1-1-1995

Dynamic Terrain: Vision Document

Mark C. Kilby

Marty Altman

Curtis Lisle

Michelle Sartor

DYNAMIC TERRAIN -

VISION DOCUMENT

Contract Numbers N61339-92-K-0001, N61339-94-C-0004
US ARMY STRICOM

July 14, 1995
IST-TR-95-15

Authors:
Mark Kilby • Michelle Sartor • Curtis Lisle • Marty Altman

Reviewed by:

Art Cortes _____

Mark Kilby _____

Marty Altman _____

Michelle Sartor _____

Visual Systems Laboratory
Institute for Simulation and Training • 3280 Progress Drive • Orlando, FL 32826
University of Central Florida • Division of Sponsored Research

Table of Contents

1.0 Introduction

Dynamic Terrain, and dynamic environments in general, has been a goal of the military simulation community since the first high fidelity image generation computers emerged on the market in the early 1980's. This goal emerges from the desire to increase the fidelity of simulations to enhance the planning, training, and evaluation of military operations. With the advent of Distributed Interactive Simulation (DIS) in the early 1990's, the goal of dynamic environments in DIS exercises has become a paramount issue. This document describes some of the challenges of introducing dynamic terrain into DIS and the approach taken by the University of Central Florida's Institute for Simulation and Training to provide high fidelity dynamic environments in DIS.

This work is sponsored by the Defense Modeling and Simulation Office via the U.S. Army Simulation Training and Instrumentation Command (STRICOM) under contracts N61339-92-K-0001 and N61339-94-C-0004. Additional support or collaboration is provided through the U.S. Army Topographic Engineering Center, Naval Research Laboratories, the U.S. Army Waterways Experiment Station, and the U.S. Army Engineer School.

1.1 Executive Summary

1.1.1 Background

This Vision Document describes the challenges of Dynamic Terrain (DT) and the approaches to developing DT for Distributed Interactive Simulation (DIS). Incorporating DT into training simulators has been a complicated process largely because the term "dynamic terrain" means different things to different people; therefore, accommodating these differing perceptions can be difficult. In addition, the transformation of static to dynamic representations in future simulators contains several problem areas, many of which are large themselves. The principal research questions in this project have been: how should dynamic terrain in the current simulator context be implemented, and how should knowledge gained in answering the first question to provide for further undetermined dynamic terrain capabilities be used? Meeting the requirements of interoperability, complexity, and abstraction have been important considerations in overcoming the challenges of Dynamic Terrain in DIS. The solution developed by the Visual Systems Laboratory (VSL) at the Institute for Simulation & Training (IST) of the University of Central Florida is termed "Shared Environment."

The relevant information for this Document comes from the literature and from research experience conducted by the VSL under the sponsorship of the US Army's Simulation, Training, and Instrumentation Command (STRICOM). To answer the research questions, data structures and algorithms that allow physically realistic real-time manipulation of simulated soil and water have been explored.

The major conclusions that emerge from this study include:

- A classic tradeoff between speed and fidelity is involved in providing dynamic envi-

ronment effects driven by numerous unpredictable events within a simulation of the real world.

- All individuals within a DIS exercise can cause some dynamic environment effects or can see the results of dynamic environment effects in forms appropriate to their particular simulator.

- DIS architecture solutions must be flexible enough to provide a dynamic, interactive environment.

    - The architecture should support multiple types of players, applications environment models, and environmental effects models.
    - The overall simulation architecture cannot be based on one vendor's approach to environment representation.
    - The architecture should be able to support the different resolutions and data requirements of diverse network simulators.

- Address these challenges and determining how the DIS architecture may be improved to provide dynamic environments and terrain, requires an object-oriented modeling and iterative prototyping approach.

    - Object-oriented approach is an iterative, incremental approach in which both the understanding of the problem and the design of a solution for that problem evolve over time from the general toward the specific.
    - With each iteration, both the knowledge about the problem and the design for a solution improves incrementally in a developmental spiral.

- New data structures are required for the Shared Environment DT solution.

    - Polygonal representation works less optimally for DT applications such as terrain following, terrain reasoning, and environmental representation.
    - Representations with more mathematical expressiveness, such as the mathematical surface, provide alternative representations while maintaining compatibility with polygonal systems.
    - The nonparametric surface family is currently used but may require future extensions.

- The active database concept represents a fundamental change to how simulators represent environments. Dynamic Terrain database abstractions support an arbitrary number of attributes, for example, soil strength, temperature, and water depth, within the active database.

- The active database query mechanism is intentionally decoupled from underlying representations in Dynamic Terrain.

    - Clients of the active database are isolated, object-style, from how the data is

stored.

- Within the active database, data is bilinearly interpolated between quadruples of data points in world space to find the locations of the underlying data.

- There should not be a single reconfigurable client-server simulation architecture for an entire network. Instead, a loose hierarchy of servers should be used.

  - A central server is the obvious solution, yet it will have problems with throughput when serving large numbers of simulators.
  - A hierarchy of servers provides the required flexibility while avoiding problems with data redundancy, latency, and loose coupling.

1.1.2 Current and Future Research

Examples of the current research in Dynamic Terrain at IST are contained within the Dynamic Terrain Developers' Kit which is available through the Tactical Warfare Simulation and Technology Information Analysis Center (TWSTIAC). Documentation and software is available for components of the Shared Environment which include entity, terrain, and fluid services; DT database used in the terrain service; and an abstract service for developing new types of Shared Environment services. Also available are DT Resources which include soil, fluid, crater, track, thermal, and minefield. Challenges addressed by the Shared Environment architecture and its components include open systems designed to support heterogeneous simulators, support multiple resolutions, support different run-time data requirements, provide consistent representation, and provide flexibility and extensibility.

Future research in Dynamic Terrain will require further work to develop approaches that work seamlessly in a DIS environment. To accomplish this, the following research questions must be answered:

- How should responsibility between multiple servers be divided?
- How should more sophisticated hierarchies of servers be supported?
- How should boundary conditions be handled?

1.1.3 Conclusion and Recommendation

The complexity associated with implementing a shared, dynamic, unscripted distributed simulation is quite high and is often underestimated. The Shared Environment, one potential solution path, provides many tools for addressing this complexity. The client/server approach used in the Shared Environment provides a consistent common interface to the state of the simulated environment. The environment state, a natural extension of the DIS protocol, can be considered in a manner consistent with vehicle state. In addition, the client/server approach allows software encapsulation and makes prototype software easier to transport between different architecture. However, many problems remain unresolved and must be addressed by further research.

1.2 Scope of Document

This document provides an overview of the obstacles and approaches to implementing Dynamic

Terrain in Distributed Interactive Simulation. In particular, a focus is placed on the specific challenges imposed by interactive simulation and, more generally, DIS, and how researchers at the Institute for Simulation and Training have devised a solution to this problem.

This document assumes that the reader is familiar with basic concepts of the Distributed Interactive Simulation standard, computing networks, and training simulation for team training as well as command and control. For more information on these subjects, refer to the DIS Vision Document and other documents on DIS which are available from the Tactical Warfare Simulation and Technology Information Analysis Center (TWSTIAC). The reader is also directed to the book on flight simulation by Rolfe and Staples.

1.3 Background

For many years, the military simulation community has been working toward incorporating dynamic terrain into training simulators and other simulations to enhance their effectiveness. The phrase "dynamic terrain" is one that means different things to different people. Some think of being able to place craters or dig holes in the ground. Some focus on being able to knock down trees or bridges or buildings. Some are interested in lakes, rivers, or oceans. Some want to see smoke, clouds, or weather. Another area of increasing interest is sensors (infrared, radiometric, and others). The list of desired dynamic environment effects is always increasing.

It is apparent that this is not simply a matter of performing slight modifications to today's simulation systems. Transforming things that are either static or not present in current simulators into dynamic things in future simulators contains several problem areas, many of which are large themselves. These include computational resources, image generation, software architectures, network bandwidth, real-time physics, data acquisition for simulation initialization, man-machine interface, fidelity, instructional design, and training effectiveness.

It has proven useful to break the "How do we do dynamic terrain?" question into two phases. First, "How do we do dynamic terrain in the current simulator context?" That is how can we use as many of the current assumptions about simulators as possible and still provide a reasonably general dynamic terrain capability? The assumptions are covered in more detail below, but generally, the context is represented by the requirements and capabilities of today's simulators. The second question is, "How do we use the knowledge gained in answering the first question to broaden the context of simulation so as to provide for more prolific, or more detailed, or as yet undetermined dynamic terrain capabilities?" There is much to learn from the first question before addressing the second.

1.3.1 A History of Dynamic Terrain at IST

The Visual Systems Laboratory at IST has been working for the last several years on the problem of Dynamic Terrain (DT) under the sponsorship of the US Army's Simulation, Training, and Instrumentation Command (STRICOM). Dynamic Terrain refers to the manipulation of a simulated terrain database during an interactive training simulation.

Historically, terrain databases for training simulators were almost exclusively for the visual

systems. Most early simulators were flight simulators, which had relatively modest requirements for the terrain database. As time passed, it became more important for the terrain database to correspond to a particular geospecific location, and for the database to contain more information.

Another important characteristic of current terrain databases is that they are static. There is no mechanism to alter the database during the simulation. This constraint is due in large part to the effort involved in creating a terrain database. Even today, when parts of the process are automated, databases still require significant hand crafting. A good high-level description of visual databases can be found in [Sieverding94]. Simply speeding up the process of editing the database is not likely to provide the desired levels of interactivity. A higher probability of success lies in restructuring the problem. Through the STRICOM Dynamic Terrain project, researchers in the Visual Systems Laboratory at IST have explored data structures and algorithms that allow physically realistic manipulation of simulated soil and water in real-time. They are also considering how multiple players can see and produce changes in the simulated terrain through use of the DIS paradigm.

A closely related issue is the changes in the database and networking protocols necessary to support higher fidelity requirements of multiple interacting players. Many attributes of the terrain can be made available. For example, vehicle mobility calculations require soil strength. Soil dynamics models need soil type and density. A hydrology model requires the rate of absorption and current moisture content. Consideration must also be given to how these attributes are changed by multiple players and distributed across a network without impeding interaction between the players and the synthetic terrain.

