DYNAMIC ENVIRONMENT SIMULATION WITH DIS TECHNOLOGY

Mark Kilby, Curtis Lisle, Martin Altman, and Michelle Sartor

University of Central Florida Institute for Simulation and Training,

Orlando, FL

ABSTRACT

During the latest DIS workshops, the addition of a Terrain Manager or Environment Manager to DIS brought out understandable differences of opinion. Options discussed have covered the responsibilities of the Terrain and/or Environment Manager, its effect on the entities, and how entities keep track of the changing environment while considering whether any fundamental goals of the DIS paradigm such as "No *central computer for event scheduling or conflict resolution" are violated. However, providing a consistent* and dynamic environment in DIS exercises requires more than single environment management module, whether it be per network or per node. Instead, modifications to the simulation support architecture as a whole, must be contemplated.

Issues such as network bandwidth, CPU performance, and scalability must be considered by a system architecture that supports dynamic environments in a distributed interactive simulation. To address this need, several architectures are presented which could support dynamic entity/environment interaction. *As each architecture is discussed, results and measurements made from prototype software are* presented to point out strengths and weaknesses. IST demonstrated networked Dynamic Terrain (DT) capability using the most recent architecture at the I/ITSEC'93 conference. The architecture supported changes to the terrain profile, an extended DIS protocol, and provided a consistent way to manage changes made by entities.

ABOUT THE AUTHORS

Mark Kilby is a Visual Systems Scientist and principal investigator of IST's Dynamic Terrain project. Mr. Kilby received his BSEE and MSEE from the University of Florida

Curt Lisle is a Visual Systems Scientist at IST and analysis team leader on the Dynamic Terrain project. He received his BSEE from Georgia Institute of Technology and MS in Computer Science from the University of Central Florida (UCF). Also, Mr. Lisle is currently a Computer Science Ph.D. student at UCF.

Martin Altman is a Visual Systems Scientist at IST and architecture team leader on the Dynamic Terrain project. He received a BS in Computer Science from UCF, Summa Cum Laude.

Michelle Sartor is a Visual Systems Scientist at IST and real-time modeling team leader on the Dynamic Terrain project. She received a BS in Biology from Florida State University and a BSEE and MSEE from UCF.

INTRODUCTION

The purpose of a simulation support architecture which provides a Dynamic Environment within a Distributed Interactive Simulation (DIS) exercise is to enhance the effectiveness of the exercise. This enhancement is achieved by allowing entity simulators to affect and be affected by changes in the simulated environment. To provide this enhancement to the current DIS paradigm, several issues must be examined.

This paper presents issues that should be studied when defining a system-level architecture for Dynamic Environments in the DIS paradigm. First, concerns of the DIS community with respect to Dynamic Environments are discussed. Second, the objectives and assumptions for providing a Dynamic Environment within the DIS paradigm are presented. To meet these objectives and substantiate these assumptions, our research has explored a number of architectural options for Dynamic Environments within the DIS paradigm. These architectural prototypes and analytical results are presented next. The most recent prototype was demonstrated at the 1993 I/ITSEC conference. Lessons learned from this implementation are discussed. Finally, conclusions on results to date are presented.

Dynamic Environment Issues in the DIS Community

The past several DIS workshops have focused on two main issues concerning Dynamic Environments: a central environmental server and definition of the PDUs to transmit environmental changes during a DIS exercise.

Central Server or No Central Server. The concept of a Terrain Manager or Environment Manager in the DIS paradigm has generated understandable differences of opinion. The discussions in previous DIS workshops have covered the possible responsibilities of the Terrain and/or Environment Manager , its effect on the entities, and how entities keep track of the changing environment -- all the while considering whether this violates any fundamental goals of the DIS paradigm such as "No central computer for event scheduling or conflict resolution" (IST, 1993a).

Most of these discussions have focused on a single architecture which solves all the needs of the DIS community for a dynamic environment. In turn, much of the debate on this anticipated single architecture has centered around whether a central server is a necessary or desirable foundation of the architecture. A central server will have problems with throughput as the scenario scales to a larger number of entities. However, distributed architectures have problems with data redundancy, latency, and loose coupling between the CPUs on different simulators. The choice of options can not be resolved until dynamic environment requirements for a variety of possible applications in DIS are developed and prototypes to explore and meet these requirements are constructed.

