1-1-1994

Architectures For Dynamic Terrain And Dynamic Environments In Distributed Interactive Simulation

Curtis Lisle

INSTITUTE FOR SIMULATION & TRAINING

FINAL REPORT: Mobility Across Dynamic Terrain - An Engineering Change Proposal to the Dynamic Terrain Testbed Project

Contract Number N61339-92-K-1 (P00001)
IST-TR-94-20

University of Central Florida

ARCHITECTURES FOR DYNAMIC TERRAIN AND DYNAMIC ENVIRONMENTS IN DISTRIBUTED INTERACTIVE SIMULATION

Curtis Lisle, Marty Altman, Mark Kilby, Michelle Sartor
Institute for Simulation and Training, The University of Central Florida

Keywords: Dynamic Terrain, DIS, Architecture, PDU

1.0 Abstract

This paper presents issues that should be studied when defining a system-level architecture for dynamic environments in the DIS paradigm.To address these issues, several architectures which can support dynamic entity/environment interaction not currently available on today's simulators are presented. As each architecture is discussed, results derived from architecture simulations and measurements taken from prototype software are used to draw conclusions about the strengths and weaknesses of the architecture.

IST demonstrated networked Dynamic Terrain (DT) capability using the latest version of the architecture at the I/ITSEC'93 conference. Lessons learned from the development and performance of this early prototype are presented.

2.0 Too Many Assumptions, Too Little Data?

2.1 "To Serve or not to Serve, That is the Question..."

Over the past several DIS workshops as discussions have begun over the addition of a Terrain Manager or Environment Manager to DIS, there have been understandable differences of opinion. The discussions have covered what the responsibilities of the Terrain and/or Environment Manager would be, its effect on the entities, how entities keep track of the changing environment, and more -- 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"[1].

As members of our community continue this discussion, it is important to remember that assumptions should be backed with analytical or empirical data since it affects the evolution of the design process. We urge the community-at-large to make sure that enough evidence has been presented to support the assumptions which are now driving the PDU definitions.

Disagreement on the "golden architecture" (which solves all the DIS communities needs) 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 up into a large number of entities. However, distributed architectures have problems with data redundancy, latency, and loose coupling between the CPUs on different simulators.

We believe it is not optimal to look for the "golden architecture" which will solve all DIS requirements, especially since there is a great variety of intended eventual uses for DIS with conflicting needs. *Consider accepting a different architecture for different applications.* For example, the central server will be cost-effective if it provides enough performance for a particular application with only a few entities. 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.[2]

To support a variety of disparate needs with a single system, the system's architecture must be reconfigurable. Flexibility is the *foundation* the architecture needs to exhibit.

2.2 Premature PDU Definition

As mentioned earlier, evidence should be presented to support assumptions driving the PDU designs. However, PDU designs have been presented in the

Our thanks to the US Army Simulation Training and Instrumentation Command (STRICOM), who funded this work (contract N61339-92-K-0001).

1. [IST 93] DIS Operational Concept 2.3, IST-93-25, Institute for Simulation and Training, Orlando, FL,pg. 4.

2. [Kamsickas 93] "Distributed Simulation: Does Simulation Interoperability Need an Environment Server", I/ITSEC'93 Conference Proceedings, pp.236-244.

Simulated Environment Subgroups without analytical results validating the designs. In particular, we are concerned that the definition of the PDUs be done in concert with the definition of the architectures because they will be communicating via the PDUs once they are adopted by the DIS standard. We feel that data (in the form of analytical or empirical results) needs to be brought to the table for analysis before additional assumptions are made. This allows the community to make better-informed decisions as the standard evolves.

2.3 DT System Design Issues

The question of how Dynamic Terrain can be integrated into an interactive, distributed simulation can actually be broken down into two questions. First, we must consider how to integrate a dynamic terrain capability given current concepts of distributed simulation technology (i.e., our current simulation context). Second, we must determine how to expand this current simulation context to allow for future, and as yet unknown, applications of dynamic terrain. In this paper and our current research, we address both of these issues.

When working at an abstract system design level, it is important to keep several issues in mind:

- The system should strike a balance between the constraints of network bandwidth, CPU performance, and software flexibility.
- The system should strike a balance between forward thinking and backward compatibility.

3.0 Basic Goals and Assumptions

Several design goals are fundamental to our work on an architecture which supports dynamic terrain and increased entity/terrain interaction. These goals are thematic, and variations are found in many of the major components of the system. These goals include:

1. *Decoupling the applications from the form of the underlying data* - This is imperative when the focus is on a flexible, extensible architecture. It is not safe to assume that data forms currently in use will necessarily be used in the future, nor is it safe to assume that new functionalities added in the future will necessarily be able to use any current data form. Some of the principal data forms are the terrain database, the PDU structure, and the additional attributes that will be needed to support higher fidelity interactions.
2. *Finding effective data abstractions for the principal data forms* - Decoupling the applications from the forms of the data requires the development of data abstractions that have sufficient expressive power to be generally useful. In other words, the form of the data used by the environmental models and transmitted to the simulators to reflect the changing environment should provide information on the environmental state at any resolution, size, or orientation. It is also desirable for these abstractions to be as clean as possible to simplify their use and to provide a standard interface to the environmental state for heterogeneous simulators. The real key to a flexible extensible software system is the elegance and expressive power of its abstractions.

> As we look for simulators to perform at higher fidelity levels, we should consider more robust data structures to support these levels of fidelity.

3. *Support for arbitrary numbers and types of attributes* - To provide for future demands, no artificial constraints should be placed on the number or type of terrain attributes. This goal applies in general to the data used by the system, but it also applies specifically to each simulator database instance at runtime.
4. *Flexibility, extensibility, and configurability* - To support our research goals, as well as to facilitate vendors' efforts to field simulation systems with higher fidelities, our architecture must address these three points. The broad spectrum of potential users of dynamic terrain capability only underscores the need for flexibility within the architecture. The fidelity levels and interactions desirable in future systems require an approach that provides extensibility. The research effort itself, as well as the end user, should be provided as much latitude as is feasible with respect to configurability.