To address these issues, three concepts must be considered which are interoperability, complexity, and abstraction. These concepts are described below.

1.3.2 On Interoperability

Interoperability is a concept that is neither fully defined nor completely understood [Mamaghani94, Cortes94, Kamsickas93, Downes-Martin91, Fullmer90]. It is related to the connection of dissimilar systems in the same virtual space, but it is surely more than that. From a subjective standpoint, interoperability is achieved if the perception of the virtual space is sufficiently similar when viewed from different simulators. From a more technical standpoint, interoperability is difficult to define because "sufficiently similar" is highly dependent on the objectives of the simulation.

An issue that has a direct bearing on achieving interoperability is the fidelity levels of the simulations. This can take two forms. The more obvious has to do with mixing fidelity levels. (1) Trying to interoperate simulations becomes increasingly more difficult as the difference in fidelity levels increases. The difference in fidelity level can also be referred to as the "fidelity differential" [Knight90]. A corollary is (1a) trying to interoperate simulations becomes increasingly more difficult as the number of fidelity levels to be included in a given scenario increases.

Another interoperability difficulty associated with fidelity level is (2) trying to interoperate simulations becomes increasingly more difficult as the overall fidelity of the models increases.

This one is slightly more subtle and is related to the complexity of the higher fidelity simulations. Higher fidelity models typically display added functionalities, more encompassing semantics, and the associated increases in complexity. If these higher fidelity models were developed by different organizations working under different projects with different goals, it is likely that a significant amount of work would be required for these models to interoperate at all.

As the discussions about interoperability continue, three issues become more clear. Two are well stated by Woodard and the third by Riecken,

Requirements for interoperable environments must be developed, through analysis and experimentation, to define metrics for interoperability based upon the type of entity simulated and tasks to be performed on the network [Woodard94].

Contributions by the entire simulation community toward solutions are needed to achieve the goals of interoperable simulation networks [Woodard94].

Increasing interoperability means making explicit and public as many of these expectations as possible [Riecken93].

IST has also engaged in studies that partitions the problem and identifies quantifiable parameters affecting interoperability. The ultimate goal, "is to define interoperability in a user-oriented framework" [Cortes94].

1.3.3 On Complexity

The inherent complexity of a battlefield environment (virtually any environment for that matter) is obvious. This stems from such things as the large number of elements in the environment, the number of possible states for each element, and the interactions between elements that cause state transitions. It is also clear that any system created to simulate such a complex environment to any degree beyond the trivial must itself, be a complex system. Here a complex system is one that contains a large number of components that interact in nontrivial ways.

It is important to note that there are limitations to the amount of complexity that we as humans can deal with at any given point in time. Experiments by psychologists tell us that the maximum number of chunks of information that an individual can simultaneously understand is seven plus or minus two [Miller56]. The good news is that, "Complexity frequently takes the form of hierarchy and that hierarchic systems have some common properties independent of their specific content" [Simon81].

A hierarchical system contains components whose own structure is made up of subcomponents each with their own semantics. In some sense, a component's behavior is a composite of the behaviors of its subcomponents, but it is often more. Consider an airplane which has an engine to provide thrust, wings to provide lift, and controls that alter control surfaces which change the flight path. The behavior of the airplane is neither that of an engine nor that of a wing, but is a synergistic combination of the behaviors of all its components.

One useful attribute of hierarchical systems is the distinction between interactions among components and interactions within components. Simon goes on to say,

Intracomponent linkages are generally stronger that intercomponent linkages. This fact has the effect of separating the high-frequency dynamics of a hierarchy-- involving the internal structure of the components--from the low-frequency dynamics-- involving interaction among components [Simon81].

1.3.4 On Abstraction

Abstraction is a powerful means by which humans cope with complexity. An abstraction can be strong or weak in terms of its expressive power. It is possible for any particular abstraction to be strong in one sense and weak in another as measurement of the strength of an abstraction must be based on the semantics to which the abstraction applies. Booch's definition is,

An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provides crisply defined conceptual boundaries, relative to the perspective of the user [Booch94].

Consider this definition closely. An abstraction should capture the essential characteristics, or the key semantics of an object. A mature abstraction will provide crisply defined conceptual boundaries making it easier to understand what the object represents. A good abstraction will be specified relative to the perspective of the user, that is to say within the user's context. Booch later clarifies, "Meaningful abstractions can never be crafted in the absence of a well-defined context" [Booch94].

These concepts of Interoperability, Complexity, and Abstraction have been driving forces in overcoming the challenges of dynamic terrain in DIS. We present these challenges next.

2.0 The Challenges of Dynamic Terrain

Providing a terrain environment that is responsive to events that occur during a Distributed Interactive Simulation exercise presents several challenges. These challenges not only exist for development of Dynamic Terrain, but they also arise when providing any dynamic environment effect within the DIS paradigm. These challenges are described in the sections which follow.

2.1 Interactive Simulation and Real-Time Environment Changes

Providing dynamic environment effects that can be initiated by numerous *unpredictable* events within a simulation of the real world presents a classic tradeoff between speed and fidelity. This particular tradeoff is no different for one simulator or a network of simulators. There are numerous ways to approach this tradeoff to implement dynamic environment effects. Since training is a primary application area of DIS, the visual components of dynamic environment effects are considered of high importance. However, to provide a solution to the problem of real-

time environment changes in an interactive simulation for a wide variety of applications requires emphasis on behavioral as well as visual aspects of the environmental change. In other words, the environment must not only look different, it should behave differently due to the event in the simulation which caused the environmental change.

For instance, to represent a crater resulting from an air-deployed munition explosion could require a simple visual model placed in the location of the explosion. Such an approach requires minimal time to produce the desired dynamic environment effect. For a particular training application, it may become important to represent craters for different types of munitions. Based on this condition, different visual models of the crater could be developed. If varying soil conditions are critical to the scenario, then different visual models can be developed based on soil type and munitions type. If angle of impact also produces a significant visual effect, then visual models will be developed that vary with all three parameters. Each characteristic of the scenario that is used to produce that environment effect can produce numerous visual models that must be stored, indexed, and accessed during the simulation. If a sufficient number of models are generated, then the time to retrieve the appropriate visual effect model may become prohibitive. Consider that a particular scenario may use 20 different munitions types in a gaming area with five different soil types and angle of impact specified in five degree increments ranging from 90 degrees (straight vertical drop of munitions) to 20 degrees from horizontal. This would produce 20x5x15 models or 1500 visual models for craters, which is only one possible dynamic environment effect. Perhaps a DIS exercise for training may not require this much variation (i.e., fidelity). However, it is possible that other dynamic events or other applications may require this level of fidelity. For instance, this example does not consider the behavioral effect on the terrain, such as the change of the terrain trafficability due to the cratering and explosions.

A second approach to provide real time dynamic effects is to calculate the effects, both visual and behavioral, via algorithms. Several algorithms may be used to calculate the different effects resulting from an environment changing event. Using our previous example, one algorithm may be used to calculate the shape of the crater due to soil type, munitions type, and angle of impact. A second algorithm may determine changes to soil properties that effect trafficability. Such approaches can require more time to produce the effect than a precalculated model. However, calculation of the effect can provide greater variety to the effect based on significant factors of the simulation scenario. Using the previous crater example, a single algorithm could be used to calculate the disturbance to the soil based on the type of munition, soil type, and angle of impact. This is much easier to manage in a software system design than the 1500 models of the previous example. Thus, we trade the speed at which the effect is produced in the virtual environment for greater flexibility in how that effect is represented and managed within the simulation system.

A typical approach to apply algorithms for dynamic environment effects is to modify existing analytical models. This approach possesses some pitfalls in implementing dynamic environment effects. One pitfall is that these models cannot be used "as is" and require modification due to the differences between an analytical simulation and a real-time immersive environment. An analytical simulation is typically developed to model some physical phenomena within the known boundaries of the assumptions of the model. For instance, an analytical model of a ground vehicle such as a tank may model drivetrain, suspension, brakes and other systems to an accuracy of ±5%. That model can then be used to predict how the vehicle will perform under various conditions

which can be simulated. Thus, the analytical model is designed for computational completeness and accuracy and therefore assumes that a certain sequence of calculations always takes place. In contrast, a model developed for an immersive virtual environment (IVE) emphasizes execution time over accuracy of the model. In an IVE, a visual scene must be produced within a certain time frame. By making the time frame sufficiently short, the illusion of smooth, realistic motion of objects in the IVE is produced. Thus, each IVE model must execute within a certain segment of the time frame so as not to disrupt the illusion of the entire scene. To accomplish this, the IVE model may also execute a certain sequence of calculations, but that sequence will be cut short if they require more time than allowed in the time frame. For instance, consider a changing visual scene which represents a vehicle driving through the countryside. As the view changes due to vehicle motion, the scene first contains seven trees and then contains a forest of 100 trees. Based on the computer producing the images, it is possible that all the individual trees will not be rendered due to insufficient time to calculate the three dimensional perspective within the given time frame of the computer. Thus, fidelity must be sacrificed for speed in an IVE. Thus, converting an analytical model to an IVE model changes the emphasis of the model from a computational emphasis to a time emphasis. This introduced additional assumptions into the model that may conflict with the original assumptions or compound the assumptions that make the model difficult to use. However, *it is much easier to remove fidelity and then optimize the simulation for speed than to optimize first and add fidelity later.*

A second pitfall of adapting existing analytical models is the data required by the models. Typically, such models are developed to represent behaviors of some object on a broad classification of conditions and not specific conditions in the environment to be simulated. This will also reduce the fidelity of the dynamic environment effect to be represented. For instance, a ground vehicle simulator may be developed for mission rehearsal which may take into account effects on the vehicle's mobility due to terrain and weather conditions. Furthermore, the users may wish to train in an IVE that replicates a geospecific location. The data for that area of the world may not be available either through restrictions on distribution, poor distribution, or difficulty in collecting the type of data required for the model. In such cases, an estimate of the data must be fabricated which induces errors to the simulation beyond the errors introduced by the assumptions of the model. Thus errors due to the input data as well as errors induced from assumptions and modifications to the model must be examined to determine if a model will perform sufficiently for an IVE.