Environment PDU Definitions. Much of the discussion on PDUs has assumed that PDUs will be sufficient to maintain temporal and spatial consistency between the diverse number of possible players in a DIS exercise and for the wide range of applications proposed for DIS (e.g., training, mission planning and rehearsal, doctrine development, virtual prototyping) (IST, 1993a). However, the supporting architecture must be considered as well. Furthermore, these PDUs need to support a variety of data requirements, both current and future. Each environmental PDU designed must consider the data requirements of the simulation platforms, the various environmental models, and the variety of applications proposed for DIS. Otherwise, unnecessary PDUs may be developed that add to the overall complexity and maintainability of DIS-compliant simulator hardware and software.

Objectives of a Dynamic Environment Architecture In DIS

To address these issues, objectives must be set for evaluating solutions to provide a dynamic, interactive environment for DIS exercises.

(O1) Sufficient Performance. Because the architecture must support several different types of simulations, it must support changes to the environment and provide these updates to all DIS nodes at a rate at least equivalent to the update rate of the fastest simulator in a particular DIS exercise. 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 (IST, 1993). Also, performance is an important criteria, but not the most important. Focusing only on performance will deliver a short term solution

which precludes other equally important objectives (listed below). These additional objectives could provide for the longevity and widespread use of the DIS standard.

(O2) Consistency of Representation. The 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 entities. The intent is to reduce correlation problems.

(O3) Scalability. One of the goals of the DIS standard is to support forces of varying sizes. Therefore, architectural solutions should work equally well for 10 or 10,000 entities. 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.

(O4) Flexibility and Extendibility. Architectural solutions to provide a dynamic, interactive environment requires 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 (IST, 1993a). 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.

(O5) Sufficient Realism. Different exercises as well as individual simulators within an exercise will require different levels of spatial and temporal fidelity. The architecture should support these multiple levels of fidelity at any resolution, size, or orientation.

Assumptions

To support the objectives listed previously, assumptions have been made in our research which have been derived through a number of sources (DIS, 1993, Lisle, et.al. 1994, E2DIS, 1993). Our research focuses on dynamic terrain, yet these assumptions are equally applicable to a complete dynamic environment.

(A1) Reconfigurability. It is unlikely that a single "golden" architecture will support all current and future DIS applications. Other authors have pointed out that the types of environment information required for a DIS exercise will vary depending upon the types of entities that are interacting and the goals of the exercise (Kamsickas, 1993).

This makes the choice between a central versus a distributed server unclear. A central server provides a consistent representation and reduces the processing power required of the individual nodes. However, the central server proves unscalable for a large number of entities and is not inherently robust (i.e., a single point of failure). A distributed server proves more scalable but comes at the cost of data redundancy, latency, and correlation complexity.

A hybrid system may be the best approach (Kamsickas, 1993). A single "golden architecture" which will solve all DIS dynamic environment requirements may be unfeasible, especially since there is a great variety of intended uses for DIS. Instead, different architectures for different applications may address the problem more effectively. For example, the central server will be cost-effective if it provides enough performance for a particular application having only a few entities. It may also prove to be the best solution for local collections of legacy simulators which cannot be easily modified. A fully-distributed system would be best for an application of DIS over a Wide-Area Network with low bandwidth in between DIS cells. A hybrid between centralized and distributed systems is an effective compromise for many applications. To support a variety of disparate needs with a single system and to address objectives O3 and O4, the system's architecture should be reconfigurable.

(A2) Unscripted changes to the Terrain Maintained Through an Active Database. Instead of maintaining a list of environment changing events, all changes will

be incorporated into an active terrain database. Some image generator vendors are currently considering active databases as a modification to their current designs (Rich, 1992).

(A3) Applications Decoupled from Data Structures. To support objectives O3 and O4, the entity simulators should not depend on the structure of the data used by the simulation architecture to transmit and maintain a Dynamic Environment. These simulations should only depend on the content of the data. This can be provided through a standard interface, or query mechanism.

