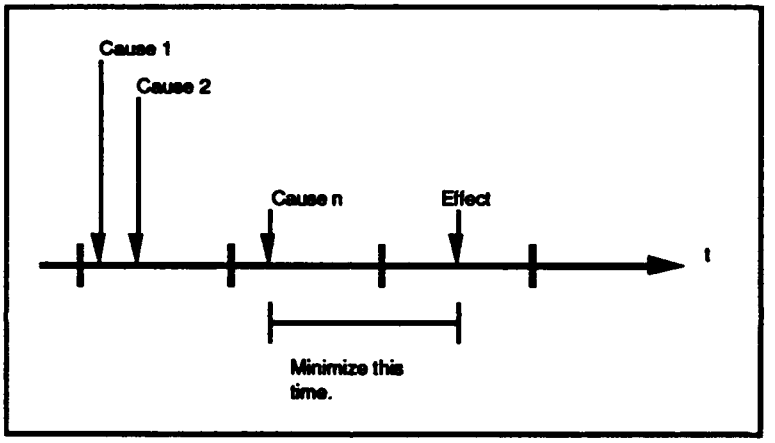


Figure 7-4: Dynamic Terrain Timeline

Figure 7-4 shows a timeline of causes which combine to produce a given effect on the terrain. We wish to minimize the time and distance between the causes and effects, to avoid introducing visual anomalies to the observer (e.g., a long time between detonations and crater formation). As we have seen in previous sections (*ref space-time coherence*), these latencies are due to internal processing and external network latencies.

Concurrency and Serializability

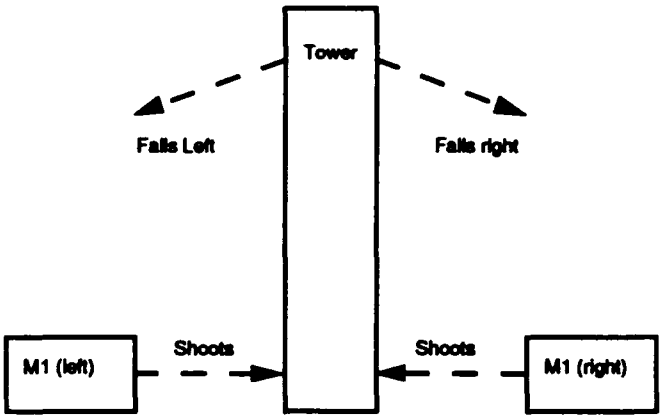


Figure 7-4: Tower Falls in the Direction of Shooter

In order to illustrate our point, we use the example of two M1s concurrently shooting at the base of a tower. We assume that the tower will fall to the side upon which it is shot.

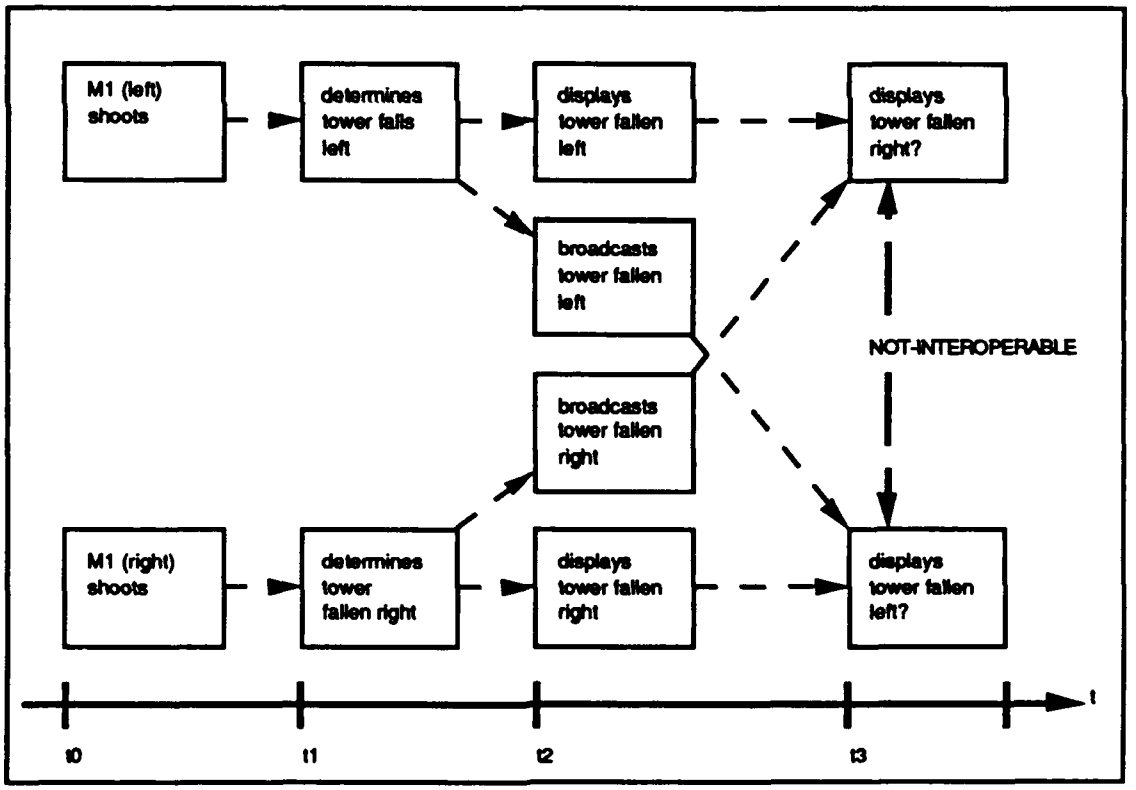


Figure 7-5: Tower Causality Diagram

Figure 7-5 shows the corresponding causality diagram. Here, in order to get smaller real-time latencies, concurrency is maximized. Each shooter determines the cause and the effect. Notice that:

- The effects calculations are serialized differently for each shooter, and the "correct" damage at the time is clear in only local frames of reference.
- Time-stamping is not sufficient to resolve this. For instance, even if the events occurred at different times (here they don't), a a simulation that "changed its mind" at t3 based upon timestamps would still have displayed "incorrect" information between t2 and t3.
- The final results for each are not well-defined. What is the correct display for all?
- Does it matter which way the tower falls, as long as it falls only once (does not flip back and forth), and that it is the same for all observers?