A final challenge to implementing unpredictable dynamic environment effects is the basic design assumption of many contemporary simulators which states that the environment is unchanging. In the early days of training simulation, simulators were designed for short scenarios so that individuals would learn specific skills [Rolfe86]. Such scenarios may have lasted a few hours at most and it could be assumed that weather conditions, time of day, and soil conditions would not be altered significantly. To meet the timeframe requirement of IVEs, simulator hardware was built on this assumption. As multiple individuals now begin to participate in such simulations, scenarios are projected to last an entire day or longer and changing environmental conditions are immanent. Thus, to effectively implement dynamic environment effects in IVEs will require a fundamental change in simulator design. In addition, it is unlikely that existing simulators will be able to adapt to this new capability without an extensive retrofit. An analogous situation would be attempting to run an Indy 500 car in an off-road endurance race. The Indy car is designed to travel

at a high rate of speed, but it does so in an environment that changes very little. In an off-road endurance race, a vehicle must be designed to adapt to numerous and changing environmental conditions and therefore trades flexibility for speed in its design.

2.2 Distributed Simulation

The Distributed Interactive Simulation standard assumes that multiple heterogeneous simulators will communicate across a network so that participants within the individual simulators feel that they are participating in the same events as all other participants in the exercise [IST93a]. For dynamic environment effects, this implies that *all* individuals within a DIS exercise can cause some dynamic environment effects or see the results of dynamic environment effects in forms appropriate to their particular simulator. This presents several challenges which are listed below.

2.2.1 Open Systems Design to Support Heterogeneous Simulators

One of the greatest challenges lies in the interconnection of systems from many different vendors. Working in a heterogeneous computing environment is inherently more difficult than in a homogenous one. Functional and implementation differences between interacting simulators can yield unrealistic advantages for some and deficiencies for others that can adversely affect the training objectives [IST93b]. These interoperability differences are most often the result of differing semantic interpretations, subtle or otherwise. These semantic interpretations are directly incorporated into the design of each simulator. Thus, a key challenge is to provide system-level support for dynamic environment effects within a DIS exercise that will not be biased toward a particular vendor. Thus, the overall DIS simulation architecture cannot be based on one vendor's approach to environment representation. Also, the architecture should be able to support the different resolutions and data requirements of the diverse simulators that may exist on a network. For instance, a flight simulator may only need to visualize large patches of terrain and vegetation at low resolution, but a ground vehicle simulator may require high resolution terrain profile, vegetation cover, and soil type to determine how well it traverses the terrain and presents a realistic representation of the vehicle's motion.

2.2.2 Support for Current and Future Vendor Technology

A related issue for DIS is the goal of interconnecting both new and existing systems. Fielded versions of existing simulators are typically referred to as legacy systems. This goal of retrofitting existing simulators to incorporate new technology has understandable political and economic advantages. However, as often as this concept gets mentioned, many people overlook the difficulty of this goal. The engineering effort required to retrofit existing systems so that they can interoperate with new dynamic environment functionality can be significant. As mentioned previously, such a retrofit can imply a significant change in the design semantics which requires extensive modification to the simulator. Some have suggested that simple modifications to the legacy systems are possible and only an interchange of data is required. Despite the fact that there has been limited success with such endeavors, Mamaghani warns that:

Interchange of data does not guarantee interoperability. Just because two systems can babble on

the network and exchange data does not mean that they have interoperated [Mamaghani 1994].

In addition, new systems which are designed to operate within the DIS paradigm must be able to exhibit new capabilities and yet still interoperate with these legacy systems.

2.2.3 Sufficient Performance

Any architecture that supports dynamic environments in DIS must also support several different types of simulations. Therefore, the architecture must support changes to the environment and provide these updates to ***all*** simulators in a DIS exercise at a rate at least twice as fast as the update rate of the fastest simulator in the exercise, according to communications theory [Couch87]. This does not mean transmitting PDUs at this rate. The current DIS standard provides mechanisms for reducing the communication of state changes of entities through dead reckoning mechanisms [IST93a]. Thus, the architecture must support the data update rates of the individual simulators while reducing the communication between simulators to avoid communication bottlenecks. Furthermore, it is likely that "sufficient update rates" will vary from exercise to exercise since different exercise scenarios will require different simulators to be used.

2.2.4 Consistent Representation

The simulation architecture must also provide a uniform representation of the environment to all entity simulators and a uniform methodology for accessing and changing the environment to players in a DIS exercise. Thus, a standard mechanism must be developed to view and induce environmental changes by the entity simulators. This mechanism must allow for environment data to be translated to and from the simulator's native representation to a common representation for use by other simulators in a DIS exercise. The intent is to reduce correlation and interoperability problems described previously.

2.2.5 Scalability

Another goal of the DIS standard is to support forces of varying sizes. Therefore, architectural solutions should work equally well for 10 or 10,000 entities. This goal presents an additional challenge to incorporating dynamic terrain in DIS exercises. The architecture should support an increasing number of entities and their interaction with the environment through the addition of processing resources only. No change to the architecture design should be required.

2.2.6 Late-Joining Players

Players joining after the start of an exercise must be able to rapidly update their environmental database to match the current simulated environment. This implies that any changes made to the environment by other simulated entities should be available to the new players.

2.2.7 Fault Tolerance

Due to the amount of resources that may be dedicated for a particular DIS exercise, the architecture should be minimally robust to prevent interruption due to minor errors. Since the environment, and the terrain in particular, are key components of most military training

simulations, a catastrophic error in the dynamic terrain simulator can potentially affect all other simulated entities and bring the entire exercise to a halt. Reinitializing a simulation exercise due to a dynamic terrain simulation failure is an unacceptable and expensive option. Therefore, the architecture should be able to compensate and recover from such errors during an exercise so that a minimal number of players are affected.

2.2.8 Sufficient Realism for Multiple Applications

Another major challenge facing DIS is the wide scope of applications. Early implementations of DIS, like SIMNET before it, were large vehicle-level virtual simulations. However, the recent past has seen an increasing desire to broaden DIS not only in the areas of testing and evaluation, but also to expand DIS to handle interoperation with constructive and live components. Thus, DIS is not only being developed for training, but for other application areas such as mission rehearsal, doctrine development, virtual prototyping of new equipment, and others [IST94]. Each of these application areas and the technologies used to support them has added a variety of requirements to DIS that are complex and potentially conflicting.

2.2.9 Flexibility and Extensibility

Architectural solutions to provide a dynamic, interactive environment require flexibility in a number of areas. First, the architecture should support multiple types of players whether they be man-in-the-loop simulators, constructive simulations, computer generated forces, or a combination of these. Second, multiple types of applications must be supported including training, testing, mission planning, and mission rehearsal [IST94]. Also, the architecture must support multiple environment models and environmental effects models. An environment model refers to a simulation of naturally occurring phenomena while environmental effects refers to the effect of this phenomena on man-made devices or other natural phenomena. Therefore, rainfall would be provided by an environment model while effects on sensors and vehicle mobility would be provided through environmental effects models. These objectives for flexibility not only apply to current applications, but future applications as well. Thus, the system developed should accommodate future applications with minimal modifications and should be easily reconfigured to accommodate different types of simulations, different applications, and new models.

3.0 Approaches to Developing Dynamic Terrain for DIS

The challenges presented previously of "fidelity vs. speed" and DIS, can be approached in multiple ways. Researchers at IST have explored several approaches while following an object oriented methodology and an iterative prototyping scheme. In particular, we have examined the data structures, models, and architectures that would make dynamic terrain and dynamic environments possible in DIS. These approaches are summarized below.

3.1 Initial Assumptions

To address the need to remain compatible with current simulation technology, several

assumptions were made. These assumptions were followed throughout the prototyping of various architecture designs to provide dynamic terrain in DIS. These assumptions are listed below.

3.1.1 2 1/2 Dimensional Representation

Even though simulated entities have the freedom to move about freely in the three dimensional space of the virtual environment (subject to their own capabilities), the terrain is often referred to as being 2 1/2 dimensional. In other words, the only part of the terrain that is usually relevant to a given scenario is the surface or "skin". This same assumption is followed by most, if not all, image generators.

3.1.2 Polygon Based

Many internal simulation models, as well as most current image generators, are polygon based. The terrain is essentially a large (open) polyhedron.

3.1.3 Elevation Source Data

The elevation source data for any given terrain often comes in a gridded format where for each x,y location there is one associated elevation value. Several resolutions, or grid spacings are available, but the resolution is typically no finer than 1 arc second (roughly 30 meters). If smaller grid spacings are desired, the implication is that some form of mathematical interpolation will be performed to find intermediate values.

3.1.4 Feature Source Data

In current training simulator technology, features represent any object that is located on the terrain and is not represented by a specific entity simulation. Examples include rocks, trees, roads, rivers, and buildings. The feature data for any given scenario often comes in some two dimensional format where features are represented as projected points, lines, or areas. The 2D representation implies that some polygonization must be done and that features must be "planted" or "decaled". For example, a road might be represented as a sequence of 2D points (a polyline). In order to work with this road, its width is used with the original points and the underlying terrain profile to determine appropriate "road polygons".

3.1.5 No Multi-Valued Areas

There is no concept of multi-valued areas (x,y locations having more than one elevation value), hence there is no generalized mechanism for representing say a cave or a tunnel. Such things, if modelled, are considered as features or are otherwise handled as special cases.

3.1.6 Multiple Attributes

The current context for simulators does include the concept of multiple attributes for any given x,y location, even though this is often underutilized. In some cases, the only attribute of interest is the elevation profile. Often, other attributes are used such as colors and texture coordinates. Sometimes attributes such as soil type are needed. In any case, it is reasonable to assume some

number of attribute values are associated with any given x,y location.