(A4) Arbitrary Number of Terrain Attributes. Typically, elevation and slope are the most frequently used attributes of the terrain. However, future applications will require such attributes as temperature, soil strength, and moisture content. Furthermore, different entity simulators need different attributes. For instance, a flight simulator with infrared sensors may only require terrain elevation, slope, and thermal attributes. However, a tank simulator would require additional terrain attributes to compute effects such as mobility.

(A5) Dynamic Update of Environment State. Just as Entity State PDUs transmit the change of an entity's state, terrain state PDUs should convey the change in the state of the environment.

(A6) Open System Design. Solutions to providing the Dynamic Environment within a DIS exercise should not be biased toward one vendor (e.g., a particular polygonization scheme).

(A7) Consistency with current vendor technology.

(A8) Support for Verification and Validation. The simulation architecture must provide a means to validate models and simulations in DIS exercises with minimal intrusion.

(A9) Support for Updates to Late Joining Entities. Players joining after the start of an exercise must be able to rapidly update their environmental database to match the current simulated environment.

(A10) Hierarchical Configurations. Certain exercises may require areas of terrain at different resolutions. Hierarchical approaches are well known solutions to this problem.

(A11) Geographic Segmentation. Allow for separate Terrain Managers to be responsible for geographically unique portions of the gaming area during an exercise.

(A12) 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.

(A13) Support Entity Simulator Queries for Environment Data both Locally and Remotely. Simulators should be provided with environmental data when required and not when it can be supplied by the architecture.

(A14) Assume and Relinquish Control of Steady State Objects. The Dynamic Environment support architecture could assume and relinquish control of steady-state objects (such as destroyed vehicles, bridges, buildings). It will also be capable of removing objects from the simulation as appropriate.[1]

(A15) Large Scale Scripted Environmental State Changes. The architecture should include the ability to replay scripted large scale environmental state changes to support simulator preprocessing of predistributed data. This will aid physically-accurate sensor simulations which currently require preprocessing to allow real-time performance. These predistributed databases can be replaced with models as they become available.

EVOLUTION OF A SOFTWARE ARCHITECTURE TO SUPPORT DYNAMIC TERRAIN

In seeking out architectures that will satisfy the objectives and qualify the assumptions listed previously, our research has explored a number of architecture options for providing a Dynamic Environment to DIS exercises. The variations of the architecture and analytical results are discussed below after a review of some initial design decisions.

[1]This issue is still under debate within the DIS Simulated Environment Working Group with some members being opposed to this assumption (Kamsickas, 1993).

Initial Design Decisions

To be consistent with current CIG technology (see A7), several design decisions were made.

2 1/2 D Representation. The portion of the terrain that is usually most relevant to a given scenario is the surface or "skin". This is sometimes referred to as a 2 1/2 dimensional representation.

No Multi-Valued Areas. An artifact of the 2 1/2 D representation, there is no concept of multi-valued areas (locations having more than one elevation value). Thus there is no generalized mechanism for representing caves or tunnels. Such things, if modeled, are considered as features or are otherwise handled as special cases.

Support Polygonal Based Representations. Most image generators are polygon based. In this representation, the terrain is essentially a large polyhedron. While research should explore alternative representations, we cannot ignore systems that are polygon based. It is also important to note that the different polygonization schemes have traditionally been a source of problems when linking heterogeneous simulators.

Gridded Source Data. The elevation source data for any given piece of terrain often comes in a gridded format where for each location there is one associated elevation value.

New Concepts in Environment Data Representation

To satisfy several objectives and assumptions (O2 - O5, A1 - A7) research was conducted in determining new data structures and data access methods for an environment that could support current polygonal-based training simulations as well as future applications. First, mathematical surfaces were determined to be the optimal choice in obtaining resolution independence. A terrain accuracy experiment using the 1992 I/ITSEC source database for the interoperability demonstration indicated that these data representations could represent the same terrain with fewer data points than equivalent gridded representations from which polygonal databases are currently constructed. Also, these surfaces were not subject to the errors of resampling at different orientations as are gridded representations (Lisle, et. al., 1994). Such representations may even produce very efficient environmental PDU designs.