In conclusion:

- Real-time considerations drive us to maximize concurrency.
- Limiting anomalies and preserving interoperability drive us to mandate serializability.
- Unfortunately, concurrency and serializability are antagonistic goals.

7.1.4.2 Determining Cause and Effect in a Distributed Fashion

Single causes are depicted in Figure (detonations, above), which shows detonations causing craters to appear, bridges to be blown, and buildings to be damaged. Effects are easy to compute in a concurrent, distributed fashion from single causes, produced by single computers.

However, Figure (bridge, above) shows a more complex cause and effect relationship. The bridge has a weight limit which, when exceeded, will cause the bridge to be damaged. How can effects be computed from distributed causes, contributed be different computers? Shall each entity compute the effect of the sum of its causes, and *all other* entities' causes, on all terrain? *What if the effect on the bridge is cumulative over the course of an exercise/session? Shall the entities keep track of the history of causes for all terrain?*

7.1.4.3 Protocol Definition

In this section, we'll examine the technical challenges involved in defining a Dynamic Terrain network protocol.

Will it be Self-Healing?

A "self-healing" protocol is its own virtue. It has simplified initialization/rejoin/restart characteristics, and momentarily losses in connectivity are not calamitous to the exercise/session.

The self-healing property of entity state appearances is achieved by simply rebroadcasting the entire state of the world at least every 30 seconds. This is not so brute-force as it seems, because the entity states do change more often than 30 seconds, on average twice per second. Also note that reliable packet delivery is not a problem here, if entity states are rebroadcast every half-second on average.

However, this might not be an optimal self-healing strategy for dynamic terrain elements, which probably will not change twice per second. Furthermore, reliability is more critical for dynamic terrain so that each observer sees the same environment, free of individual anomalies.

What are the Semantics/Naming

SIMNET and DIS have a well-specified semantics for naming entities (the entity identifier, and the entity type schemes). However, a dynamic terrain protocol will need the semantics to be able to refer to individual elements of the terrain database (at least by location), and to dynamically change them. Such network protocol semantics do not yet exist.

Defining Interoperable Incremental Changes

Dynamic terrain must specify changes that achieve the desired effect, and that preserve interoperability equivalence (see Figure database tree above).

7.1.4.4 Software Engineering Many Cause-Effect Relationships

This section shows two polar opposite approaches to software engineering of dynamic terrain. This first attempts to fully distribute the simulation of the terrain, and the second centralizes the simulation.

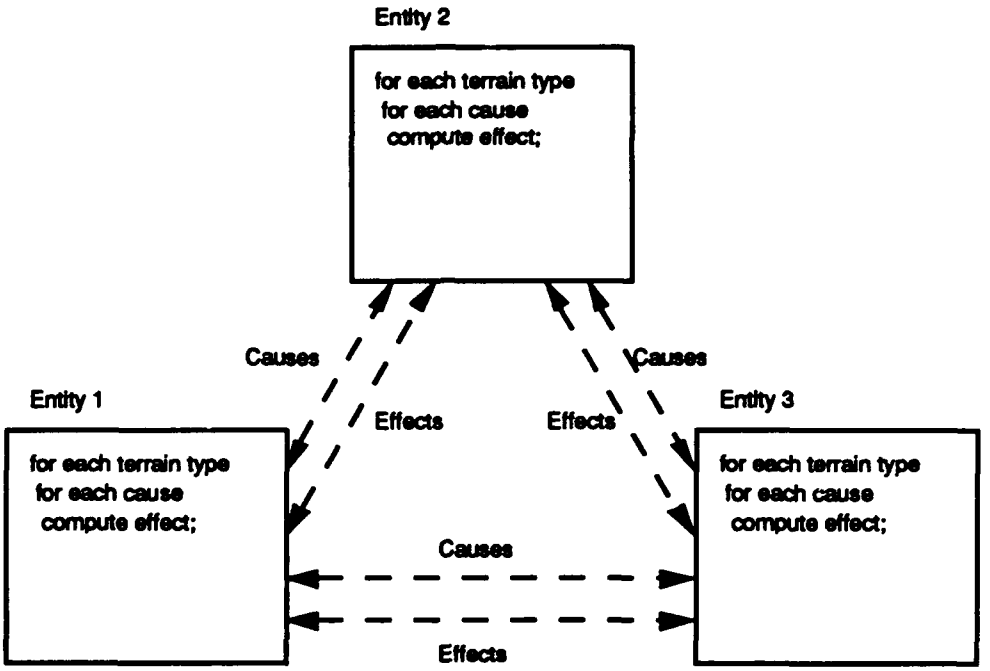


Figure 7-6: Distributed Approach to Dynamic Terrain

Figure 7-6 shows an attempt to fully distribute the simulation of the terrain. However, in "fully distributing" the simulation, no one host has responsibility for the simulation.