3.2 Object-Oriented Modeling and Iterative Prototyping

To address the challenges presented previously and to determine how the DIS architecture may be improved to provide dynamic environments and terrain required a new approach in tackling DIS problems. The cornerstone of the approach is the use of object oriented analysis and design techniques [Booch91, Rumbaugh91, Meyer88, Entsminger90, Mullin89, Nelson91]. The object oriented paradigm is characterized by an iterative, incremental approach in which both the understanding of the problem and the design of a solution for that problem evolve over time from the general towards the specific. A cycle is established roughly as follows:

1. Classify the level of understanding - Basically, "How mature is the current understanding of the problem?" This can range over a spectrum from very nebulous to rather clear.
2. Identify and enumerate high-risk areas - At the current level, what are the areas that will have the most impact on the architecture of the solution? What things are potential problem areas?
3. Make conscious decisions about the high-risk areas - Which of the high-risk areas will be addressed in the current iteration? The decision is based on the level of understanding about the particular problem as well as the relative priority of the problem. It is often useful to push a problem into the next iteration when the level of understanding should be higher.
4. Choose additional functionalities - Make decisions about added functionality or higher fidelity for components of the system that will be carried over from the previous iteration.
5. Design this iteration of the solution - Using general knowledge about the problem, any knowledge gained in previous iterations, and what is known about the high-risk areas to be dealt with in this iteration, design a solution. It is sometimes desirable to modularize the high-risk problems so that several approaches can be implemented and analyzed. This might spawn a subtask to take a look at several approaches before returning to the design.
6. Analyze and critique the iteration - This helps clarify what was learned in the iteration and feeds the next cycle.

This cycle is iterated, and with each iteration both the knowledge about the problem and the design for a solution get incrementally better. Some iterations will contain major renovations in the design, and others will not. Typically, the major changes will be worked out in the early iterations. That is the goal of going after the high-risk areas early rather than putting them off.

The high risk areas pursued in IST's Dynamic Terrain research are the challenges addressed previously and each was addressed in this iterative process. In particular, issues of the models, architecture, and data structures were addressed in each iteration which is referred to as the Development Spiral [see Figure 1]. In our research, the Development Spiral first addressed the

creation of a model for a specific dynamic environment effect. Then this model is integrated into the simulation architecture. In some cases, the architecture must be adapted to accommodate the model and in other cases the model is adapted to fit the architecture. The next loop of the spiral begins with a new dynamic environment model. With each pass through a loop of the spiral, new functionalities are explored for providing dynamic environments in DIS and new techniques are discovered as to how DIS and distributed virtual environments may evolve to accommodate a more dynamic simulated environment.

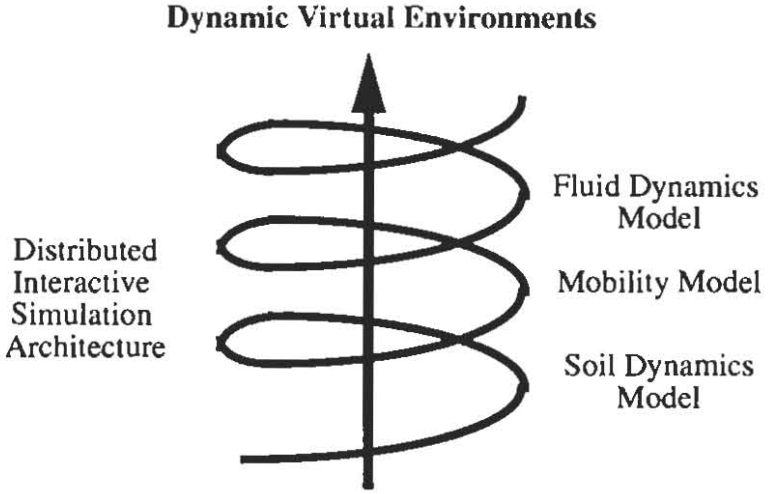

Figure 1. The Iterative Development Spiral

This development spiral approach was followed throughout the Dynamic Terrain research program for incorporating effects such as dynamic soil, hydrology, and vehicle mobility into the simulated terrain. This approach allowed us to examine and develop three versions of a distributed simulation architecture, the latest of which is presented in the accompanying documents. This architecture proves to be the most flexible, extensible, and most capable at supporting current and future simulation systems.

3.3 New Data Structures for Dynamic Environment Effects

The robustness and extensibility of a software system strongly depends on the data abstraction selected. Consider that systems simply access and affect the values of the inherent data structure. If this is so, the choice of data structure is critical since all the functions a system performs will be accomplished by affecting the state of its internal data structure.

Most simulation and image generation systems on the market use a polygonal representation for the environment and the objects contained in the environment. For example, the terrain surface is composed of a mesh of triangles. Trees, tree lines, and forest tree canopies are represented by textured polygons standing above the skin of terrain triangles.

It is our position that the polygonal representation works well for rendering (i.e, viewing) in high-

performance graphics systems but less optimally for other simulator applications such as terrain following, terrain reasoning (e.g., for computer-generated forces), and environmental representation. We believe that these and other non-rendering applications for simulators will be experiencing the most growth in networked simulator design over the coming years.

It is understandable that the polygonal database is the reference used by today's simulators. However, as we look for simulators to perform at a higher fidelity level, we should be willing to consider a more robust data structure to support the additional fidelity.

Terrain following algorithms which involve real vehicle dynamics require smoothly changing terrain slope contours in order to correctly simulate the forces on the vehicle. If the terrain is represented using a polygonal mesh, the slope changes suddenly as the vehicle encounters a polygon boundary -- inducing an anomaly in the vehicle behavior which is uncharacteristic.

3.3.1 Mathematical Surfaces as a Basic Data Structure

As part of our goal of exploring alternative representations while maintaining compatibility with polygonal systems, we sought a representation that had more mathematical expressiveness while still providing the basis for a polygonal form. In this context, a good representation is one that is both clean and easy to use, and at the same time has enough expressive power to represent a broad spectrum of terrain attributes.

One alternative that has proven particularly useful is the mathematical surface. Note that the term mathematical surface is used generically to refer to any of several representations that might be used for any particular terrain attribute. Tessellating a surface to generate polygons is a relatively simple operation that is based on a more deeply rooted concept. *Mathematical surfaces can be sampled anywhere, not just where there is a data point.* This property overcomes the errors induced from sampling a polygonal or elevation grid representation. This concept is crucial to our further developments.

In particular, we have currently chosen to work with members of the nonparametric surface family, while recognizing that some future problems may be best represented with fully parametric surfaces. A class hierarchy was designed to encapsulate these representations. What is most important about this class hierarchy is recognizing that each of these surface representations is functionally equivalent to the others. While certain terrain attributes might be best modelled with a particular representation, any other representation could be used with predictable differences. Architecturally speaking, it is important that any attribute be able to be modelled with any representation.

3.4 Dynamic Update of Terrain State - The Active Database Concept

A software database abstraction was constructed to support an arbitrary number of attributes (e.g. soil strength, temperature, and water depth) for a particular area in the terrain database. The assumption is that simulator system design will have less complexity if it is built on a solid

fundamental data abstraction. Our abstraction, the multi-plane active dynamic terrain database, is shown in Figure 2. All attributes are layers in the conceptual model of the DTDB. Therefore, any application programs which are built using this abstraction have immediate access to any attributes contained in the DTDB.

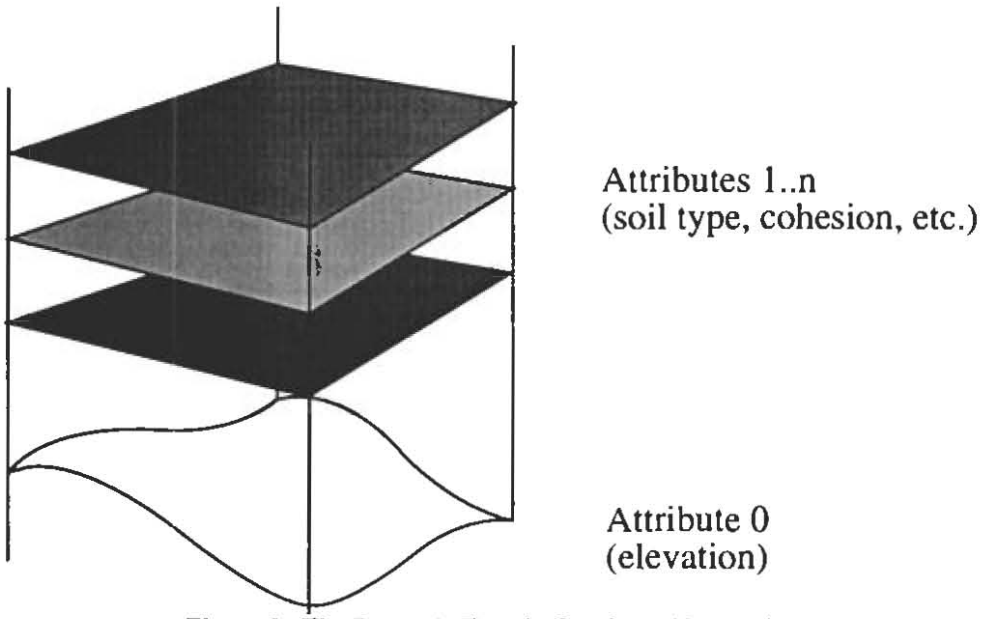

Figure 2. The Dynamic Terrain Database Abstraction

3.4.1 The Database Query

The fundamental question to be asked of an active dynamic terrain database element is:

> *"What is the value of attribute attr at location x?"*

This question serves as the basis for all of the particular queries made by *any* application that uses the DynamicTerrainDatabase class. That is, any application should be able to retrieve any terrain attribute (e.g., elevation, slope, vegetation, soil type) with the assumption that this is the most current and accurate value. This assumes that the DynamicTerrainDatabase class can be queried for any subset of any attribute within the database at any resolution and at any time while changes are occurring.