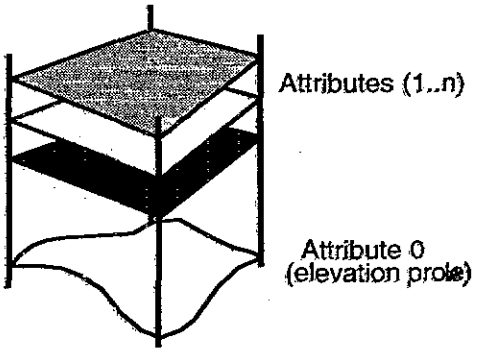

Figure 1: The Dynamic Terrain Database

To simplify the use of this new data structure by different simulators, a standard conceptual model was also developed. This conceptual model is referred to as the Dynamic Terrain Database (DTDB) (see Figure 1). Different attributes, including the terrain surface, are represented as layers within the database with each layer being represented by a mathematical surface. To access the data of an attribute layer, a standard interface to the database is provided in the form of an arbitrary quadrilateral area. That is, a simulator can request data in the form of gridded area of any size, resolution, or orientation. From this gridded data, a polygonized terrain surface can be easily generated.

Architecture 1 - Central Server

This first architecture (see Figure 2) was designed to serve two main purposes. First, the initial design was to address the fundamental problems in Dynamic Terrain (see O1 - O5 and A2 - A7). Second, the design should help determine how DIS could benefit from existing academic research.

Definitions. The following terms are used in describing Architecture 1:

Dynamic Terrain Portal (DTP): The DTP is the only part of the system with a connection to both the DIS network and the internal Dynamic Terrain architecture. It serves as the interface to and from the DIS network. Its responsibilities can be grouped into three main categories. First, inbound task responsibilities include watching the DIS network for terrain changing events (e.g. bulldozer digging, detonations on terrain).

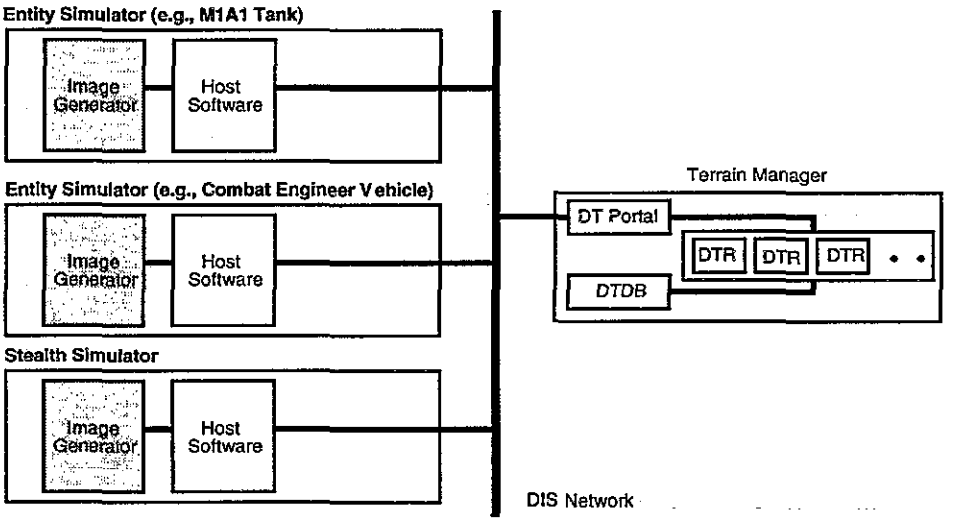

Figure 2: Architecture 1

Second, internal task responsibilities include handling details of the DT Simulation (e.g. Remote Entity Approximations, interaction between the DTDB and DTRs). Finally, outbound task responsibilities include transmitting terrain changes to simulators via new DIS messages.

Dynamic Terrain Resource (DTR): A specific physics-based algorithm is contained inside each DTR. The DT architecture allows for DTRs to be added as they are designed.

Dynamic Terrain Database (DTDB): This object maintains the simulation's environment database. Ground elevations and terrain attributes are stored here with any information necessary for the function of the DTRs.