We have enumerated several major assumptions about our current simulation context that define in some sense where we started and how much of the problem we are trying to deal with right now. It is important to keep in mind that these assumptions scope our efforts. These major assumptions include:

1. *2 1/2 D representation* - Even though simulated entities have the freedom to move about freely in 3D (subject to their own capabilities), the terrain is often referred to as 2 1/2 D. In other words, the only part of the terrain that is usually relevant to a given scenario is the surface or "skin".
2. *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). There is no generalized mechanism for representing caves or tunnels. Such things, if modelled, are considered as features or are otherwise handled as special cases.
3. *Polygon based representations* - Most image generators, and de facto many simulation models, are polygon based. In this representation, the terrain is essentially a large polyhedron. While research should explore alternative representations, we cannot (at least for the short term) ignore systems that are polygon based. It is also important to note here that the different polygonization schemes have traditionally been a source of problems when linking heterogenous simulators.
4. *Multiple attributes* - The current context for simulators does include the concept of multiple attributes for any given 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 color and texture coordinates, surface material codes, and soil types. In any case, it is normal to associate several attributes with any given location.
5. *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. Several resolutions are available, but the resolution is typically no finer than 1 arc second (roughly 30 meters). If other spacings are required, the implication is that some form of mathematical hocus-pocus will be performed to find the intermediate values.

4.0 Data Structures are Foundations

4.1 Impact on System Architecture

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, treelines, 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 (viewing) in high-performance graphics systems but less optimally for other simulator applications such as terrain following, terrain reasoning (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.

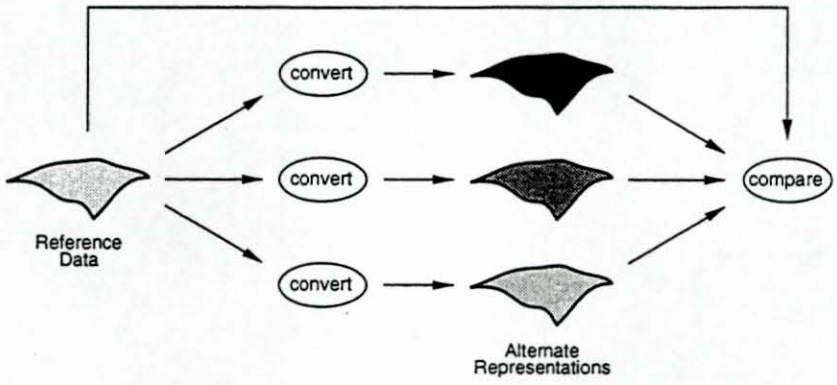
Terrain following algorithms which involve real vehicle dynamics require smoothly changing terrain slope contours in order to correctly simulate the forces on the vehicle. If the terrain is represented using a polygonal mesh, the slope changes suddenly as the vehicle encounters a polygon boundary --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

4.2 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.

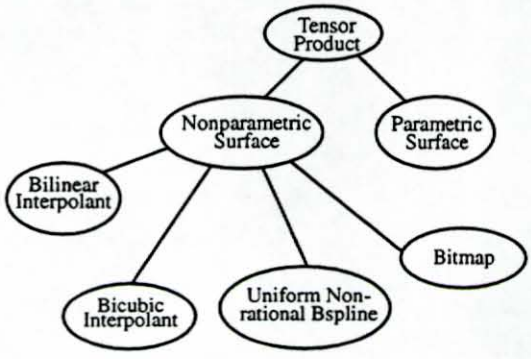
One alternative that has proven particularly useful is the mathematical surface. Note that the term mathematic surface is used generically to refer to any of several representations that might be used for any particular terrain attribute. Tesselating a surface to generate polygons is a relatively simple operation that is based on a deeper rooted concept. Mathematical surfaces can be sampled anywhere, not just where there is a data point. This concept is crucial to our further developments.

FIGURE: Surface Accuracy Experiment



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 as shown in Figure "Surface Class Hierarchy".

FIGURE: Surface Class Hierarchy



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.

4.3 An Accuracy Experiment

An experiment to study the behavior of several surface representations in the class hierarchy was conducted at IST. A section of SIF 30-meter data from the Hunter Liggett I/ITSEC'93 database was used as the basis for comparison. The process is diagrammed in Figure "Surface Accuracy Experiment".

Simulation applications may choose different surface representations based on a tradeoff between available CPU power and the accuracy of a terrain patch. The accuracy of a particular surface representation is effected by the resolution of the original "raw" database, the terrain roughness, and the number of control points used to generate a surface. Fewer control points means less data sent across the network.

4.3.1 Data Sets

For this experiment, two data sets were taken from the Hunter Liggett gridded elevation database released in Project 2851 SIF for the I/ITSC'92 InterOperability demonstration. Each data set contained 129 x 129 elevation posts at one-arc second spacing.

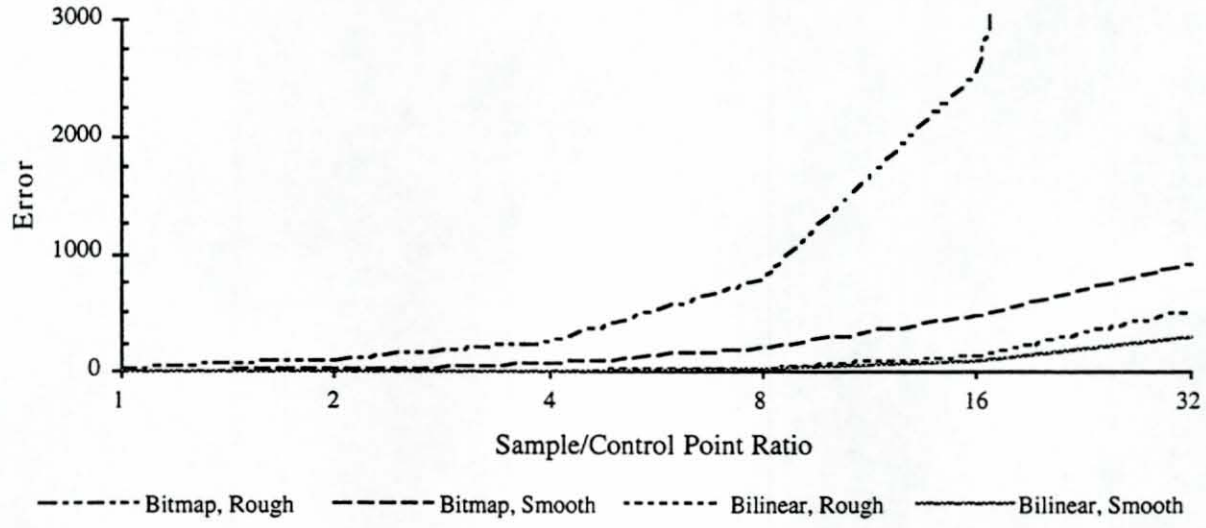
The selected data sets represented two categories of terrain roughness, σ_t[3]. Data set 1, with $\sigma_t = 77$, was classified as rough terrain while data set 2, with $\sigma_t = 45$, was classified as moderate terrain.