3.4.2 The DTDB Abstraction

It is important to note that we have intentionally decoupled the query mechanism from the underlying representation. *Clients of the active database do not have, nor should they have, any preconceived notions about how the data are stored.* The original data may in fact have been an elevation grid, but to the user of the active database it shouldn't matter. As shown in Figure 3, the client application queries the database with specific expectations of the form, size, and resolution of the returned data. Within the active database, the data is then transformed from the internal data structure into the format anticipated by the client application with the client application being completely unaware of this transformation. This decoupling capability is a direct benefit of the

object oriented approach described previously and will prove even more useful as new types of data are incorporated into the system as a whole. It stands to reason that other desirable attributes may be better represented by something other than "standard" terrain structures (such as an elevation grid).

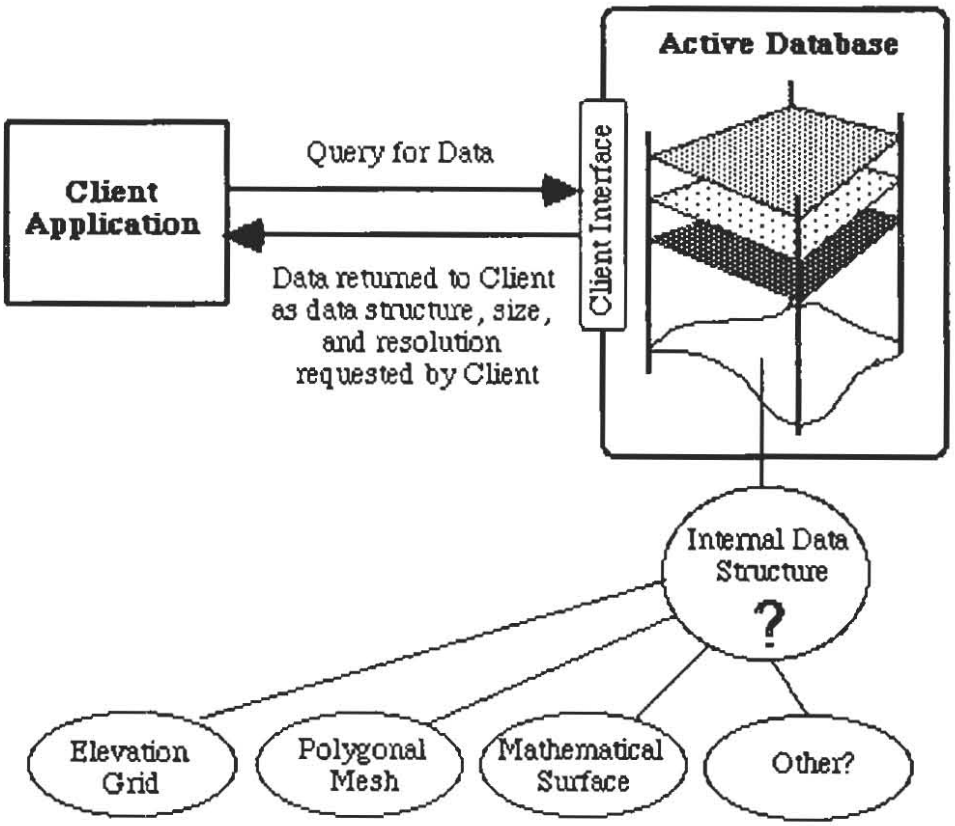

Figure 3. Decoupling the Application from the Data

The client should be free to make requests for data at any point (within the extents). Given that, the database element must be able to support queries that are arbitrarily spaced. This becomes a requirement of the underlying representation and a motivation for developing the mathematical surface representations for the terrain. For the purposes of understanding the DynamicTerrainDatabase class it is sufficient to assume that the underlying representations will return values anywhere within their extents.

Therefore, for any point within the extents of the database, there exists a vector (0..n) of information describing that point. This vector will contain elevation, as well as all other attributes specified for that particular scenario. Thinking in terms of the fundamental query mentioned above, getting fresh data from the DynamicTerrainDatabase is a sequence of queries that span some area and some number of attributes. In a reciprocal way this fundamental question also serves as the basis for the update mechanism. That is, when a change to an attribute is desired the question becomes a request to set the value of attribute attr at location x.

3.4.3 The Area Form of the Query

Practically speaking, when clients want fresh data they will most likely want it over some area. To this end, the actual query methods must be set up to handle the area form of the query as shown in Figure 4. The area form allows the client to specify a minimal amount of information and in return receive an arbitrary amount of data. From the standpoint of the DynamicTerrainDatabase class, it doesn't matter how big the area is or what the requested resolution is. The resolution could just as easily be at one hundred meter as it could be centimeter.

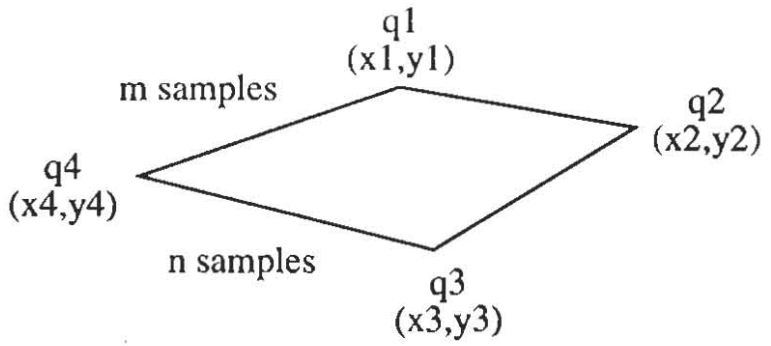

Figure 4. The Area Form of the DTDB Query

3.4.4 Advantages to Using the DTDB

Several advantages are associated with the goal of identifying a solid fundamental data abstraction in general and with the DTDB in particular.

The expressive power of the area form of the query becomes more clear when one considers that the size, shape and orientation of the query are arbitrary. Consider again the terrain following algorithm for a tracked vehicle simulation. One approach is to find the elevation and surface normal under each road wheel and use this information to determine the overall vehicle orientation. Traditionally, this operation involves an intimate knowledge of the underlying terrain data and can be adversely impacted by different underlying representations. This causes a need to reimplement the same terrain following algorithm for each representation. Using the proposed database element and its query mechanism, this terrain following algorithm is not only cleaner (and simpler to implement), but only needs to be implemented once.

This abstraction can serve as the local active database element resident on each simulator. To support dynamic manipulation of terrain attributes, the database can no longer be passive. It must become an active component of the system. Consistent with the DIS concept that each simulator maintain its own current view of the environment, it makes sense to abstract the behaviors of that active database element and then use the abstraction in each simulation node. In some sense, the local database element serves as a remote approximation for the "real" terrain being handled somewhere else.

Throughout, the DTDB is responsible for maintaining the state of the several terrain attributes. The state of terrain attributes can be considered analogous to the state of vehicles in a DIS

simulation. Consistent with DIS, we believe that occasional broadcasts of terrain state and certain types of local dead reckoning of terrain attributes can be used to reduce network bandwidth consumption.

By providing a local active database and the ability to transmit terrain state across a network, late joining players are also accommodated. Since all terrain events are stored implicitly by the terrain state within the active database, an entity joining sometime after the start of the simulation exercise need only request the latest state of those terrain attributes of interest. This would require a few updates to the local active database of the new entity for initialization. This is certainly a less complex approach than alternate methods that have been proposed in the past. Such approaches assume a chronology of terrain events or a collection of terrain objects resulting from terrain events can be transmitted to the late joining entity for construction of the current terrain database. These approaches would require significant initialization time on the part of the new player.

Active database elements for other classes of objects in the environment can be developed in a similar fashion as the DTDB. Several other classes of objects in the simulated environment (such as cultural features) might be well served with an active database element of their own, similar in form to the DTDB. Again, *the power is not in the implementation, but in the abstraction.*

3.5 Reconfigurable Client-Server Simulation Architectures

To provide a dynamic environment within a distributed interactive simulation requires mechanisms for one player to perceive or cause environment changes simultaneously and without interference by other players except were physical laws of the simulated world provide constraints. These mechanisms are embodied within the simulation architecture which consists of:

- the entity simulators which contain data representing the entity the simulator represents, the models and mechanisms to change that entity's state data, and communication mechanisms,
- the environment simulator which contains data representing the environment, the models and mechanisms to change the environment data, and communications mechanisms,
- the information transmitted between entity and environment simulators about the entity state, environment state, and events which alter those states, and
- the mechanisms which pass information between the simulators.

Examining these components of a distributed interactive simulation architecture can reveal some significant issues. First, the entity simulators and environment simulators are functionally equivalent. Each must maintain the state of an active object within the simulation. Therefore, the

environment can be viewed as another entity within the simulation. Such an approach allows for a conceptually simpler design which is easier to maintain, extend for new applications, and scale for increasing number of entities. Second, the information exchanged between the simulators is typically viewed at the same level of importance as the simulators while the information exchange mechanisms are viewed at a low level. In following an object oriented approach, this emphasis is reversed such that the exchange mechanisms are considered at the same level of the architecture design as the simulators, while the details of the information exchange are a lower level issue. These issues are further addressed below.

3.5.1 The Shared Environment Concept

Research at IST in dynamic environments has focused on a theme which can be characterized as the search for a substrate that supports dynamic, unscripted interactions within a distributed simulation. We are not only interested in higher-fidelity simulation models, but also in how those models can interact with lower fidelity models and where such interactions are reasonable. We are interested in a clean architecture that allows significant flexibility to support future research and development as well as proving useful to the community. In short, we are studying what must be done in order to support the desired functionalities of future simulation systems.

Over the course of our efforts, we have studied and experimented with several approaches to simulating certain components of the environment and with several approaches to how a potential solution might be structured [Lisle93, Moshell94, Horan93, Lisle94, Kilby94]. Other researchers have also studied this problem area [Downes-Martin91, Kamsickas93, Gehl90], as well as similar problems in other domains [Clark90, Herring93, Pratt91, HP93, Turner91].