Analytical Results. A software simulation of this system was developed to evaluate system behavior against DIS PDU streams. The following is a brief summary of the lessons learned from this experience:

- Data transfer traffic between DTRs, the portal, and the database was heavy. These functions should be on the same CPU or a shared memory multiprocessor.

- The design of the database was more critical to overall system performance than was originally expected, particularly in the area of record-locking and its affects on timing of multiple jobs simultaneously processed by the Terrain Manager.

- The DTP became a bottleneck since it is the only point of connection between the DIS network and the DTRs.

- This architecture was not sufficiently scalable because of access conflicts to the single, shared Database resource and the traffic load through the DT Portal.

Summary. It was understood before the development of Architecture 1, that this design did not completely fulfill the objectives presented earlier. However, it was conceived as a step in the process of requirement definition and provided useful experimental results on the central server approach.

Architecture 2 - Distributed Server

Architecture 2 (see Figure 3) is based on Architecture 1 with more detail focused on the Terrain Manager implementation (Horan et. al., 1993). Development began with several goals:

- Develop details of the Terrain Manager design. Look for efficiency increases and any major problems of the original architecture design (e.g., the DTP bottleneck).

- Expand the parallelism of the Terrain Manager. Look at the component responsibilities and how they could be allocated among a varying number of parallel processors.

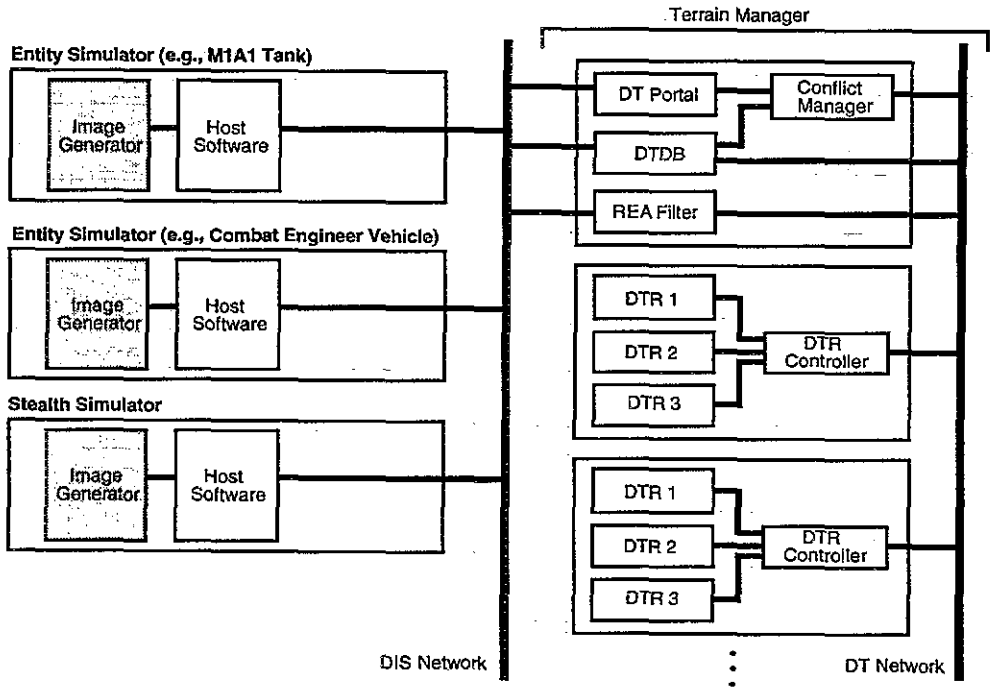

Figure 3: Architecture 2

• Support dynamic load balancing among multiple processors to consolidate all physical modeling jobs working on a specific geographic area to one processor. This permits sharing of a local database cache by the DTRs.