Consider the case of SIMNET/DIS direct fire. The shooter determines the hit (cause), and the victim determines the damage (effect). An approach like that above would have the **shooter** determining damage. This has several drawbacks:

- Each entity that causes something has to know the effect on each and every possible other entity. It is very hard to upgrade such a system. Our Figure shows that to add a new terrain type involves upgrading each and every entity.
- A minimal, extremely low-cost entity simulation would have to include code to interact with all terrain types just to play at all. If it didn't, then there would be anomalies.
- Such an approach allows several different and disagreeing models of terrain interactions. For instance, M1s from one simulator manufacturer could knock down houses, but others couldn't.
- It is hard to change the fidelity of the terrain model to suit the objectives of the exercise/session.

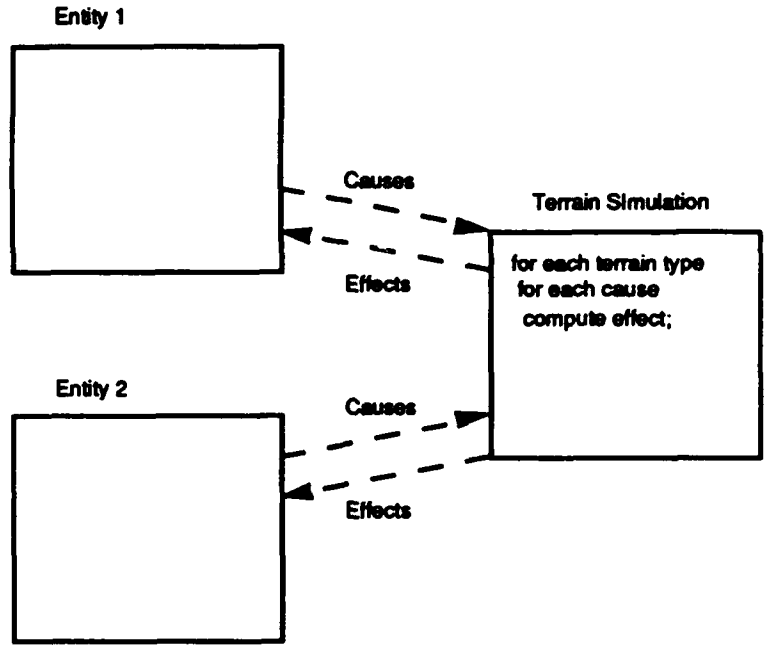


Figure 7-7: Dynamic Terrain Approach Analagous to Direct Fire

Figure (above) solves these problems in an analogous fashion to SIMNET/DIS direct fire. The entities determine the causes, and the terrain determines the effects.

7.1.5 Approach(es)

In this section, we'll compare and contrast three approaches

- A fully-distributed, CIG-centric model, where each entity computes causes and effects.
- A database server model, where the server responds to read/write requests.
- An Object-Oriented model, where causes are concurrently computed, but effects are serialized.

7.1.5.1 CIG-centric Model

This model is depicted in Figures (three-entity and causality diagrams). The effect on the terrain is implicit in the cause.

Advantages

- This approach is fully distributed.

- Arguably, it has the best real-time performance.

Disadvantages

- Lacks serializability of effects, leading to visual anomalies.
- Cannot upgrade terrain model without upgrading all entities.
- Cannot select different terrain fidelity models without selecting different versions of entity code.

7.1.5.2 Server Model

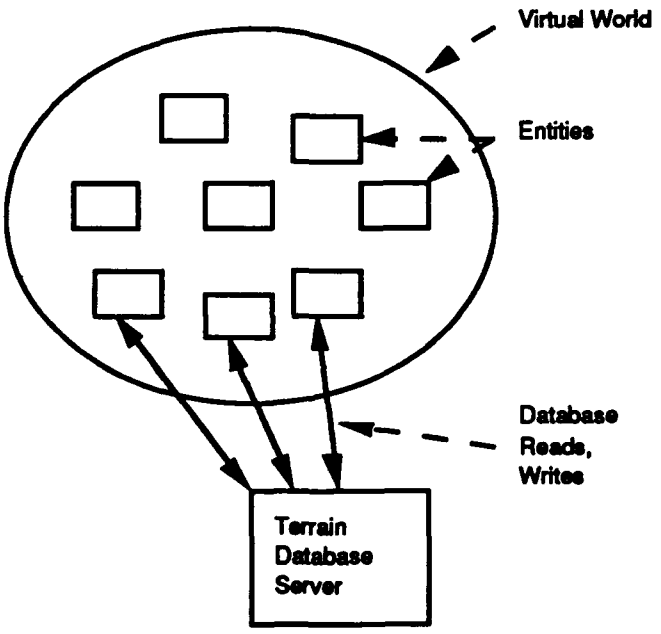


Figure 7-8: Server Model

This model is essentially treats the terrain database like a large, passive data structure. In this model, entities like NFS-treatment of tdb

7.1.5.3 Object-Oriented Model

A given computer hosts an M1 simulation entity. Is it a "server" for that entity? Well, yes and no. "Yes," because it alone runs the simulation of that M1 entity..

But "no," because the other entities in the simulation do not need this entity to participate in the simulation. It is for this reason that we claim the object-oriented model is not a server-model.