4.3.2 Process

Terrain patches of 129 x 129 sample elevation posts were estimated for each surface representation and for each set of control points. Sample to control point ratios of 1, 2, 4, 8, 16, and 32 were tested for bitmapped (gridded) and B-spline representations.

3. [Ackeret 90], Ackeret,James, "Digital Terrain Elevation Data Resolution and Requirements Study - Interim Report", ETL-SR-6, U.S. Army Corps of Engineers Topographic Engineering Center, November, 1990.

FIGURE: Elevation Estimate Error



Differences between estimated and original elevations were used to measure the accuracy of the surface approximations. Error regression values were computed for each sample/control ratio for each representation for both data sets using:

$$\text{Error} = [\Sigma (x^2)] / n$$

where

$$x = (\text{Elevation}_{actual} - \text{Elevation}_{estimate})$$

and

$$n = 129 \times 129 = 16641$$

4.3.3 Conclusions

Error was greater for rough terrain than for moderate terrain. Values for the bilinear B-spline and bitmapped representations for both data sets are graphed in the Figure "Elevation Estimate Error". Bicubic and biquadratic B-spline representations yielded results similar to those of the bilinear representation.

This plot shows that as the sample/control ratio increases, elevation differences increase resulting in greater error .

Bitmap error was considerably greater than that of the B-splines. Thus, gridded data may not be the best method of data storage. Data compression can be achieved through judicious selection of both a surface representation method and the number of control points. As the terrain roughness increases, more control points are necessary to maintain data accuracy.

5.0 The Dynamic Terrain DB (DTDB)

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 "The DTDB Abstraction". 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, positionally registered with respect to each other.

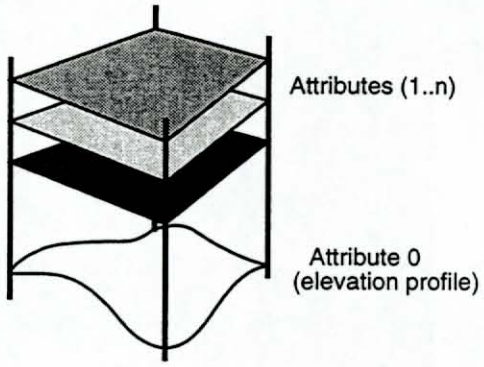
5.1 The Database Query

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 provided by the DynamicTerrainDatabase class.

FIGURE: 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. This decoupling of the user from the data representation 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).

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. 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*.

5.2 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 "The DTDB Query". 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.

The actual area form query looks like:

getAttributeValues(attr, q1, q2, q3, q4, m, n, buffer)

where:

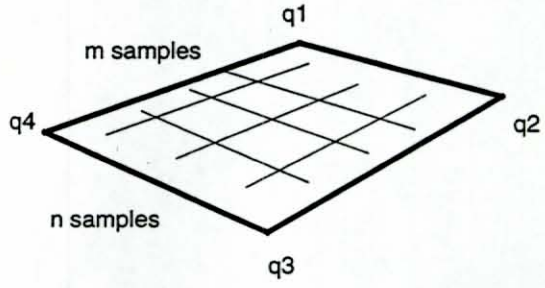
attr is the attribute to be queried,

q1..q4 are points that define the area,

m and *n* are the number of samples in each direction, and

buffer is the storage area into which data will be placed.

FIGURE: The DTDB Query



The *q1..q4* coordinates are in world space and represent an arbitrary quadrilateral area. The DynamicTerrainDatabase bilinearly interpolates between *q1..q4* using *m* and *n* to find the locations of each individual point that is used to query the underlying surface representation. The values returned by the underlying representation are stored in the data buffer. In addition to attribute values, surface normals and parameter values can be returned via variations of this query.

Another practicality associated with queries is that clients who want values for several attributes will typically want them over the same area (values of *q1..q4*) and at the same resolution (values of *m* and *n*).

To support this, a multiple attribute query has been set up to get values for an arbitrary number of attributes over the same area at the same resolution.

5.3 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.

1. *The expressive power of the area form of the query becomes more clearwhen 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.

2. *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.
3. *Throughout, the DTDB is responsible for maintaining the state of the several terrain attributes.* The state of terrain attributes can be considered analagous 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.
4. *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.

5.4 A Prototype Image Generator Using this Abstraction

A Dynamic Terrain Visualization System (DTVS) has been developed to evaluate the difficulty of processing DT PDUs containing mathematical surface control points in a simulator application. The DTVS runs on Silicon Graphics workstations and functions as a software image generator. On high-performance workstations equipped with a Reality Engine graphics pipeline, DTVS can be locked at a specific frame-rate to get real-time performance. A number of software classes have been designed, developed, and instantiated to facilitate the simulation of a Dynamic Terrain skin with SGI Performer based geometry and updates.

A Dynamic Terrain Manager is created in the application process to handle the high level aspects of DT management. The DTVS constructs a dynamic database out of a series of rectangular patches of terrain that are placed in an arrangement progressing from high-density near the eyepoint down to low-density at the horizon. Densities and the number of different levels of detail are changable options. A separate asynchronous process performs the generation and modifications to the leaf nodes of the Performer database.

6.0 What Should be in the PDUs?

From a system perspective, the 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 the I/ITSEC'93 conference demo (see Section 11.0: "IST's DT Demonstration"), an ESIG-2000 using a polygonal mesh was interacting with an SGI Onyx using mathematical surfaces.

With this in mind, the Environmental PDU already proposed in the DIS community should be evaluated to determine whether it conveys the right information. Specifically, the community should understand which portion of the environmental modeling problem this PDU solves.

7.0 Proposed Requirements of the Terrain Manager Architecture

In this section, a proposed "wish list" of requirements for the Terrain Manager's architecture is presented. It is understood that all of these requirements may not (and probably will not) be solved by a single architecture -at least not in the initial implementation. Several groups from the DIS community have contributed to this list. The most notable contributions are from IST, the Environmental Effects in DIS (E^2DIS) Architecture group[4], and the DIS Land Subgroup[5].

R1: Use a scalable architecture for the Terrain Manager in DIS. Performance should be scalable by adding processing power only.