Additional components were defined in Architecture 2 to improve functionality. First, the functionality of the DT Portal (DTP) was split between the DTP, the Conflict Manager, the Remote Entity Approximation (REA) Filter, and the DTR Controllers. In this new design, the DTP monitors the DIS PDU stream for environment (i.e., terrain) changing events. When one is detected, the DTP determines the physical model invoked, assigns a priority to this task, then hands the task over to the Conflict Manager. The Conflict Manager, in turn, determines the machine on which to run the physical model based on available processing resources. The Conflict Manager hands the tasks to a specific DTR Controller, a component which controls all DTRs on a specific machine. The Conflict Manager also receives changes to the DTDB from a DTR and resolves conflicts when two or more DTRs wish to modify the same location in the database. Once the DTR Controller receives a task, the task is then passed to a DTR to run the physical model and compute the change to the environment. Also, the REA filter is used by the DTRs to monitor an environment-changing event once it has been identified by the DTP. The DTDB, while similar to the version in Architecture 1, now has access to the network for directly sending out changes to the database via PDUs. Thus, the communications responsibilities of the DTP are spread across the DTP, REA Filter, and the DTDB and reduces the communications bottleneck problem.

Analytical Results. A software simulation of this architecture provided significant lessons. Since the DTP was simplified, its main purpose became management of the processing jobs within the Terrain Manager. However, even with a dedicated dead-reckoning input path (i.e., the REA Filter), network I/O was still a limiting factor. With this design, the limit was not between the Terrain Manager and the DIS network, but within the manager itself. The DTDB serialized transactions from multiple DTRs by queueing requests. Thus, the parallel processors and the heuristics developed to locate and migrate jobs according to the CPU assignment of the job and the CPU load were not fully exercised.

Summary. This architecture was more scalable than the first. However, the limitations of a single database showed up more clearly. This architecture also proved more fault tolerant

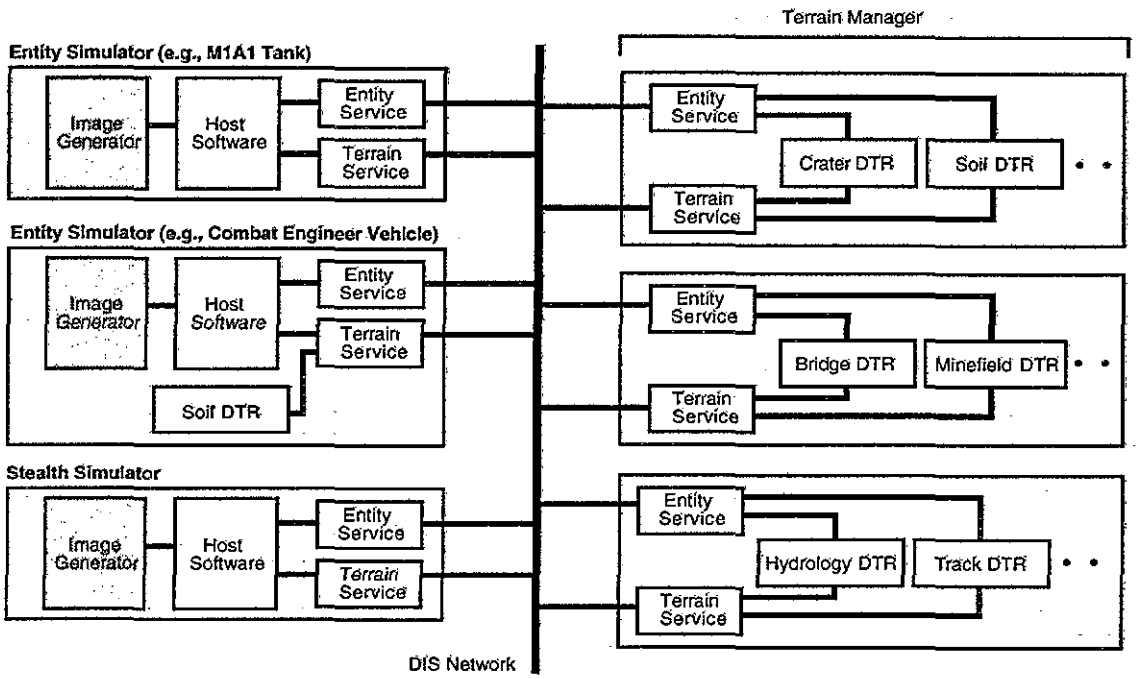

Figure 4: Architecture 3

(Horan, 1993). Failures of any of the DTR Controllers or DTRs were easy to recover. However the DTP and the REA Filter still required redundancy if the system was to remain tolerant of hardware failures.