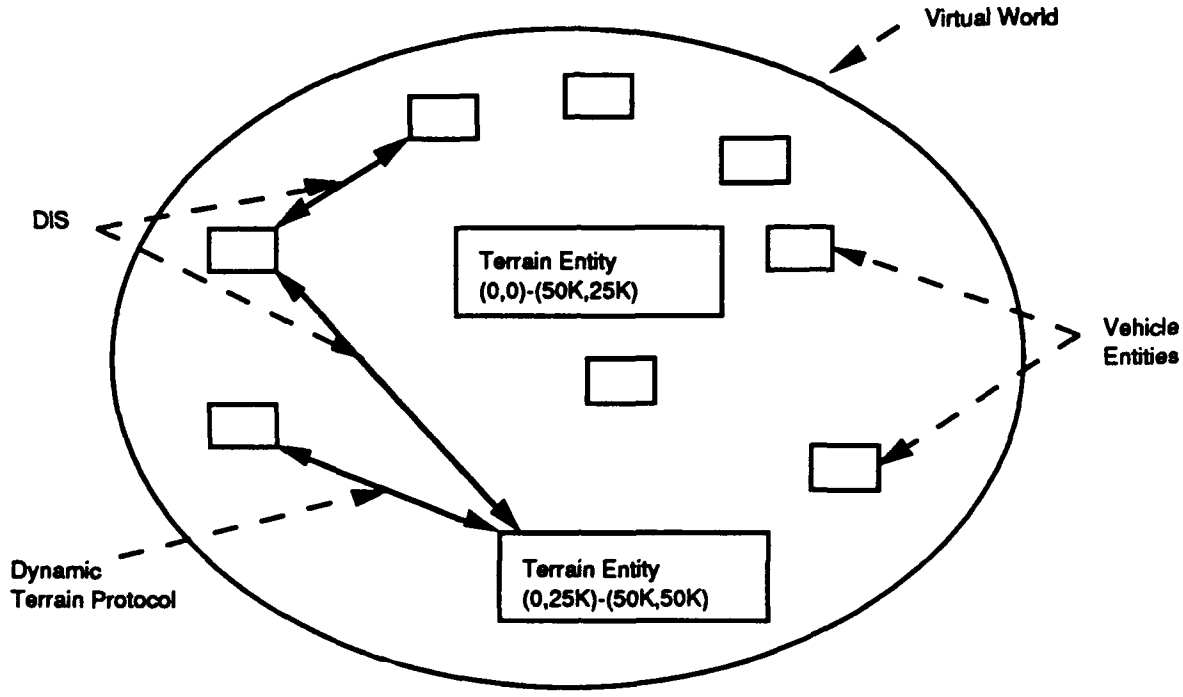


Figure 7-9: Object Oriented Model

In this object-oriented model, terrain:

- is a name-able entity.
- is "active." It responds to messages just like other entities do.
- determines the effects upon itself based on network inputs, just as other entities do.

7.2 Electronic Warfare

7.2.1 Overview

The outcome of modern battlefield engagements is becoming increasingly reliant on effective use of the electromagnetic spectrum. For this reason extensive use of ECM and ECCM techniques have become an integral part of any comprehensive battle plan. Consequently the EW (Electronic Warfare) portion of simulation efforts has also increased in scope and in priority. EW simulation efforts are often divided into two broad interrelated areas termed environment generation and environment perception.

The environment generation function relies on an underlying database which describes the characteristics and behavior of each possible emitter type. In large EW simulation efforts each type description is referred to as an emitter model and

all models become database records. Generally for small engagements the total number of model types is in the range of several hundred. Larger multi-force battle areas may exceed several thousand emitter types. The current trend is to allow the user access to the models through a SQL (Standard Query Language) to facilitate the addition, deletion, and update of models. Thus it is becoming common to hold these models in a relational database and access them through a commercial application software package such as ORACLE.

In a number of cases, emitters do not exist as independent systems and only exist as part of a larger system which may contain a number of emitters. Surface to Air Missile (SAM) systems, Navel ships, and most aircraft, fall in this category. For example a SAM site may contain an acquisition radar, target illuminator, command guidance beam, and several communications sources which act in concert as part of a single entity composed of numerous RF sources. Usually all emitters associated with a platform or site are combined and treated as part of a single overall model. Thus models can become quite complex and encompass a number of individual emitters.

A given model will have two groups of attributes. The first is parametric and describes all sets of measurable characteristics associated with the various possible emanations of each emitter. This includes both the parameters which can be measured on a per pulse basis (such as frequency and pulse width) and those which are derived from a set of pulse measurements (such as Pulse Repetition Interval). There are also a number of inter and intra pulse agility issues which may or may not have to be modeled, depending on the capabilities of the perception problem.

A parameter set is often termed an emitter mode and mode is sometimes synonymous with function. For example, for a fire control radar type, there may be a separate set of operational parameters (i.e. mode) for each of the radar functions such as search, acquisition, track, launch, etc. However, more likely a given mode will be legitimate for several functions and some functions will have the choice of many different modes. Attributes which belong to this parametric category include frequency, pulse width, pulse repetition interval, polarization, scan pattern, power level, and antenna patterns.

The second category of model attributes deals with its various modes as a function of time. This sequencing is governed by weaponeering, tactics, doctrine, operator proficiency, and C3 considerations. The cues for the changes are normally based on command, time, range, or response to another system's initiative. Often these considerations are lumped together under the heading of tactics.

It is apparent that model development can become a complex task. Therefore to minimize overall cost and promote both computational and storage efficiency the models are normally developed only to the level they can be perceived by the sensors of ownship. From the emitter modeling standpoint, sensor perception can be subdivided into three categories: 1) systems that output symbology only, 2)

systems that output portions of the instantaneous radiated environment and, 3) systems that output a combination of 1 and 2. In general a tactical RWR (Radar Warning Receiver) is an example of the first and second categories and Elint, ESM and SOF receivers are examples of the second and third categories.

As an example the ALR-56 Radar Warning System processes radar intercepts, performs signal identification, and displays symbology for the highest priority detected threats. Lethality level is implied in the type of symbol displayed and its location of the RWR display. A number of tones are synthetically generated and represent warning as opposed to analog samples of the radiated environment. Thus a simulation only needs to know which threats are present to generate the right tone.