R2: Use an extensible architecture for the Terrain Manager. As new applications are developed, the architecture should support extensions.

R3: The architecture should be flexible with respect to implementation. This enables DIS users with different goals to adapt common portions of the Terrain Manager architecture to suit their needs.

R4: Include the ability to update late joining entities.

R5: Support a hierarchical solution where local terrain managers support entities at their site. The local terrain managers will interact across a WAN DIS link to maintain consistency.

R6: Support Geographic Segmentation - allow for separate Terrain Managers to be responsible for geographically unique portions of the gaming area during an exercise.

R7: Fault tolerance achieved through a redundancy scheme.

R8: Support a query mechanism for use by entity simulators to access environmental data locally or across the DIS network.

R9: The Terrain Manager will 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[6].

R10: Resolution independence - the Terrain Manager should support varying levels of terrain resolution. Also, it should be provide environmental information which simultaneously serves entity simulators working at different internal terrain resolutions

R11: Dynamic update of environmental state will be generated via new DIS PDUs.

R12: Look towards VV&A - Any model or database that is added to DIS should be independently verified. In addition, the communication architecture and the protocol for handling environmental effects should be independently verified.

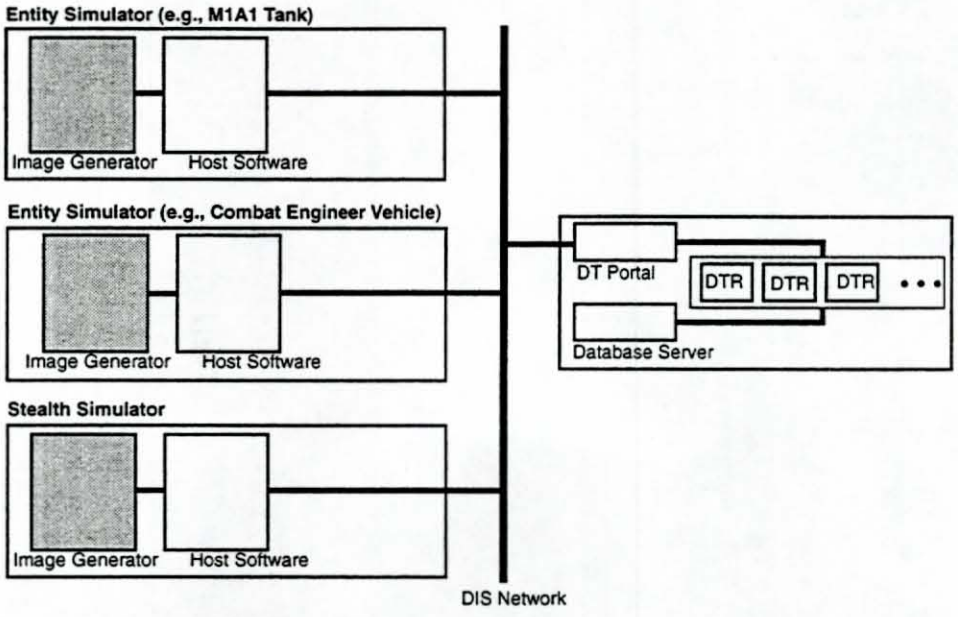
---

4. References used from meeting notes: E^2DIS Architecture Task meetings, August 1993, November 1993.

5. Developed during the Land Subgroup meeting, recorded in: "9th Workshop on Standards for the Interoperability of Defence Simulations", Volume II - Minutes from the Working Sessions, IST-CR-93-39.2, IST, Orlando, FL

6. Included because of recommendation by the Land Subgroup. However, [Kamsickas 93] advises against this approach. Our work has not addressed this requirement yet.

FIGURE: Architecture Number One



R13: Independence from any particular simulator vendor's polygonization scheme so no existing systems are excluded. However, it is assumed that the majority of simulators will use polygonal data structures for the next several years.

R14: Be consistent with or work in concert with the direction vendor products are developing.

R15: Yield increased entity/environment interaction and raise the amount of training applications possible using the DIS architecture.

R16: Incorporate appropriate-level physics of the environment and environmental effects into Distributed Simulations.

R17: Allow time-dependent models for slow, continuous representation of environmental changes (e.g. slowly-changing soil strengths as soil responds to sun intensity).

R18: Allow multiple simulation types. The architecture must be capable of handling a mixture of constructive, live, and virtual systems on a heterogeneous network.

R19: Support the Radiometric environment. Prepare for DIS extending into the area of physically-accurate sensor models. Dynamic environmental information must be available for the sensor models.

R20: The Terrain Manager should include the ability to replay scripted environmental state changes to support simulator preprocessing of predistributed data. This will aid physically-accurate sensor simulations which currently require a large amount of preprocessing to allow real-time performance. These predistributed databases can be replaced with deterministic models as they become available.

8.0 Architecture Candidate Number One

This architecture was developed first by the DT team at IST. Its focus was simplicity of design. The intent was to spend minimal time on architecture integration and optimization issues, therefore focusing on the fundamental problems which must be solved by the architecture. The architecture is shown graphically below in Figure "Architecture Number One".

8.1 Definitions

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:

• *Inbound Tasks* - Watch DIS traffic to identify events which cause dynamic terrain events (e.g. Bulldozer digging, munition impact on terrain).

• *Internal Tasks* - Handle the DT Simulation details (e.g.Remote Entity Approximations, interaction between Database and DTRs).

• *Outbound Tasks* - Output new DIS messages to simulators (e.g., Packets to represent crater caused by Detonation PDUs, packets to represent berms, craters, and new terrain elevations).

Dynamic Terrain Resources (DTRs): A specific physics-based algorithm is contained inside each DTR. The DT architecture allows for DTRs to be added as they are designed. Specific DTRs currently implemented include soil slippage and cratering from munition impact. Vehicle tracks and water flow (hydrology) models are under development.

Dynamic Terrain Database: This object maintains the simulation's environment database. Ground elevations and terrain attributes are stored here along with any information necessary for the function of the DTRs. The database design will evolve during the course of the project, but always in a way compatible with the industry standards for visual system databases.

8.2 Design Goals of Architecture #1

During the development of architecture #1, we were investigating the scope of dynamic terrain in the DIS paradigm. Our first architecture was designed to serve several purposes:

Provide Information - Yield as much information as possible about the dynamic terrain problem - particularly in the areas of network bandwidth, CPU performance, and the consequence of partitioning a database geographically. For example, "How many transactions are generated on the database for a particular network load?", "What is the data transfer rate between the DTRs and the Database? (i.e. do they have to be on the same CPU?)".

