1-1-1990

A Testbed For Automated Entity Generation In Distributed Interactive Simulation

Gilbert Gonzalez

INSTITUTE FOR SIMULATION AND TRAINING

A Testbed for

Automated Entity Generation in Distributed Interactive Simulation

Intelligent Simulated Forces Laboratory

Gilbert Gonzalez
Daniel Mullally
Scott Smith
Amy Vanzant-Hodge
John Watkins
Douglas Wood

IST

August 15, 1990

Institute for Simulation and Training
12424 Research Parkway, Suite 300
Orlando FL 32826

University of Central Florida
Division of Sponsored Research

IST-TR-90-15

A TESTBED FOR AUTOMATED ENTITY GENERATION IN DISTRIBUTED INTERACTIVE SIMULATION

Technical Report

PUBLICATION NUMBER IST-TR-90-15

GILBERT GONZALEZ
DANIEL MULLALLY
SCOTT SMITH
AMY VANZANT-HODGE
JOHN WATKINS
DOUGLAS WOOD

INTELLIGENT SIMULATED FORCES LABORATORY

The Institute for Simulation and Training
12424 Research Parkway, Orlando, Florida 32826

15 AUGUST 1990

University of Central Florida

Table of Contents

1 INTRODUCTION

1.1 PURPOSE

This technical report is submitted as the first of two deliverable items specified under Task 3c of Workplan 2, dated 13 November, 1989 for DARPA contract N61339-89-C-0044, **INTELLIGENT SIMULATED FORCES: EVALUATION AND EXPLORATION OF COMPUTATIONAL AND HARDWARE STRATEGIES.** It discusses the configuration of the "Semi-Automated Forces" testbed hardware and software demonstrated at the May 1990 Quarterly Review at IST, and the capabilities of the testbed as planned and as currently implemented.

1.2 DISTRIBUTED INTERACTIVE SIMULATION

Distributed Interactive Simulation (**DIS**) is an exercise in which a number of simulation devices are interconnected and the individual entities generated thereon are able to interact within a shared virtual environment. The simulators may be interconnected at one geographic location by a **Local Area Network** (**LAN**), or they may be physically dispersed but linked via a Long Haul Network.

DIS may be applied to simulations of many types but this discussion addresses the application of battle simulation involving a number of different weapons or sensor platforms. A simulator may generate one or more platforms. Within this discussion, the term **entity** is used to describe a platform such as a single aircraft, armored vehicle, missile site, dismounted infantry unit, etc.

Within a DIS exercise, a collection of simulated entities that are operationally linked to cooperate (or in this case, fight together as a unit) will be called a **force.** A computer generated environment within which the simulated entities interact will be called an **arena.**

1.3 ROLE OF AUTOMATION IN DIS

Certain aspects of the training applications of DIS may require the generation of large numbers of entities in an arena to train a relatively small number of individuals. This is a valid use of a large scale system, however, using a fully manned simulator to generate each entity in a large force would require a large investment both in equipment and human resources.

In some cases the training may be directed toward commanders and there may be no requirement to train the individual crews

and squads of manned vehicle simulators. In other cases an opposing force may be required and there may be few, if any, individuals familiar with enemy doctrine and procedures to provide the crews for manned enemy vehicle simulators. Indeed, there will probably be no manned simulators designed specifically to represent enemy vehicles.

In such situations, the need exists for automated players. These are entities whose behavior, including that of their simulated crews to observe, maneuver, communicate, and generate and carry out plans, is generated by software. The term **Computer Generated Force** (**CGF**) is used in this discussion to refer to the combined hardware and software system used to generate a set of automated players. It is also used to refer to the collection of players themselves as a cooperating military unit.

Depending on the number of entities simulated and their organizational structure, a CGF may be tasked with certain missions or goals. These goals may be specified at the beginning of an exercise by an individual acting as an exercise controller or they may be provided during the exercise by an individual acting as the commander of the automated force. At some level, though, there must be at least one instance of human mission assignment or control.

An interaction or interleaving of human control and software generated behavior is possible at levels below the highest. When the option is provided for replacement by direct human intervention of functions available via software, a **Semi-Automated Force** (**SAFOR**) exists.