A simple ALR-56 sensor simulation examines the skin power received at the aircraft for each active radar to determine if it can be detected. If detectable, an appropriate delay is added for the scan on scan problem and the symbol and/or tone associated with that radar type is displayed if appropriate. Here the database need only contain type, power and mode. None of this information needs to be transmitted at a high rate.

The biggest drawback to this level of simulation is that it does not account for the ambiguities inherent in the identification process. That is, several radar types will overlay significantly in parametric space. This leads to numerous misidentifications that the combatant has to deal with during an engagement. The sensor response in such situations is typically to assign the most likely or the most lethal identity to the signal, when in fact, any one of several radars types may have produced the intercept. Since not all theaters possess the same weapon systems to the same extent, the ambiguity problem is different for each theater of engagement.

The problem also changes as a function of time as new systems or new radar modes are fielded. This has given rise to extensive use of theater specific EID (Emitter Identification) tables by modern sensor systems. In fact, several sensor types employ more than one EID on a given mission. The most common occurrences are aircraft which use altitude dependent EID's. The problem is also slightly different from sensor type to sensor type, because they do not all measure the environment in the same manner or with the same precision.

A somewhat larger definition requirement is dictated by sensor systems which output portions of the demodulated electromagnetic spectrum directly. This is seen to a small extent in such systems and the ALR-69 which can output demodulated audio. It is seen to a larger extent with such systems as the APR-46A Panoramic Receiver which also incorporates a video display for directly viewing pulse trains. In order to be able to generate the proper video display for this type of receiver, the models must also contain information on instantaneous pulse behavior. This includes intentional and unintentional pulse to pulse and pulse group to pulse group parametric variations.

Examples of required characteristics in the frequency domain include hop, sliding, phase coding, and chirp. Necessary PRI modulations include jitter, stagger, sliding, and wobbulated. Similar pulse width modulation characteristics would also have to be included in the database definitions.

Note that the requirement for pulse waveform fidelity does not complicate the time coherency problem. Here the requirement is for event coherency and not for fine grain waveform coherency. That is, if a JARM transitions to a agile parameter mode, all panoramic receivers should note this change at the same time (within human response time). However, because no EW operator views the display of any other aircraft, each receiver presentation may concurrently be in a different portion of the same agile pattern.

RF Radar emitters are employed in the electronic combat environment for a multitude of purposes. A radar can be used solely for target detection (early warning, height finding, search), for target monitoring (acquisition and tracking), or for weapon delivery (missile guidance, fire control). It can also be used for airborne navigation purposes, such as ground mapping, altimetry, or terrain following/terrain avoidance (TF/TA).

The data base characteristics of the radars in the Battlefield Data Base may be divided into two classes - static (ID, platform, type, function, waveform type, scan characteristics, etc.) and dynamic (modulation mode, antenna scan mode, PRF mode, position in many cases, etc.). The implementation of DIS is made much more tractable by only updating the required dynamic characteristics while retaining the static parameters in the data base of each computer. The dynamic parameters can be much more limited and, in many cases, act as a pointer to the correct set of static characteristics in the data base. This greatly reduces the amount of data which must be transmitted between simulators.

In low fidelity simulations, only a very limited number of these characteristics are required. However, if the data base is to be compatible with advanced, high fidelity simulations, then the capability to handle a set of characteristics such as these would be required. Simulations which realistically consider the ECM threat environment and ownship emissions cannot ignore a more detailed definition of emitter characteristics.

Existing simulation standards and draft standards do not provide the above level of detail. Thus, existing work on high fidelity models should be expanded to the DIS radar cases to accommodate the next generation of DIS systems.

IR The category of infrared (IR) emitters represent sources of IR radiation that interact with various passive (one-way) systems in the electronic combat environment, such as Infrared Warning Receivers (IRWRs), IR seekers in missiles, imaging infrared (IIR) sensors, forward-looking infrared (FLIRs) sensors, etc. Steady state IR emitters include exhaust plumes from missiles, rockets or aircraft; internally heated objects such as tanks (hot engines), jet aircraft tail pipes, or factory smoke stacks; ground fires; and solar-heated objects.

with a single icon or mode number), unique message traffic to be simulated, tactics, equipment, location, and the link to the C3 net.

Correlation issues are expected to be modest, since delays of up to a second in the onset of communications are rarely of concern, and position correlation errors are smaller than existing df equipment capabilities. A more significant problem is that of correlation to the C3 net and the fact that highly classified data is involved concerning tactics, equipment, and alternate signal routes (such as switching to landlines, different frequencies or equipments, etc.). Likewise IFF is an important consideration.

C3I High fidelity simulators involve at least 4 levels of command, control, and communications. The required data base involves location, performance (timing, throughput rate, load capability, etc.), communication links, equipment, and command structure at each node. Also required are representative message traffic and algorithms for their use. In addition, the data base must include the descriptions of the links which exist in the C3 net or nets. This information includes method used (radio/landline/satellite/etc.), equipment used, descriptions of equipment (where applicable), alternate routes when reconfiguration occurs, and specific characteristics where these are selectable.

7.2.2 Electronic Warfare and tactical communications

The arena of electronic warfare (EW) and tactical communications poses special problems to DIS technology. Issues to be reckoned with include: the volume and pace of transactions in an EW environment, integration of analog voice communication with the digital Virtual Network, simulation of a plethora of tactical digital communication protocols, and the modeling of electro-magnetic (EM) propagation and attenuation with respect to terrain and atmospheric considerations. In this section, we will focus on those issues that can lead to a disruption of time/space coherence on the Virtual Battlefield. In this narrow realm, the problems that crop up are: non-correlation of hits and evasions, entity-to-entity differences in signal propagation, and the reliability and sensibility of tactical communications.