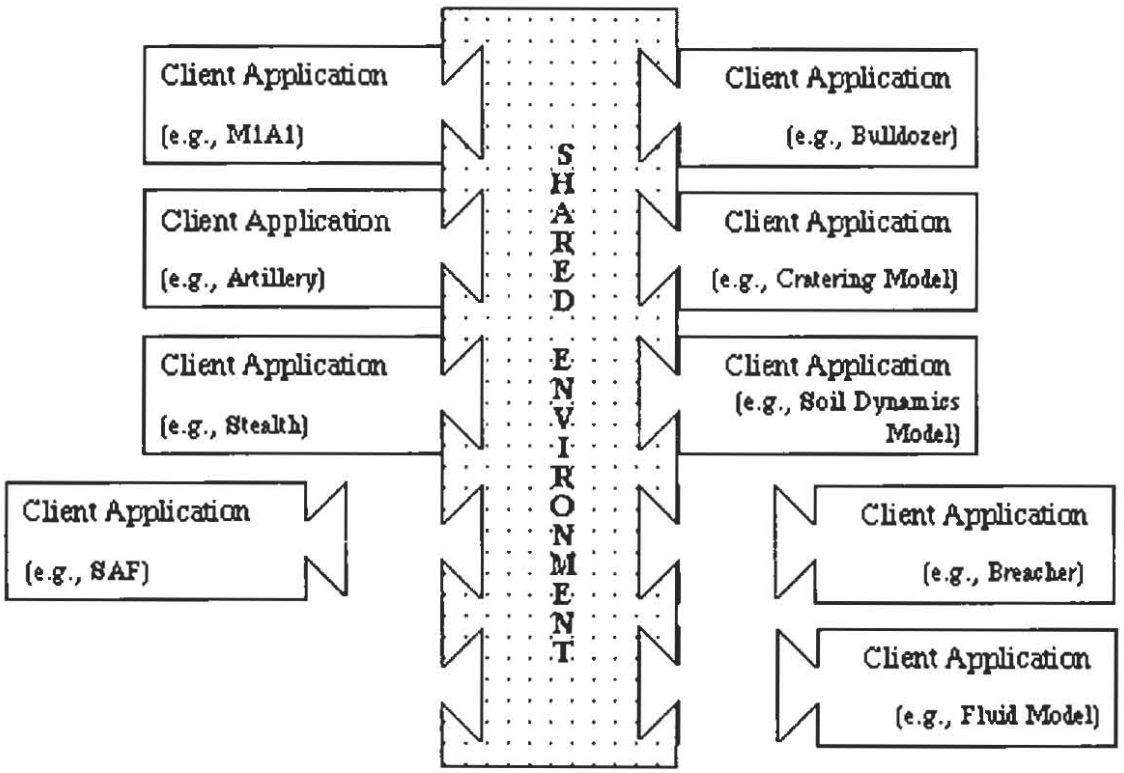

Figure 5. The Shared Environment Concept

Our research has lead us to a possible unifying semantic that we refer to as a *Shared Environment.* The Shared Environment concept represents a flexible, highly configurable, representation of the state of the environment (see Figure 5). This approach seeks to define the interface between the state of the simulated world and the entities and functionalities that make up and affect that world. Further, it begins to segregate responsibility between the components that simulate the environment and the components that simulate "normal" entities. This approach is based on client-server systems. However, this should not be confused with the Environment Manager or Server currently being debated within the DIS standard. Instead, it is better to consider the Shared Environment as a foundation upon which different designs of an environment manager can be built. In the shared environment concept, the interface of the client application (e.g., the simulators) does not change if the implementation of the Environment Manager changes.

The goals of the shared environment are that it must be scalable, vendor-independent, robust, and be able to provide an easily understood interaction between players and the environment (i.e., a clean interface), support dynamic, unscripted changes to the environment (i.e, a flexible interface), and minimize interoperability issues (i.e., a consistent interface).

The benefits of a shared environment approach include:

- A clearly defined semantic for the shared environment will allow for detailed analysis of the scalability concerns. A well-defined shared environment layer is a potential hid-

ing place for techniques that mediate the problems associated with growth to 1000 or 10,000 player scenarios. If such techniques could be devised and implemented as part of the shared environment, then *all* clients of the environment could benefit *with little or no change to the client applications.*

- Vendor-independence is an important requirement from the outset, and needs to be considered *concurrently* with other design requirements. Anyone who has been involved with making dissimilar systems work together (particularly systems from different vendors) can appreciate this.

- Combining multiple fidelity level simulations is a process plagued with difficulty. A unifying semantic and a clearly defined shared environment will alleviate much of this difficulty.

- Dynamic, unscripted changes have almost always been available to players but are virtually nonexistent for components of the environment. If recent trends are any indication, future simulation users will increasingly demand dynamic interactions with the elements of the simulated environment. The shared environment can provide this capability.

By defining simple semantics for the entire architecture, responsibilities of the components of that architecture can be clearly defined and the complexity of those components can be reduced. This again is a benefit of an object oriented approach and is a major step towards a system that is more easily maintained and flexible enough to adapt to change.

The components of the architecture lie in different conceptual layers. This layer concept allows many of the simulation components to be isolated from the details of simulation management. See Figure 6. We define three layers as follows:

- Client Application Layer: Responsible for simulating some element of the shared virtual space. This element can be either a traditional entity or the simulation of some part of the environment. Client applications typically run on a single physical machine.

- Shared Environment Layer: Responsible for providing information about the shared virtual space to the client applications, and for maintaining the consistency of the representation of the shared virtual space. Some portions of the shared environment run on each physical machine.

- Communication Layer: Responsible for implementing the consistency of representation managed by the Shared Environment Layer and includes both intramachine and intermachine forms of communication.

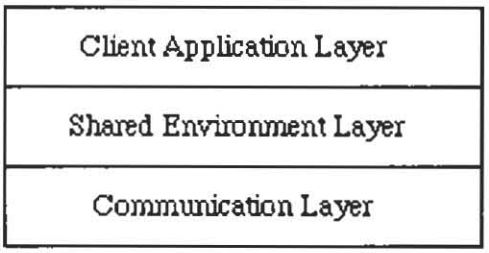

Figure 6. Layers of Abstraction in the Dynamic Terrain Database

One direct consequence of this layering is the ability to configure the environment as a central server, fully distributed, or hybrid solution with no impact on the client application programs. *The client application's notion of the "world" is fully represented by the shared environment.* The state of every element of the shared virtual space is available to the client application from the shared environment. In return, the client must only alert the shared environment of changes of state of any elements represented by that application.

More specifically, the client applications are no longer concerned with the details of communication protocols, implementations, or hardware. They communicate at a higher level by message passing with the shared environment. *We have explicitly decoupled the applications from the simulation management functionality.* It is the responsibility of the shared environment to maintain this functionality.

To illustrate the advantage of this decoupling, consider what we refer to as the *Message vs. PDU problem.* The current DIS protocol (v2.0.4) is implicitly coupled to the capabilities of ethernet. In the absence of more abstract definition, applications that use DIS tend to couple their simulations with the bits and bytes of the PDU. Hence, applications (i.e., entity simulators) are designed around the "DIS requirement" that no message exceed 1500 bytes length which is the maximum size of a broadcast packet. This requirement is an implementation issue that should not be a concern of an application that simulates some entity.

Consider the repercussions on current DIS-compatible simulators when the underlying technology changes. If the switch is made to ATM (Asynchronous Transfer Mode), for example, many simulators will have to be changed in some way. Some will propose protocol translators, which has the least impact on the system as a whole. However, real-time translators can be troublesome to use and maintain-- particularly for an evolving standard, not to mention the reduced efficiency associated with an extra level of translation. Most will directly change their applications, which increases local complexity. Without common design semantics that isolate the details of one architectural component from another, changes of this type are also likely to significantly increase the complexity of the system as a whole.

Well-defined semantics, strong abstractions, and explicit decoupling of components afford greater flexibility, easier maintenance, and cleaner transitions to new technologies and requirements. This leads to a longer life cycle both for the components and for the system as a whole.

3.5.2 An Architecture to Implement the Shared Environment Concept

At IST, an architecture which implements the shared environment concept has been constructed. This architecture, shown in Figure 7, is based upon the assumption of utility processes running in the background of all host computers in the DIS exercise. The shared environment is implemented through these utility programs, called services. From the standpoint of a client application, the entire shared environment is represented as the union of the services. Within the client/server paradigm, the services are the *servers* while the simulation application programs are the *clients.* Client applications include entity simulators as well as dynamic environment effects models, or Dynamic Terrain Resources (DTR). In this paradigm, entity simulators have the same interface to the dynamic, shared environment as a physics-based environment model. A schematic appears in Figure 8.

The Entity Service runs in its own process and serves as the intermediary between the client applications and the DIS network. There is one copy of the Entity Service per machine, and the service handles Entity State, Fire and Detonate PDUs. Each application is granted a private channel for communication with the service. This further decouples applications from each other, and allows for applications running at different frame rates to access the same service. Conceptually, this is similar to accessing the main processor in a multi-user computing environment where each user is given portions of the processor's attention (e.g., Unix). Each user "feels" as if he has the machine to himself.

We have encapsulated the functionalities of DIS communication and dead reckoning within the Entity Service. Each client application can safely assume that *the most current entity state information is available from the service.* We have also gained the capability to support a small, arbitrary number of applications on the same machine. The ripple effect is that the applications can be further decoupled from each other and each more tightly encapsulated.