Benefit from Existing Academic Research - As requirements for the architecture were uncovered, force the solution into areas where classical computer science and engineering research have already provided solutions. Such as:

1. Record locking consequences for a shared database resource. This gives insight into the structure of the Dynamic Terrain Database.
2. Distributed Computing across a conceptual network. Operating system development has addressed load balancing of simultaneous processes.
3. Job scheduling and queueing theory research can be applied to job scheduling in the Dynamic Terrain Portal.

8.3 Analytical Results

A software simulation of this system was developed first and evaluated against DIS PDU streams to investigate the behavior of the system under loaded conditions.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 the record-locking and how this affects the timing of multiple jobs simultaneously processed by the Terrain Manager.
- The DT Portal became a bottleneck since it is the only point of connection between the DIS network and the DT Resource models.
- The system throughput delay allowed for calculation and broadcast of environmental changeshad a direct impact on the complexity and cost of the implementation. This architecture was not scalable enough because of access conflicts to the single, shared Database resource and the traffic load through the DT Portal.

8.4 Summary

It was understood before the development of Architecture number one, that this design did not fulfill all the requirements of the Terrain Manager. However, it was conceived as a step in the process requirement definition and provided useful experimental results.

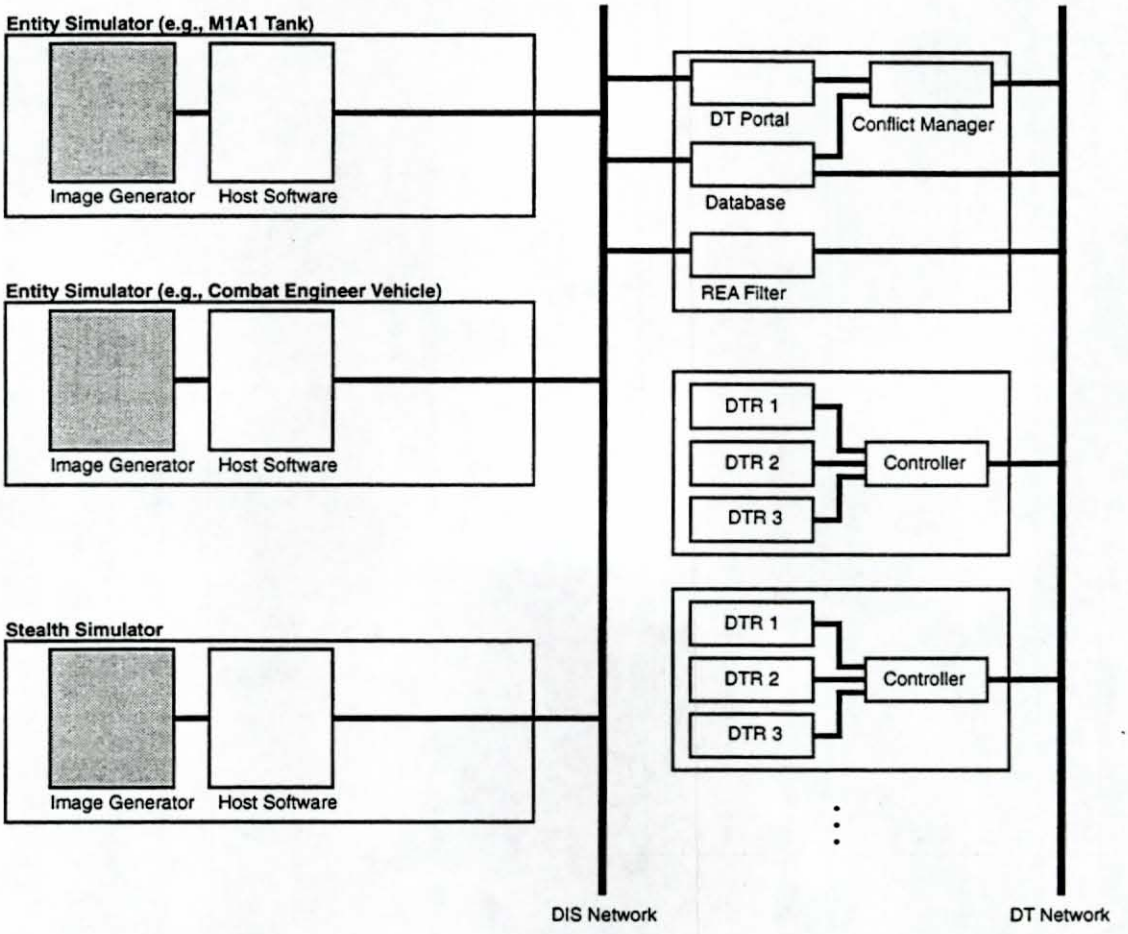
9.0 Architecture Candidate Number Two

Architecture Number Two is based on Architecture Number One with more detail focused on the Terrain Manager implementation[7]. Development began with several goals in view:

---

7. [Horan 93] , Horan,Smith,Altman,Lisle, "An Object-Oriented Environmental Server for DIS", IST Visual Systems Laboratory, presented at the Ninth Workshop on InterOperability and DIS, Sept., 1993, Orlando, FL.

FIGURE: Architecture Number Two



1. Go into greater detail with the Terrain Manager design. Look for efficiency increases and any major problems that would be uncovered by a more detailed "flushing out" of the original architecture design.
2. Expand the parallelism of the Terrain Manager. Look at the object responsibilities and how they could be allocated among a varying number of parallel processors.
3. Support dynamic load balancing among multiple processors with a desire that all physical modeling jobs working over a specific geographic area (e.g. a crater and water flow from the crater) be executed on the same processing element. This requirement was added to allow sharing of a local database cache by the Resources.
4. Avoid the bottleneck present in architecture number one: only a single interface point with the DIS network existed for all PDU traffic (the DT Portal).