7.2.2.1 Electronic/Smart Weapons

7.2.2.1.1 DIS Weapons Paradigm

The DIS weapons paradigm, inherited from SIMNET, provides for a division of responsibilities between the shooter of a weapon and the victim. The shooter must:

- Determine the flyout path of the weapon.
- Determine if the weapon hits a victim.

- Identify the victim and compute the impact point relative to the geometry of the victim.

- Communicate the hit event to the victim for damage assessment and to all other simulation nodes for proper display of the weapon's effect.

The victim is responsible for assessing and reporting damage sustained.

This paradigm represents a sensible and neat division of computing labor and imposes the least potential disturbance to time/space coherence. Note that this paradigm is particularly well-suited to the simulation of ballistic weapons which are not visually discernable, nor for which there are useful evasion maneuvers.

7.2.2.1.2 Problem: High-Speed and Complex Transactions

Simulation of the EW environment introduces smart munitions. This environment and all that it entails (seeking, acquiring, tracking, jamming, counter-jamming, and etc.) is characterized by high-speed and complex transactions which are beyond the capabilities of DIS communications. This is particularly true when high-fidelity simulators are considered. Simulation sessions consisting of linked high-fidelity simulators, modeling full suites of EW equipment, are characterized by:

- High-volume EW interaction between entities, due to the presence of multi-spectral sensitive receivers

- Tightly-coupled transactions, due to the interdependence of seeker/sensor, electronic counter measures (ECM), electronic counter counter measures (ECCM), target/decoy maneuvers, environmental effects, and C^3.

In modeling EW systems one must consider the very significant amount of data describing signal structure and the spatial/temporal characteristics of the beam. Antenna patterns and signatures must extend over a sizeable portion of volumetric space. Many kinds of radiations exist, including ones from highly-complex emitters, such as jammers and advanced radars. Regardless of the update rate on information between linked simulators, there are always EW transactions which occur significantly faster than any update rate. At the same time there are long and complex sequences which can adequately be described by defining action and initiation time.

Studies performed at Loral have shown clearly that relying on PDUs to carry the higher volumes of data required to support simulation of the EW environment leads to bandwidths which are beyond the state of the art.

7.2.2.1.3 Problem: Maneuverability and Evasion

A guided, powered munition (such as X-rod, Maverick, Stinger, guided submunitions, or other advanced guided weapons) can rapidly maneuver to hit an evading target, such as a high-speed aircraft. As the guided missile closes in on the target, the final moments of this encounter, (known as the "end game") is characterized by extreme maneuvering actions and reactions between the two entities. The volume of entity state messages that would be required to describe this complex interaction could not be transmitted with the present state of DIS network communication capabilities.

7.2.2.1.4 Symptom: Non-correlated Evasion

The time/space disruption symptom that can potentially occur in this dynamic environment is that of a non-correlated evasion as seen by the shooter and the intended victim.

The shooter controls the dynamics and kinematics of the missile flyout, as well as the complex EW reactions of the missile to the victim. These model responses are assessed against the shooter-possessed dead-reckoned model of the victim's dynamical state and the stream of EW messages from the victim that happen to make it back to the shooter in time to be reckoned in the calculations. The victim, on the other hand, also must use late and insufficient data when computing its evasive responses, both dynamical and ECM.

The result of these problems is that the shooter may determine a hit while the victim may observe the missile flying harmlessly past--either because his ECM equipment signals that the missile was defeated, or because he successfully evades the flyout path of the dead-reckoned position of the missile *(i.e.* because of network latency, the missile entity state PDUs do not arrive in time to allow the victim to maneuver against the true missile position). Alternatively, the shooter may determine a miss, while the victim may observe the missile intersecting with its own flight path, and may in fact be provoked to register a collision PDU from the event.

7.2.2.2 Signal Propagation

Weather and terrain can aggravate the potential of time/space disruptions in an EW encounter, for they too must be considered when modeling EW operations. The flyout path of the guided weapon depends not only on the flurry of EW transactions, but also in the interaction of those transactions with the environment--effects such as attenuation, noise, ducting, and diffraction.

But the problem is more general, and it represents another way that EW simulation exposes vulnerability to disruption in time/space coherence. This potential exists between classes of simulators which have different models for signal propagation. Differences in the way that these models account for

attenuation, noise, ducting, and diffraction lead to differences in the perceived Virtual Battlefield. The problem is similar to that of simulators which model different terrain fidelities. In the terrain case, visibility (radar or optic) differs between two entities because one entity recognizes a level of detail which includes an occulting feature, while another does not. In the signal propagation case, different models can lead to differential visibility in RF, IR, or EO sensor systems due to intervening weather and terrain effects which are considered in one simulator and not the other.

7.2.2.3 Tactical Communications

Voice communication still dominates the toolbox of C^3 mechanisms. SIMNET implemented local-site voice communication by CB radio. When the long-haul link between the SIMNET site at Ft. Knox and the AIRNET site at Ft. Rucker was implemented, voice implementation was forced to the realm of digitized, packetized voice-- sent over a long-haul link parallel to the one supporting entity-state transactions.

7.2.3 EW Databases

Studies at Loral have determined that expanding/enhancing the PDU set to carry the increased magnitude and fidelity of data required to support EW simulation leads to bandwidths which are beyond the state of the art. Instead, ways must be found (and have been found) to characterize EW entities by datasets, host these datasets in shared databases, and model EW operations by only transmitting PDUs which contain pointer information into the databases, thereby communicating to receiving entities the nature of the active EW operations.

This approach significantly reduces the required communication bandwidths and suggests a new paradigm for the interaction of PDUs and common databases which states:

1) The best PDU has smallest size and lowest required update rate; and

2) The best shared database supports a 'best' PDU by providing sufficient stored data (rather than the transfer of data) to drive most of the simulation computation.