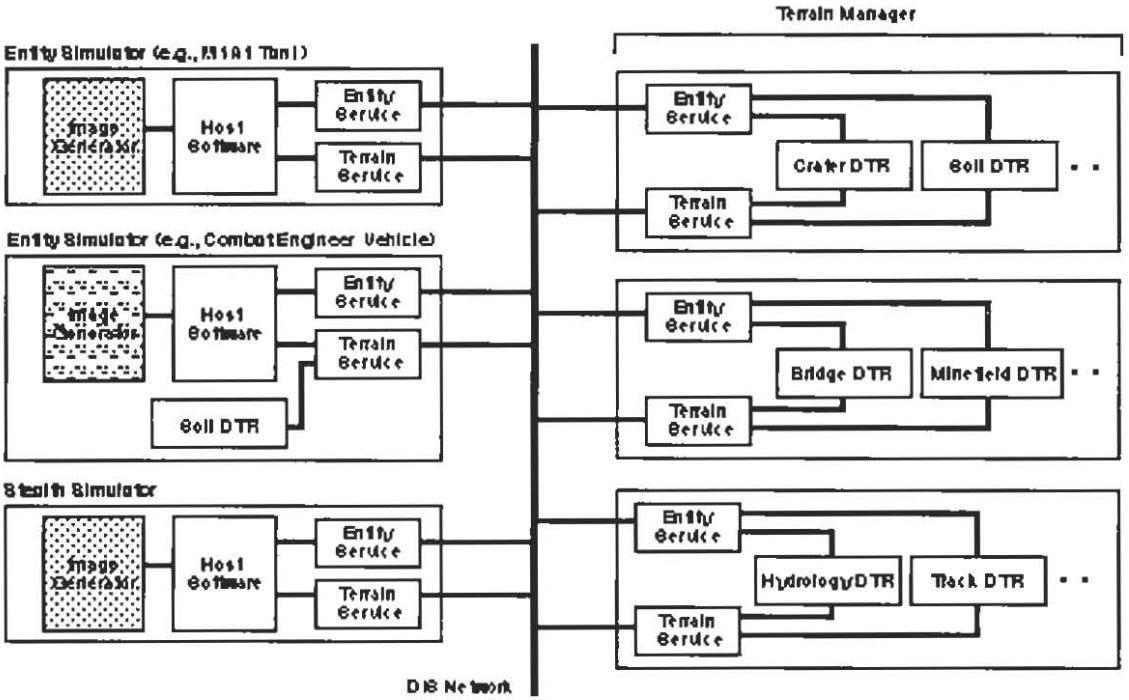

Figure 7. An Architecture to Implement the Shared Environment

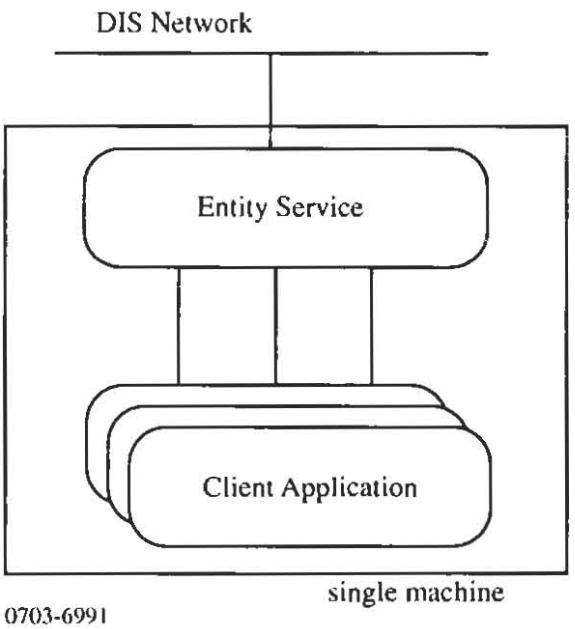

Figure 8. The Entity Service

In a fashion similar to the development of the Entity Service, a Terrain Service was created to handle the state of the terrain. The Terrain Service resides in its own process and contains an instance of the Dynamic Terrain Database (DTDB) [Lisle94]. The service acts as intermediary between the DTDB and the client applications for queries and updates, as well as handling transmission and reception of prototypical dynamic terrain PDUs.

In addition an experimental protocol was established for the Terrain Service to notify the client applications upon receipt of a terrain change. After notification, the client application must determine if it is affected by this change and needs to generate another query. An example of this interaction is the receipt of an area of terrain recently cratered. The service will notify the applications that a specific area has changed, and it is up to the application to determine whether it needs to resample or not.

Dead reckoning of terrain attributes could be made possible through the Terrain Service. This would reduce the network traffic while still allowing the applications to receive the most current state of the terrain. An example of this might be the cooling of terrain through use of a thermal attribute of the terrain database. *This is consistent with the DIS concept that a state broadcast is a limited time promise to all recipients which they can use with minor changes they perform themselves.*

3.5.3 Other Services

For purposes of prototyping approaches to hydrology and interaction with bodies of water, a Fluid Service has also been developed. In addition to the three services that we have implemented for handling entity and terrain state information we have identified other candidate services. One candidate might be called a "Culture Service". It is similar to the Terrain Service except that it handles cultural or feature information. Another candidate might be referred to as an "Object Service". It would also have a similar interface, but it would handle certain 3D objects.

3.5.4 Information Exchange (What Should be in the PDUs?)

From a system perspective, DIS Protocol Data Units (PDUs) exist to interchange the minimal information necessary to ensure that the models running on the DIS network are successfully coupled to achieve interoperability between simulators. However, the data content of inter-simulation messages (i.e., PDUs) will heavily depend on the particular mathematical models. Therefore, the environmental model executed on the terrain manager and/or on the local simulator's host computer should be considered in the PDU definition process.

In our opinion, dynamic environments PDU definitions should be consistent with that of current DIS PDUs (e.g.Entity State PDU). For instance, the Entity State PDU is an occasional state broadcast which informs all listeners of the current position and orientation of a vehicle so that they can all update their local versions of the vehicle.
Having local versions of a dynamic environment is a consistent extension to DIS. As necessary, state changes to the portions of the environment will be conveyed through the transmission of "environmental state" PDUs. The information contained in the PDUs should be whatever is appropriate for the local environmental models on each DIS node.

The PDU format should be sufficiently general so that simulators built on different data abstractions can effectively use the PDU. This allows simulators of different capability and architecture to communicate via state data exchanges. For example, at a demonstration at the 15th Interservice/Industry Training Simulation and Education Conference (I/ITSEC'93), an ESIG-2000 image generator using a polygonal mesh was interacting with an Silicon Graphics Onyx using mathematical surfaces.

4.0 Summary of Current and Future Research

Examples of the current research in Dynamic Terrain at IST is contained within the Dynamic Terrain Developers' Kit which is available through the Tactical Warfare Simulation and Technology Information Analysis Center (TWSTIAC). This packet of information has documents and software that describe the current implementation of a number of components of the Dynamic Terrain simulation architecture implemented by IST. Note that the software contained in this package is prototype code for further research in dynamic environments only and should not be considered as a commercial-grade software package. In particular, documentation and software are available for:

- Components of the Shared Environment concept including:
    - Entity, Terrain, and Fluid Services
    - The Dynamic Terrain Database (DTDB) used in the Terrain Service
    - An Abstract Service for developing new types of dynamic environment services
- Dynamic Terrain Resources - models to implement dynamic environment events
    - Soil DTR - digging soil
    - Fluid DTR - flowing water
    - Crater DTR - crater formation due to detonations
    - Track DTR - laying vehicle tracks
    - Thermal DTR - changing terrain temperature
    - Minefield DTR - simulation of a simple minefield
- More Shared Environment Clients
    - Visualizer - to view changes to DTDB during a simulation
    - IG and Simulation Host - basic components for building an entity simulator
    - ESIG Host - software to operate a modified ESIG 2000 Image Generator in the Shared Environment

- Utilities

    - DTDB Formatter - converts files of other formats into DTDB format
    - Configuration programs to initialize the Terrain and Fluid services
    - Sources and Sinks - applications to send and receive PDUs

These components of the Shared Environment concept address many of the challenges presented previously for incorporating dynamic terrain in DIS. The table below summarizes how each of the challenges are addressed by the Dynamic Terrain architecture and its components.

Challenges to DT in DIS	**How Current Architecture Answers Challenge**
Open Systems Design to Support Heterogeneous Simulators	Demonstrated with Current Software
Support Multiple Resolutions	Demonstrated with Current Software
Support Different Run-time Data Requirements	Well-suited with modifications
Support for Current Vendor Technology	Demonstrated with Current Software
Support for Future Vendor Technology	Well-suited with modifications
Sufficient Performance (Variable Data Update Rates)	Demonstrated with Current Software
Consistent Representation	Demonstrated with Current Software
Scalability	Well-suited with modifications
Late-Joining Players	Possible with modifications
Fault Tolerance	Well-suited with modifications
Flexibility and Extensibility	Demonstrated with Current Software

4.1 Areas for Future Research

Dynamic terrain is a complex subject that will require further work to develop approaches that work seamlessly in a DIS environment. Some possible areas of future research are described briefly below.

4.1.1 Simulation Management, Fault Tolerance, and Dynamic Terrain

The services developed through IST's Shared Environment concept provide an implementation layer that insulates the details of simulation management and information distribution from the simulators participating in an exercise. However, to support 1000, 10,000, or 100,000 entities

with different resolution and information exchange requirements will require further examination of the simulation management issues and how they might be supported through the services and other mechanism.

As indicated previously, the services of the Shared Environment concept are not an environment manager. Instead, they provide the basic building blocks to build various types of environment managers, as is described and demonstrated in the DT Developers Kit. Conceptually, the services can support the design of a central environment manager, a distributed environment manager, or a hybrid approach that places some of the environment manager capability within each simulator. However, more research is required to determine the optimal approach to partition responsibilities between multiple environment managers while maintaining a scalable and fault tolerant solution. This partitioning could be geographical, functional (e.g., terrain, culture, three dimensional objects), or it could be regional (e.g., land, sea, atmosphere). In each case, boundary conditions must be carefully considered so that an entity transitioning from one environment manager's area of responsibility to another's does not detect the transition. Otherwise, unrealistic behaviors result due to the artifact of simulation architecture design and the simulation effectiveness is reduced.

In considering the effectiveness of the dynamic environments solution, fault tolerance will also be a key issue in modifying the DIS architecture to support Dynamic Environments. It is possible that a single DIS exercise could potentially cost hundreds of thousands of dollars due to the investment of labor and equipment to support such an exercise, not to mention the actual participants. Thus, the environment manager, which essentially interacts with all other DIS entities, must provide a stable simulation of the entire environment. However, the issues of fault tolerance will be difficult to address until the portion of the modeling and simulation community that is developing the DIS standard agrees to some base architecture for supporting dynamic environments.

4.1.2 Real-time modifications to 3D objects

As virtual environments evolve, they will be required to support greater interaction, manipulation and modification of three dimensional (3D) representations of objects during an interactive simulation. The majority of the current research focuses only on the modeling of changes to terrain and water surfaces in an interactive simulation. Further research will be required to address these needs.