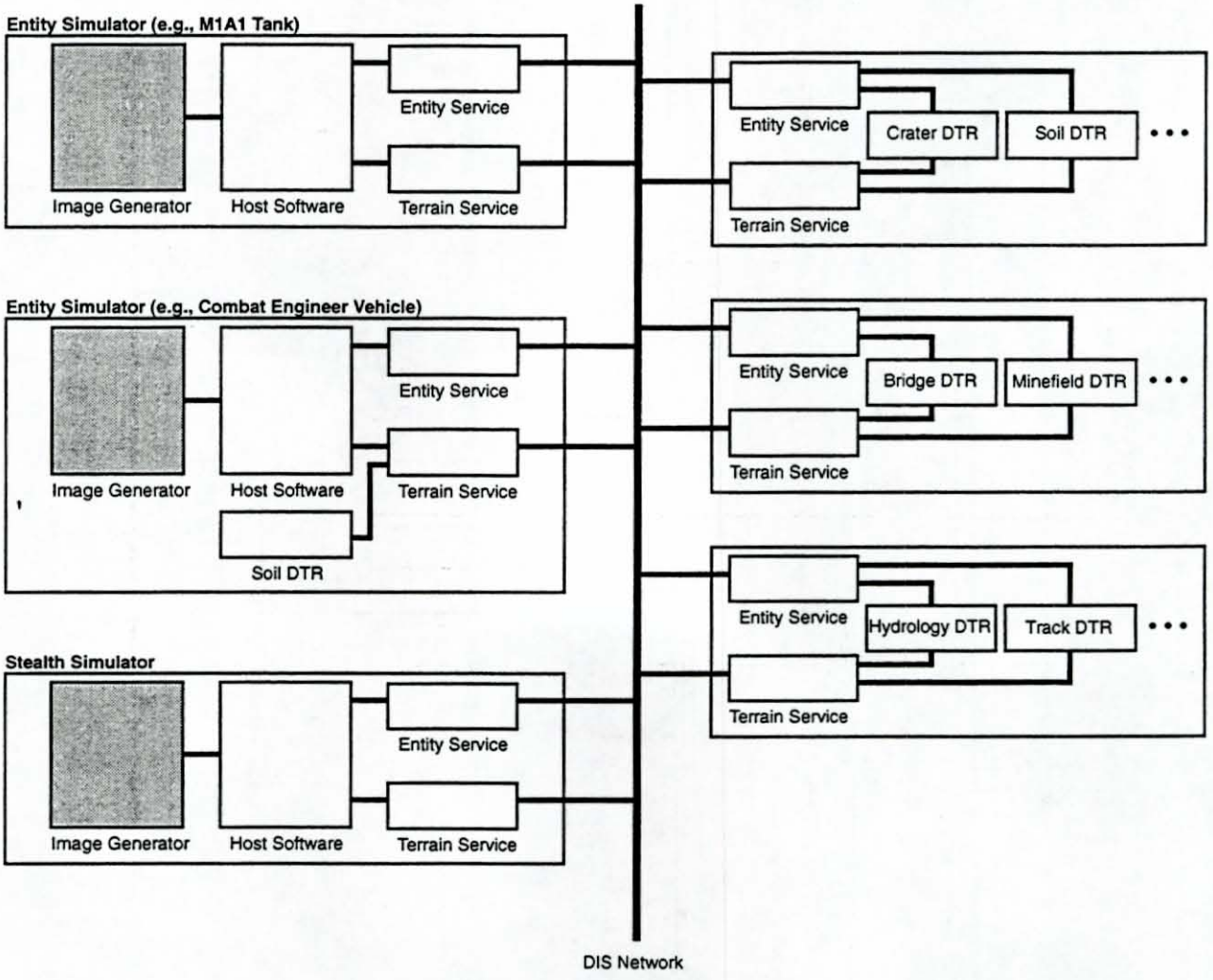
The breakdown of software objects within this version is shown in Figure "Architecture Number Two".

As the first architecture was reviewed, it was realized that this architecture could be applied to more than just the terrain. Therefore, several of the terms were redefined before the design was presented in a public forum for review:

9.1 Analytical Results

A software simulation of this architecture was performed as a Master's Thesis by a student on our project team[8]. The software simulation exhibited both expected and unexpected behavior:

FIGURE: Architecture Number Three



- A separate connection to the DIS network for Entity State PDU traffic and dead reckoning simplified the functions of the Portal. The Portal's main purpose became managment of the processing jobs within the Terrain Manager. This resulted in improved software design (more encapsulation).

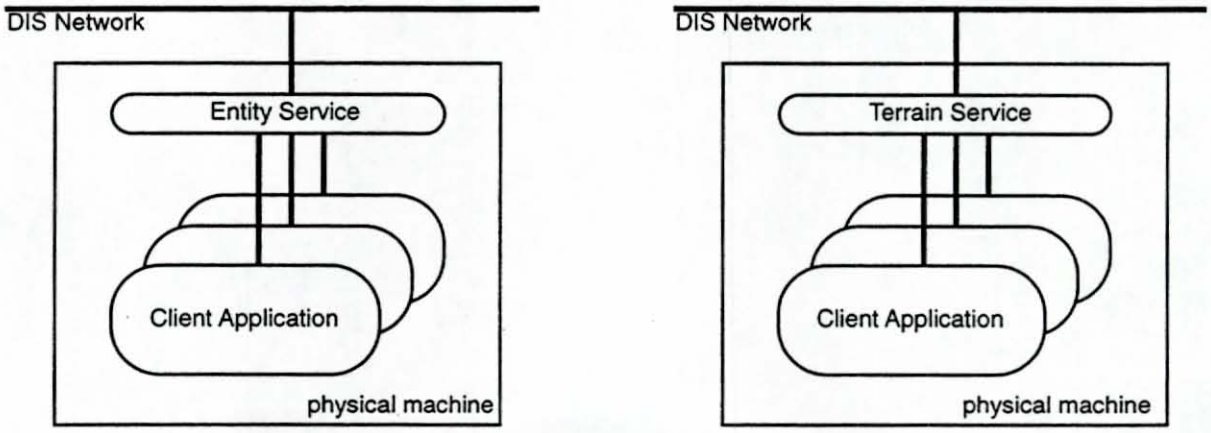
- However, even with a dedicated dead-reckoning input path to the Terrain Manager, network I/O was still a limiting factor. The parallel processors inside the Terrain Manager were not working fully loaded with jobs because of the I/O bottleneck.

- The DT database concept remained similar to that of the first architecture. However, the limitations of the single Database showed up more clearly in the simulation results with this architecture. The database model serialized transactions from multiple DT Resources (physical math models) by queueing requests.

- The heuristics developed to locate and migrate jobs according to the geographic location of the job and the load of the CPUs were not exercised often because of I/O bottlenecks between the CPUs and the network.

8. [Horan 93 #2], Horan, Bill, "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.