Identification of EW operational modes, and the organization of the common database according to these modes, yields the added benefit of reduced input data processing. As an example, it is better to store all of the modes of a given radar in the common database, and at run-time send an identification number which indicates which mode is currently active. Then, each radar-receiving entity can directly determine its simulation response to the active mode instead of having to analyze the raw data of a complex PDU for the purpose of discovering the active mode.

Work done at Loral will form the basis for investigations into the structure of the EW databases and the they will support. Loral has already demonstrated that PDUs which adequately address these issues can have as few as three (3) small fields for each emitter--when position data is already pre-stored (for fixed objects) or part of another PDU.

Below we will give examples of datasets which could accommodate different kinds of EW entities.

7.2.3.1 Radar

Radar emitters are employed in the electronic combat environment for a multitude of purposes. A radar can be used solely for target detection (early warning, height finding, search), for target monitoring (acquisition and tracking), or for weapon delivery (missile guidance, fire control). It can also be used for airborne navigation purposes, as in ground mapping, altimetry, or terrain following/terrain avoidance (TF/TA).

The characteristics of a radar in the Battlefield Database may be divided into two classes: static (ID, platform, type, function, waveform, scan type, etc.), and dynamic (modulation mode, antenna scan mode, PRF mode, etc.). The principle of database implementation of the EW environment calls for only transmitting dynamically changing characteristics of the radar, while referring to the database for information about its static parameters. The dynamic information set is limited, and in many cases can take the form of pointers into tables of static parameters. This technique greatly reduces the amount of data which must be transmitted between simulator nodes.

A candidate list of static characteristics for radar emitters follows. These data items must be appropriately distributed between the Simworld and the Battlefield Databases.

- Emitter ID
- Emitter platform (ground-fixed, ground-mobile, airborne, shipborne)• Emitter category (RF or IR)• Emitter function (radar, jammer, communications, navigation, IFF)
- Waveform (CW, simple pulse, chirp, FMCW, non-linear FM pulse, stepped FM pulse, phase coded pulse, analog or digital modulated CW)
- Interpulse stability (single stable RF, sliding RF, Multiple stable RF's, agile RF, diverse RF)
- Interpulse modulation (RF limits, frequency step size, modulation period)
- PRF/pulse mode (PRF stability, PRF modulation)

- PRF stability (stable, staggered, jittered, sliding, periodic, dwell and switch)
- PRF modulation (PRF limits, stagger ratio, modulation period, pulse width/duty cycle, pulse group definition)
- Antenna scan parameters (scan sector widths, scan frequency/period, special scan parameters)
- Polarization (horizontal, vertical, slant, right-hand circular, left-hand circular, elliptical, rotating

In low-fidelity simulations, only a limited number of these characteristics are required. However, if the database is to be compatible with advanced, high-fidelity simulations, then it should record the full set of characteristics.

Existing simulation standards and draft standards do not provide the above level of detail, which Loral has determined to be necessary based on our experience on SOF ATS, F-15, and other programs. Loral feels that this existing work on high-fidelity models should shape the DIS radar simulation standards. The proposed effort would evaluate the existing DIS standard and recommend changes to accommodate the next generation of high-fidelity EW simulation systems.

7.2.3.2 Infrared (IR)

Infrared (IR) emitters are sources of IR radiation. Steady-state IR emitters include exhaust plumes from missiles, rockets or aircraft; internally heated objects such as tanks (hot engines), jet aircraft tail pipes, or factory smoke stacks; ground fires; and solar-heated objects. IR emissions can be sensed by various passive (one-way) systems in the electronic combat environment, such as Infrared Warning Receivers (IRWRs), IR seekers in missiles, imaging infrared (IIR) sensors, forward-looking infrared (FLIRs) sensors, etc. In addition to steady-state emitters, several active (two-way) IR systems such as laser radars and laser target designators will appear to the sensing systems mentioned above.

As in the radar case, the characteristics of IR emitters on the Virtual Battlefield may be divided into two classes: static (ID, platform, type, function, IR band center, etc.) and dynamic (modulation mode, emitter size, position, etc.). As in the radar case, static data is retained in the database, while dynamic data is communicated via by special DIS PDUs.

A candidate list of static characteristics for IR emitters follows. Again, these characteristics need to be appropriately distributed between the Simworld and Battlefield databases.

- Emitter ID

- Emitter platform (ground-fixed, ground-mobile, airborne, shipborne)
- Emitter catagory (RF or IR)
- Emitter type (laser radar, laser target designator, exhaust plumes, internally-heated objects, flares, ground fires, smoke stacks, solar-heated objects)
- IR band center(s)
- IR bandwidth(s)
- Waveform (CW, simple pulse, chirp, FMCW)
- Interpulse stability (single stable, sliding, Multiple stable, agile, diverse)
- Interpulse modulation (limits, frequency step size, modulation period)
- PRF/pulse mode (PRF stability, PRF modulation)
- PRF stability (stable, staggered, jittered, sliding, periodic, dwell and switch)
- PRF modulation (PRF limits, stagger ratio, modulation period, pulse width/duty cycle, pulse group definition)
- IR scan parameters (scan sector widths, scan frequency/period, special scan parameters)
- Polarization (horizontal, vertical, slant, right-hand circular, left-hand circular, elliptical, rotating

In low-fidelity simulations, only a limited number of these characteristics are required. Even in current, advanced, high-fidelity IR simulations, fewer detailed characteristics are required than in the case of a radar emitter. However for standardization of RF/IR emitters, and to retain the capability to model future advanced threat environments, a list of characteristics such as the above should be considered.