Architecture #3 - Polymorphic Simulation Architecture

At this point in the project, several useful system designs had been developed, but a system design was still required which met the original objectives. A different approach was built based on the assumption of utility processes running in the background of all simulation host computers in the DIS exercise. The resulting shared environment was implemented through the interaction of these utility programs, called services. Using a Client/Server paradigm, the services are the servers while the simulation application programs are the clients. The idea is for the services to provide a common interface to the dynamic, shared environment for both entity simulation platforms that cause environmental changes and physics-based environment models which determine those environment changes (see Figure 4). The services make up the simulation support architecture which maintains a consistent environmental representation and isolates the environmental models and entity simulators from effects due to future design changes in this architecture.

To determine the form of this common interface, common data requirements of entity simulations and environmental models were determined. With a focus on Dynamic Terrain, these clients require constant updates on two types of data: terrain and entities in the DIS exercise. Thus, two types of services were developed. One is referred to as the Entity Service and the second is the Terrain Service.

Entity Service. One of the requirements of the DIS environment is that each node maintains representations of other entities (ghosts, or remote entity approximations) within its area of interest. Entity State PDUs are received from the DIS network and dead reckoning is performed on these approximations. In an environment where there is only one application per node (i.e., physical machine), it matters little where this functionality resides. On the other hand, if there were the capability to have several applications residing on one node (e.g., entity simulators and environmental models), the location of this functionality becomes important.

Consider the capability of several applications resident on one node. Many of the components of the DT architecture (e.g., DTP and DTRs) are individually small in resource processing requirements and do not warrant allocation of an entire machine. This leads to two approaches.

These small components can be bundled into one large, eclectic application or they could reside in separate applications. The issue then becomes one of where to place the entity tracking capability.

If grouped in one large application, the entity tracking functionality is trivially included with the DT architecture components as modules within the single application. Yet, both development and maintenance are unnecessarily complicated. If the components are isolated in several small applications on the same machine, the development and maintenance problems are reduced. Yet, location of the entity tracking functionality becomes a problem. Duplicating this functionality in each application produces inefficiency and consistency problems.

This reasoning results in a need for a single application that can perform DIS communication, dead reckon remote entities, and support simultaneous client applications that require subsets of this entity information, all running at different frame rates. This application is referred to as the Entity Service (see Figure 5).

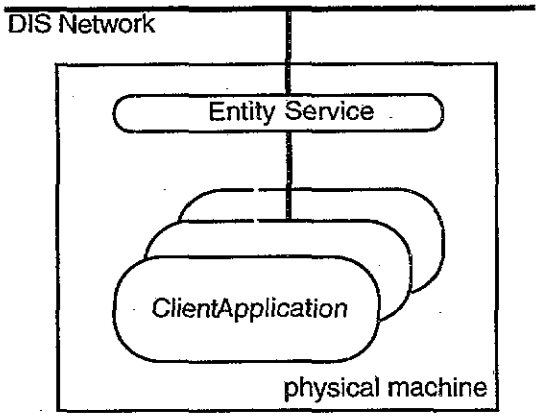

Figure 5: Entity Service

The Entity Service runs as a separate application and serves as the intermediary between the client applications and the DIS network. There is one copy of the Entity Service per machine, which handles Entity State, Fire and Detonation PDUs. Each application is granted a private channel for communication with the service. This decouples applications from one another, and allows for applications running at different frame rates to access the same service.

The functionalities of DIS communication and dead reckoning are encapsulated within the Entity Service. Each client application can safely assume that the most current entity state information is available from the service. The capability to support a small, arbitrary number of applications on the same machine is also gained. This reduces development and maintenance effort of the entire DIS simulation architecture by minimizing the "ripple effect" of design changes in one part of the architecture affecting other parts.

Terrain Service. In a fashion similar to the development of the Entity Service, a Terrain Service has been created to provide a consistent, common interface to the state of the terrain. The Terrain Service is also a separate application and contains the Dynamic Terrain Database described previously. It serves as intermediary between the DTDB and the client applications for queries and updates, as well as handling transmission and reception of environmental (i.e., 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 request the change from the Terrain Service (see A13).