FIGURE: Entity and Terrain Service



9.2 Summary

- This architecture was more scalable than the first architecture. However, the database bottleneck eventually limited the performance of this system under loaded conditions.
- This architecture improved over Architecture Number One in the area of Fault Tolerance. Failures of any of the Machine Controllers or Resources were easy to recover. However the Portal and the Dead Reckoning functions still required redundancy if the system was to remain tolerance of hardware failures.

10.0 Architecture Candidate Number Three

At this point in the project effort, we had performed several interesting systems designs but did not have a system with the flexibility desired in the original requirements. See Section 7.0: "Proposed Requirements of the Terrain Manager Architecture". Therefore, we used a different approach built upon the assumption of utility processes running in the background of all simulation host computers in the DIS exercise. The shared environment was implemented through the interaction of these utility programs, called *services*. See Figure "Entity and Terrain Service".

Within the Client/Server paradigm, the services are the servers while the simulation application programs are the clients. In this paradigm, vehicle simulators will have the same interface to the dynamic, shared environment that a physical model (e.g. a soil slumping model) will have. With this approach, the overall system looks like Figure "Architecture Number Three".

10.1 Entity Service

One of the requirements of the DIS environment is that each node maintains representations of other entities within its area of interest (ghosts, or remote entity approximations). Entity State PDU's are read from the DIS network and dead reckoning is performed on these approximations. In an environment where there is only one application per node (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, the location of this functionality becomes important.

Consider the capability to have several applications resident on one node. At first, this seems a superfluous requirement. However, further investigation reveals just how useful this could be. Many of the functionalities associated with our dynamic terrain effort are individually relatively small. In particular, many of them are too small to warrant allocation of an entire machine. This leads to two alternative fundamental approaches. Many of these small functionalities could be bundled into one large, eclectic application, or these functionalities could reside in separate applications.

If put in one large application, the question of where to place the entity information functionality is answered easily. Yet, both development and maintenance are unnecessarily complicated. If put in several small applications, the software issues are mediated by stronger encapsulation and weaker coupling between components, but where to put the entity information functionality becomes a problem.

The requirement becomes a single application that can perform DIS I/O (input/output), do dead reckoning for the remote entities, and support several simultaneous

client applications that require subsets of this entity information and running at different frame rates. This application is referred to as Entity Service.

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 EntityService 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. Each user "feels" as if he has the machine to himself.

We have encapsulated the functionalities of DIS communication and dead reckoning within the EntityService. 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 more tightly encapsulated.

> *"Complexity can be reduced by designing systems with the weakest possible coupling between modules."* [9]

10.2 Terrain Service

In a fashion similar to the development of the EntityService, a TerrainService has been created to handle the state of the terrain. The TerrainService resides in its own process and contains an instance of the DynamicTerrainDatabase class described above. It serves as intermediary between the DTDB and the client applications for queries and updates, as well as handling transmission and reception of our prototypical dynamic terrain PDUs.

In addition, an experimental protocolwas established for the TerrainService 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.

While dead reckoning of terrain attributes is not yet implemented, the TerrainService is the likely place for this to occur. This would reduce the network traffic while still allowing the applications to receive the *most current* state of the terrain. *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.*

Central server, fully distributed, and hybrid configurations of the Terrain Manager are realized by simply modifying the handshaking between instantiations of the TerrainService on each simulator.

10.3 Analytical Results

This architecture was used for the I/ITSEC'93 demonstration by IST. See Section 11.0: "IST's DT Demonstration".

10.4 Summary

Central server, fully distributed, and hybrid configurations of the Terrain Manager are realized by modifying the handshaking between instantiations of the TerrainService on each simulator. It is important to note when considering changes to the configuration that client applications are insulated from these changes. This is a beneifit from the abstraction of client applications away from the terrain data.

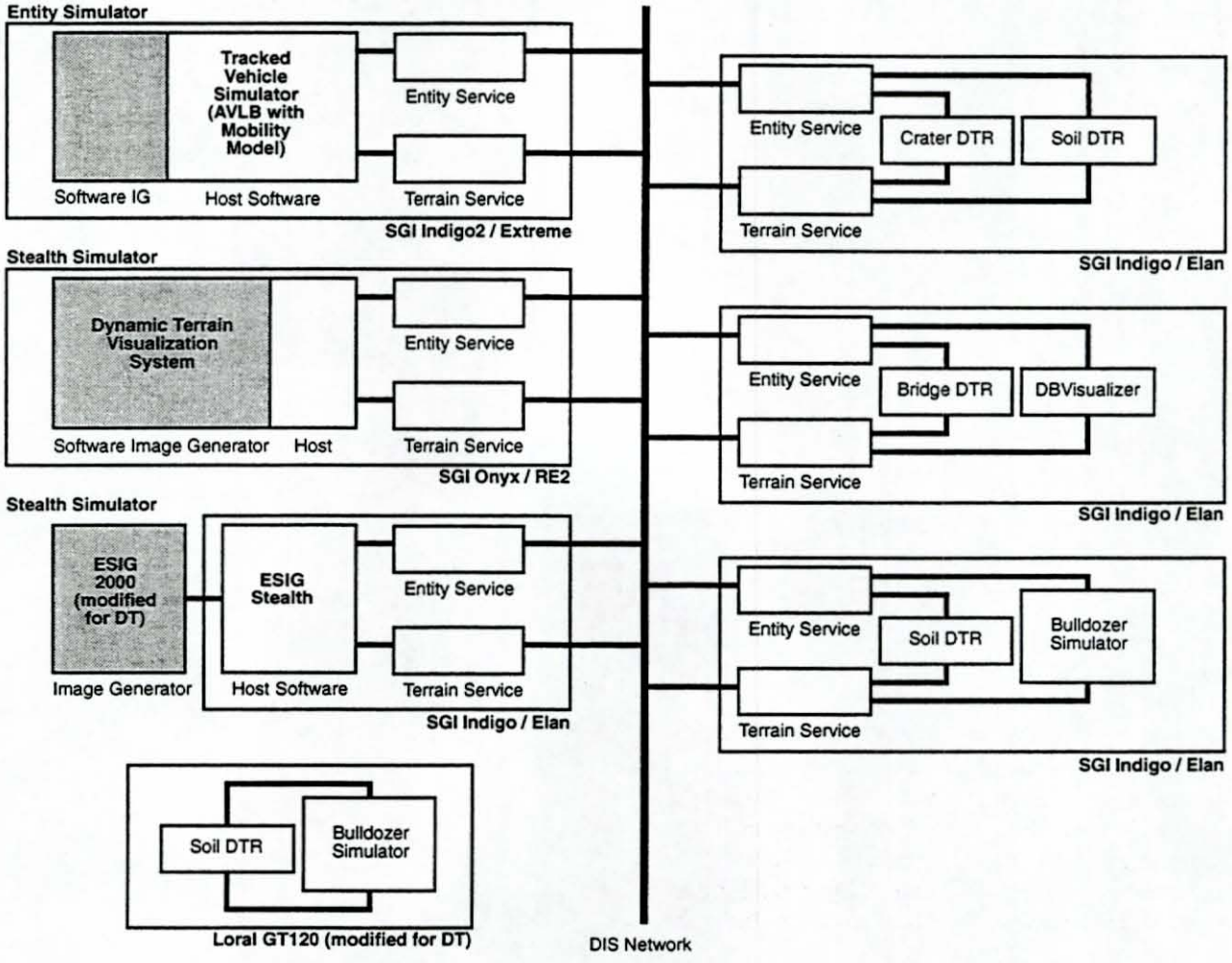
The software implementations of the physical modeling algorithms were easier to develop using this architecture due to the encapsulation of the shared environment. Even if this architecture is not adopted for widespread use in DIS, it serves as a good prototyping testbed for applications involving vehicle/environment interaction.