Some of this research is currently underway. For instance, IST is currently studying how the "surface modification" approach might be generalized for three dimensional objects and still meet the performance requirements of interactive simulations. This work is currently being conducted under contract N61339-94-K-0005 for the Naval Air Warfare Center - Training Systems Division. This work supports explosion effects on buildings due to the type of munitions, placement of munitions, and building construction material. Extensions to this work are also under way. Other areas that may require dynamic 3D objects would include high fidelity computer aided design, simulation of manufacturing operations, and training of medical personnel in various medical treatments.

4.1.3 Dynamic Correlation

As the need for greater realism in distributed simulations increase, the necessity for consistent representations of objects in the virtual environments will also increase. When dealing with distributed interactive simulation, the spatial and temporal correlation of object representations on different simulators in a DIS exercise is a primary concern. Specific methods of measuring the degree of correlation will be required and IST and other members of the DIS community are studying these issues. However, architecture is a significant factor affecting the degree of spatial and temporal correlation. It will become difficult to devise such metrics until a base simulation architecture is established which supports dynamic environments.

4.1.4 Integration of Dynamic Terrain with Additional Simulation Technology

DIS is tasked with supporting heterogeneous simulators and diverse simulator technologies, and dynamic environment technology must support them as well. However, each simulator requires some subset of the entire virtual environment and typically has a distinct and different representation from other simulators. The mathematical surfaces approach addresses the needs of many different simulators, particularly manned vehicle-level training simulators. However, other technologies have different representations of the world that cannot readily adapt to this approach. Thus, other data abstractions are required. For instance, computer generated forces (CGF), require logical representations of the artifacts of a dynamic environment-changing events, such as a destroyed bridge. Thus new representations must be explored and new methods for the technology to adapt to the changing virtual environment must also be pursued.

5.0 Conclusion

This document has described the challenges of Dynamic Terrain (DT) and the system level approaches to developing DT for Distributed Interactive Simulation (DIS). Incorporating DT into training simulators has been a complicated process largely because the term "dynamic terrain" means different things to different people; therefore, accommodating these differing perceptions can be difficult. The principal research questions in this project have been: how should dynamic terrain in the current simulator context be implemented, and how should knowledge gained in answering the first question be used to provide for further undetermined dynamic terrain capabilities? Meeting the requirements of interoperability, complexity, and abstraction have been important considerations in overcoming the challenges of Dynamic Terrain in DIS. The solution developed by the Visual Systems Laboratory (VSL) at the Institute for Simulation & Training (IST) of the University of Central Florida is termed "Shared Environment."

This research has resulted in the following conclusions regarding simulation architectures which support dynamic environment effects in distributed interactive simulations:

- Environment state can be considered in a manner consistent with vehicle state. Occasional broadcasts and local dead reckoning can be used to minimize consumption of network bandwidth. This is a natural extension of the DIS protocol.

- The diverse needs for Dynamic Environments in DIS exercises can be addressed by a reconfigurable simulation support architecture. The Client/Server approach is one means to achieve this and provides a consistent common interface to the simulated environment.

- The flexibility of the software and architecture supports a long software life cycle and makes prototypes easier to adapt into full-capability simulations.

- The environment data structure developed for dynamic terrain should support today's needs and allow room for growth through extensibility. Today's solutions should not be applied to tomorrow's problems.

- The issues surrounding dynamic terrain, dynamic environments, and the environmental manager for DIS are complex. The community should consider addressing this problem iteratively by accepting partial solutions to the problem and allowing the final solution to develop over time.

The results and recommendations included in this document come out of a detailed study of the problem of dynamic terrain in distributed simulations. However, many problems remain unresolved and must be addressed by further research.

References

[Booch94] Booch, Grady, "Growing Mature Abstractionists." From Practical Objects column, Object Magazine, 3(6), February 1994.

[Booch91] Booch, Grady, Object Oriented Design with Applications. The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.

[Clark90] Clark, James D., "Modelling and Simulating Complex Spatial Dynamic Systems: A Framework for Application in Environmental Analysis." Simulation Digest, Volume 21, Number 2, Winter 1990/1991, a joint publication of ACM Press, and IEEE Computer Society.

[Cortes94] Cortes, Art, and Guy Schiavone, "A Practical Parametric Approach to Resolving Interoperability." Proceedings of the 10th Workshop on Standards for the Interoperability of Defense Simulations, Orlando, Florida, March 1994.

[Couch87] Couch, Leon W., II. Digital and Analog Communication Systems. 2nd ed. MacMillan Publishing Company. 1987.

[Downes-Martin91] Downes-Martin, Stephen, "Seamless Simulation Literature Survey." Institute for Defense Analyses, IDA Document D-1109, October 1991.

[Entsminger90] Entsminger, Gary, The Tao of Objects. M&T Books, Redwood City, California, 1990.

[Fullmer90] Fullmer, Charles, Pamela Woodard, and Ron Matusof, "Interoperability: The Key to Successful Team Training and Rehearsal." Proceedings of the 12th Interservice / Industry Training Systems Conference, Orlando, Florida, November 1990.

[Gehl90] Gehl, Thomas, and Joseph J. Brann, "Network Requirements for Distributed Tactical Training." Proceedings of the 12th Interservice / Industry Training Systems Conference, Orlando, Florida, November 1990.

[Herring93] Herring, Charles, Biju Kalathil, and Joseph Teo, "Research in Persistent Simulation: Development of the Persistent ModSim Object-Oriented Programming Language." US Army Construction Engineering Research Laboratories, USACERL Interim Report FF-93/07, July 1993.

[Horan93] Horan, William H., Michael J. Smith, Curtis Lisle, and Marty Altman, "An Object-Oriented Environmental Server for DIS." 9th Workshop on Standards for the Interoperability of Defense Simulations, Orlando, Florida, September 1993.

[HP93] Hewlett-Packard, "HP Distributed Smalltalk Technical Information." Hewlett-Packard Company, Distributed Computing Program, 1993.

[IST93a] DIS Operational Concept 2.3, IST-93-25, Institute for Simulation and Training,

Orlando, FL.

[IST93b] Correlation Studies - Final Report, IST-93-17, Institute for Simulation and Training, Orlando, FL.

[IST94] The DIS Vision - A Map to the Future of Distributed Simulation, IST-SP-94-01, Institute for Simulation and Training, Orlando, FL.

[Kamsickas93] Kamsickas, Gary M., "Distributed Simulation: Does Simulation Interoperability Need an Environment Server?" Proceedings of the 15th Interservice / Industry Training Systems and Education Conference, Orlando, Florida, December 1993.

[Kilby94] Kilby, Mark, Curtis Lisle, Marty Altman, and Michelle Sartor, "Dynamic Environment Simulation with DIS Technology." Proceedings of the 16th Interservice/Industry Training Systems and Education Conference, Orlando, Florida, November 1994.

[Knight90] Knight, S.K., "Issues Affecting the Networking of Existing and Multifidelity Simulations." 2nd Workshop on Standards for the Interoperability of Defense Simulations, Kissimee, Florida, January 1990.

[Lisle93] Lisle, Curtis, and J. Michael Moshell, "Object-Oriented Physical Modelling." Journal of Systems Engineering (1993)3:191-201, Springer-Verlag, 1993.

[Lisle94] Lisle, Curtis, Marty Altman, Mark Kilby, and Michelle Sartor, "Architectures for Dynamic Terrain and Dynamic Environments in Distributed Interactive Simulation." 10th Workshop on Standards for the Interoperability of Defense Simulations, Orlando, Florida, March 1994.

[Mamaghani94] Mamaghani, Farid, "What Makes Virtual Systems a Reality." From SIGGRAPH '94 Course Notes, Digital Illusion: Distributed Interactive Entertainment, Orlando, Florida, 1994.

[Meyer88] Meyer, Bertrand, Object-oriented Software Construction. Prentice-Hall, New York, New York, 1988.

[Miller56] Miller, G., "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information." The Psychological Review, Volume 63 (2), March 1956.

[Moshell94] Moshell, J. Michael, Brian Blau, Xin Li, and Curtis Lisle, "Dynamic Terrain." Simulation 62:1, 29-40, Simulation Councils, Inc., 1994

[Mullin89] Mullin, Mark, Object Oriented Program Design, with Examples in C++. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[Nelson91] Nelson, Michael L., "An Object-Oriented Tower of Babel." OOPS Messenger, Volume 2, Number 3, July 1991, ACM Press.

[Pratt91] Pratt, David B., Phillip A. Farrington, Chuda B. Basnet, Hemant C. Bhuskute, Manjunath Kamath, and Joe H. Mize, "A Framework for Highly Reusable Simulation Modelling: Separating Physical, Information, and Control Elements." Proceedings of the 24th Annual Simulation Symposium, New Orleans, Louisiana, April 1991.

[Riecken93] Riecken, Mark, and Sheila O'Brien, "DIS Synthetic Environment Interoperability Issues: Level 1, 2, and 3." 9th Workshop on Standards for the Interoperability of Defense Simulations, Orlando, Florida, September 1993.

[Rolfe86] Rolfe, J.M. and Staples, K.J. Flight Simulation. Cambridge University Press. 1986.

[Rumbaugh91] Rumbaugh, James, Michael Blaha, William Premerlani, Fredrick Eddy, and William Lorenson, Object-Oriented Modeling and Design. Prentice Hall, 1991.

[Sieverding94] Sieverding, Michael J., "DoD Training System Digital Data Base Requirements, A Management Tutorial." Proceedings of the 1994 Image VII Conference, Tuscon, Arizona, June 1994.

[Simon81] Simon, Herbert A., The Sciences of the Artificial. Second Edition, The MIT Press, Cambridge, Massachusetts, 1981.

[Turner91] Turner, Stephen J., "Towards Portable and Transparent Parallel Simulation Protocols." Proceedings of SIMTEC '91, International Simulation Technology Conference, Orlando, Florida, October 1991.

[Woodard94] Woodard, Pamela S., "Requirements for Interoperable Environments." Proceedings of the 1994 Image VII Conference, Tuscon, Arizona, June 1994.