Analytical Results. Currently, this architecture is under study and analytical results are not available at this time. However, this architecture was demonstrated at the 15th Interservice / Industry Training Systems and Education Conference 1993 (I/ITSEC'93). The demonstration showed *unscripted* changes to the gaming area database due to detonation of munitions and digging of anti-tank ditches and berms. Demonstration hardware consisted of several Silicon Graphics (SGI) workstations and an ESIG-2000 computer image generator on a local DIS network. SGI Indigo workstations operated various components of the Dynamic Terrain Architecture as well as simulators for bulldozers, an AVLB, and artillery rounds. An SGI Onyx and the ESIG-2000 both functioned as stealth platforms capable of simultaneously viewing all Dynamic Terrain changes occurring during the DIS simulation. The Onyx used IST-developed visualization software. Additional assistance was provided by Loral Advanced Distributed Simulation. Providing a modified GT120 image generator and using IST's dynamic soil model DTR, Loral demonstrated how this technology might be used to change SIMNET terrain databases in real-time. This demonstration achieved our project goal of exploring a means of integrating new DT

technology into both current visual systems and future simulation systems (see O4, A6, and A7).

Summary. Central server, fully distributed, and hybrid configurations can be realized by Architecture 3. More research is required to determine which configuration should be used for a particular exercise. It is important to note that client applications are insulated from changes to the configuration. This benefit is gained by separating the terrain data structures from the client applications (i.e., entity simulators). Also, the software implementations of physical modeling algorithms were easier to develop using the services. Next, since access to environment state goes through an extra level of software indirection, flexibility is achieved at the cost of some performance. However, this performance can be regained by optimization. Finally, load balancing or associated database conflict resolution management has not been implemented at the time of this writing. We expect the flexibility provided by the Terrain Service to make the architecture tolerant of changes in this area.

SUMMARY

This paper has addressed several issues that must be considered when addressing a system-level architecture for Dynamic Environments in DIS. These issues are as follows:

- 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 results and recommendations included in this paper evolved from a detailed study of the problem of Dynamic Terrain in DIS. However, many problems remain unsolved and should be addressed by continued research.

ACKNOWLEDGMENTS

This work is funded by the US Army Simulation Training and Instrumentation Command (STRICOM) under contract N61339-92-K-0001.

REFERENCES

E2DIS (1993). E2DIS Architecture Task meeting minutes. August 1993 and November 1993.

Horan W., Smith M., Altman M., Lisle C. (1993), "An Object-Oriented Environmental Server for DIS", in Proceeding of the 9th Workshop on Standards for the Interoperability of Defense Simulations, IST-CR-93-39.2. September 1993. Institute for Simulation and Training, Orlando, FL. pp. 15-23.

Horan, W. (1993), "Scheduling in a Distributed Application to Support Environmental Servers for DIS (Distributed Interactive Simulation) Exercises", Thesis Report, College of Engineering, University of Central Florida, December 1993.

IST (1993a). DIS Operational Concept 2.3, IST-93-25, Institute for Simulation and Training, Orlando, FL.

IST (1993b). Land Subgroup meeting minutes, in Proceeding of the 9th Workshop on Standards for the Interoperability of Defense Simulations, IST-CR-93-39.2. September 1993. Institute for Simulation and Training, Orlando, FL. pp. 43-86.

Kamsickas, G. M. (1993) "Distributed Simulation: Does Simulation Interoperability Need an Environment Server?", in Proceedings of the 15th Interservice/Industry Training Simulation and Education Conference (I/ITSEC). November 1993. Orlando, FL. pp. 235-244.

Lisle, C., Altman, M., Kilby, M., Sartor, M. (1994) "Architectures for Dynamic Terrain and Dynamic Environments in Distributed Interactive Simulation", in Proceeding of the 10th Workshop on Standards for the Interoperability of Defense Simulations. March 1994. Institute for Simulation and Training, Orlando, FL.

Rich, H. H. (1992). "The Active Database - Using Software to Save CIG Hardware", in Proceedings of the 14th Interservice/Industry Training Simulation and Education Conference (I/ITSEC). November 1992. Orlando, FL. pp. 858-866.