Since access to environment state goes through an extra level of software indirection. Flexibility is achieved at the cost of some performance. The non-optimized version of software currently running on Silicon Graphics workstations works for prototype purposes, but lacks the performance needed for high-fidelity training systems.

---

9. [Stevens 79] Stevens, Myers, Constantine, "Structured Design, in *Classics in Software Engineering*", New York, NY, Yourdon Press.

FIGURE: IST's I/ITSEC 1993 Dynamic Terrain Demonstration



Load balancing or associated database conflict resolution management had not been implemented at the time of this writing. We expect the flexibility provided by the TerrainService to make the architecture tolerant of changes in this area. We also acknowledge that more research is necessary to find the tradeoffs between the central server, fully distributed, and hybrid configurations.

11.0 IST's DT Demonstration

The Institute for Simulation and Training demonstrated a working Dynamic Terrain system at the 15th Interservice / Industry Training Systems and Education Conference 1993 (I/ITSEC'93) in Orlando, FL. Essentially an implementation of Architecture Three, the demonstrated capability included un-scripted changes to the gaming area database which were then displayed on ESIG-2000, Loral GT120, and Siligon Graphics Onyx Image Generators. The demonstration network was as shown in Figure "IST's I/ITSEC 1993 Dynamic Terrain Demonstration".

The demonstration consisted of a local DIS network with a collection of Silicon Graphics Workstations operating various components of the Dynamic Terrain Testbed. Included on this local network were simulations of earth moving equipment (such as bulldozers), utility vehicles, and other simulated events that change terrain (such as artillery rounds). This demonstration addressed our project goal to explore means of integrating new DT technology into both current visual systems and future simulation systems.

To illustrate this goal, a modified ESIG-2000 was provided by Evans & Sutherland. Through this

collaboration, the demonstration included a networked bulldozer simulation. This simulator, running on an SGI workstation, could cut into the simulated terrain with the resulting changes appearing on the ESIG-2000 and other machines on the local DIS network. The ESIG-2000, along with other SGI-based simulators, had the capability to generate craters instantaneously due to incoming rounds produced by other simulated entities (Figure "Dynamic Terrain on the Evans & Sutherland ESIG-2000"). The SGI Onyx was functioning as a Stealth platform using the IST-developed Dynamic Terrain Visualization System software (Figure "Dynamic Terrain on the Silicon Graphics Onyx").

Additional assistance was provided by Loral Advanced Distributed Simulation. Providing a modified GT120 image generator and using IST's dynamic soil model, Loral demonstrated how this technology could be used to change SIMNET terrain databases in realtime. Thus Loral illustrated how portions of IST's Dynamic Terrain Testbed could be implemented in currently fielded systems. This collaboration with industry vendors provided valuable feedback on IST's DT approach.

12.0 Summary

This paper has addressed several issues that must be considered when addressing a system-level architecture for dynamic environments in DIS. These issues have been developed through implementation of multiple prototype architectures to support dynamic environments during the course of Dynamic Terrain research conducted at IST. These issues are listed as follows:

- Environment state can be considered in a matter consistent with vehicle state. Occasional broadcasts and local dead reckoning can be used for network bandwidth. This is a natural extension of the DIS protocol.
- The Client/Server approach allows software encapsulation and makes prototype software easier to transport between different architectures.
- The flexibility of the software and architecture supports a long software lifecycle 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. Don't try to apply today's solution to tomorrow's problems.
- The issues surrounding dynamic terrain, dynamic environments, and the environmental server 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 paper come out of a detailed study of the problem of dynamic terran in distributed simulations. However, many problems remain unresolved and must be addressed by further research.

FIGURE: Dynamic Terrain on the Evans & Sutherland ESIG-2000

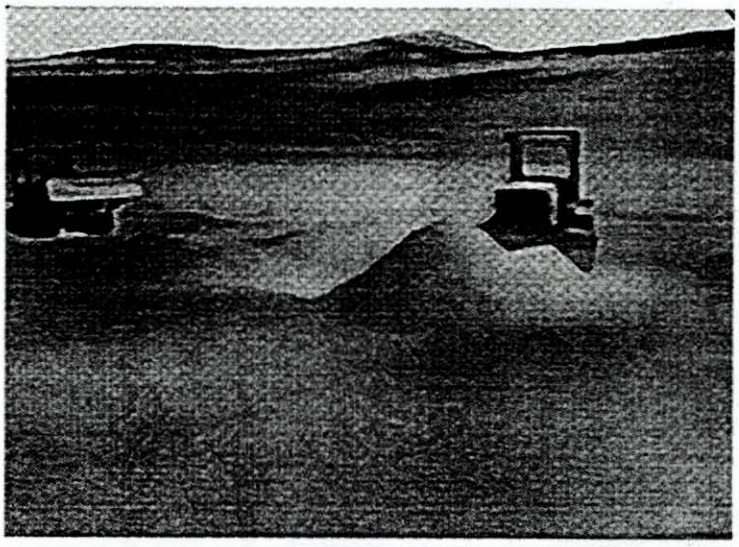


FIGURE: Dynamic Terrain on the Silicon Graphics Onyx