7.3 Exercise Management

Initialization, restart, replay, and coordination of exercises is needed for practical realization of training and development goals in Distributed Simulation. These tasks, however, are somewhat in conflict with the basic principles of autonomous, non-centralized DIS system design. Section 2.3.1 described the distributed autonomous simulation entity, and how it does not depend on any higher level of coordination, synchronization, or control. Exercise management, however, is needed in order to repeat experiments and training scenarios to distill

Solar-heated objects without either internal heat sources or jet/rocket engines will not be further discussed herein, but they can be simulated by the image generator as part of a FLIR. Ground fires could be house fires, forest fires, field fires, or fires caused by battle damage. In addition, several active (two-way) IR systems such as laser radars and laser target designators will effectively appear as one-way IR sources to the passive systems mentioned above.

The characteristics of the IR emitters in the data base may be divided into two classes, static (ID, platform, type, function, IR band center, etc.) and dynamically changing characteristics (modulation mode, emitter size, position in many cases, etc.). As discussed previously, the implementation of DIS is greatly facilitated by only updating the required dynamically changing characteristics while all the static parameters remain retained in the data base of each computer

In low fidelity simulations, only a very limited number of these characteristics are required. Even in current advanced, high fidelity IR simulations, a lessor number of these detailed characteristics is required than with a radar emitter, for example. However for standardization of RF/IR emitters and the capability to consider future advanced threat environments, then such a list of characteristics should be considered.

Existing simulation standards, draft standards, and even proposed standards do not provide the above level of detail. However, Loral feels that the existing work can be expanded to consider the present standards and recommend changes to accommodate DIS.

Radio Communication The simulated battlefield will include communications among the entities participating in the battle. The fidelity of the implementation of communications capabilities will play a large role in determining the realism and training value of the simulation. Initial implementations of radio net simulation will be simple models considering Line of Sight, distance from transmitter to receiver, jamming on or off, and first talker control (i.e., second and subsequent simultaneous transmissions on the same frequency are ignored). Future implementations will improve the fidelity of this simulation by adding additional parameters such as environmental effects, multipath, jamming effectiveness, and multiple user interference. The communications models in the Simworld Data Base must support wideband intercept EW receivers and are interactive with the C3 net. Therefore the data base components will involve the signal characteristics (which can be identified with a single icon or mode number), unique message traffic to be simulated, tactics, equipment, location, and the link to the C3 net.

Because of the relative simplicity of radio traffic the data base components are relatively limited. Terrain (other than occulting of high frequency signals) is not important and normally simple one-over-r-squared computations will suffice for signal strength. Therefore the data base components to be defined in the proposed study are expected to involve the signal characteristics (which can be identified

useful data, and as an automated mechanism for bringing multiple simulation nodes into an exercise. This seeming inconsistency is resolved when the philosophical statements of section 2.3.1 are more rigorously defined.

A distributed simulation node is capable of joining or leaving the network at any time without affecting the continuation of the battle. Battle observers would simply see the entity (or entities) associated with that node appear or disappear. Exit from a battle can be achieved by several means, ranging from halting the simulation software to pulling the physical network wire off the simulation node. The SIMNET system was designed like this because of the potentially tremendous size of the networked system. If failure of a single node caused the entire system to halt, the utility, reconfigurability, and maximum size of the system would be severely curtailed. To implement this node independence, a "self healing" protocol was designed. By listening to the network for a certain period of time (in SIMNET, 30 seconds), a simulation node will receive all the environment updates it will need to reconstruct the Virtual Battlefield. From that point onward it sends and receives periodic updates when Virtual Battlefield conditions change. This is what is meant by saying that nodes are not dependent on any higher level of coordination, synchronization, or control.

It is possible, however, to implement higher level coordination alongside the "self healing" paradigm. Some of the capabilities necessary for battle coordination are:

- Entity activation/deactivation.
- Recording and replay of exercises.
- Restart exercises from known battlefield states.
- Remote configuration management.

SIMNET makes use of activation and deactivation packets within the simulation protocol. Activation/deactivation are transactions which require a response, and are addressed to specific individual nodes. Though this is a departure from the self-healing broadcast paradigm, it still permits the system as a whole to be independent of any specific node failing to respond. The DIS protocol currently recommends the use of interim activate/deactivate packets similar to SIMNET.

SIMNET makes use of Data Loggers which timestamp and record every packet on the network. Replaying the packets at the same relative interval as when they were recorded results in an exact re-enactment of the battle. No additional packets or paradigms are required.

Restarting exercises from a known battlefield state requires available simulation nodes, activate/deactivate packets, and one additional class of packets, referred to in SIMNET as status packets. Not all the information necessary to re-

activate an entity is contained in the entity appearance packet. The status packet contains low-bandwidth information corresponding to state variables which do not change very often. In SIMNET, these packets are transmitted once every 30 seconds, or whenever the status change warrants an immediate transmission. These are typically infrequent events. Restarting an entity at a known state can be accomplished with the above method. Restarting a SAFOR system at a known *cognitive* state is far more difficult. The inference processes present in a SAF system do not lend themselves to easy data encapsulation and network transmission. Restarting a SAF system from Data Logger recordings has never been attempted in SIMNET and is a subject of further research. The mechanism for SAF restart will probably resemble the status packet in nature, therefore will be supportable by the proposed DIS architecture.

Software configuration management, network operations, and physical system diagnosis are all necessary functions which fall outside the realm of the SIMNET and DIS simulation protocols. The real time restriction and self-healing paradigm can both be relaxed for this class of interaction. In SIMNET, a separate protocol, known as the management protocol, is used for these purposes. The DIS architecture is capable of supporting these interactions through expansions to the DIS protocol, and coordination of centralized management with the Cell Interface Units.