1.4 THE STATE OF THE ART OF AUTOMATED FORCES

Many different approaches have been taken to solve the problem of simulating the behavior and performance of military units. Few simulations, however, have been of the realtime/DIS type. At the beginning of this project, a survey was conducted. It was the conclusion of the study that:

> "...the emphases of the activities in intelligent simulated forces could be classified into two categories. The first category can be referred to as Intelligent Simulated Forces....This type of effort tends to focus on the development of models which exhibit realistic micro behavior at the individual vehicle to company level. The second category of models can be classified as Battlefield Simulation...It tends to deal with larger units of the force and seems to emphasize exercising of strategies and tactics." [2]

Numerous efforts have been directed toward the second category such as the "hex-based" wargames like CACTUS. The report stated that the BBN SAFOR (vintage August 1989) exemplified the first category and...

> "was an example of a model that appeared efficient when directed toward the application for which it was intended, but began to break down when extended to cover both classes of models." [2]

BBN's SAFOR, as currently implemented, (software version V3.9) requires intensive micromanagement by the human operator to the extent that he cannot play the part of the individual directing the force (platoon commander, squad leader, etc.) without concentrating considerable attention on the actions of individual entities. This result was apparently foreseen but judged to be an acceptable tradeoff for the risk inherent in attempting to implement an AI based solution to the generation of behavior.

In August, 1989, an independent panel reviewed BBN's efforts and made a number of observations and recommendations that have influenced the direction of this effort. They indicated that there is some value in improving route planning and terrain reasoning techniques and that some issues, such as learning, may be very difficult, but would contribute significantly to the realism of the behavior generated, see [5].

IST researchers feel that it will not be possible to scale up such a system to generate large scale intelligent forces without a proportional scaling of the effort required by human controllers. They also feel that the specification of "new" behavior within that system will require a great deal of hard-coded effort to compensate for a lack of a solid foundation of intelligent "automated" behavior from the entity level upward through all levels built upon them.

At this time there is a need to determine the operations, functions, requirements and components that are fundamental to the generation of credible and robust behavior of automated entities in a DIS environment. This project is an attempt to discover, implement, and demonstrate these requirements in the form of a research testbed to be made accessible to others working in the area.

1.5 COMPONENTS OF A COMPUTER GENERATED FORCE

Within DIS, interaction between simulators is accomplished through the exchange of messages or **Protocol Data Units** (**PDU**).

For the purposes of this discussion, a PDU will be equated to a single message transmitted via a network. PDUs are used to:

- Convey the appearance or location of some entity
- Describe some specific action, such as firing a weapon or energizing a radar
- Describe a request on the part of an entity, such as a request for resupply or repair
- Represent an acknowledgement of a request
- Convey simulated communications information, such as a radio transmission
- Organize and control the simulation session and perform other tasks related to information gathering for training purposes

All players on a DIS generate PDUs describing themselves, and all will receive PDUs generated by other players. Like a manned simulator, a CGF is a source of PDUs. The difference is that the behavior of the vehicles and other entities provided by the CGF is generated by software and not by direct human control, see [23].

To be useful and to provide a credible simulation, the behavior expressed via a CGF's PDUs must be indistinguishable from that generated by manned simulators. Therefore, what is needed is a mechanism to simulate enough of the actions of the automated entities to allow PDUs to be built in realtime to give them the appearance on the network of manned simulators. They must move realistically, appear to make reasonable decisions, respond appropriately to changes in the environment and otherwise interact properly with the other elements of the DIS.

For simulated entities to move reasonably and appear realistic to observers in the DIS, some aspects of physics must be considered when determining their locations, speeds, and accelerations. To accomplish this, a **Dynamics Model** must be implemented for each kind of entity.

One of the fundamental requirements of an automated entity is that it appear correctly positioned in space. Correct placement of land vehicles on the surface of the terrain requires information about the elevation and slope of the surface and even about the material of which the surface is made.

The mechanisms by which a vehicle's physical behavior is kept consistent with the physical characteristics of the environment is called **Terrain Correlation.**

In addition to generating cues describing the physical behavior and appearance of the entities (acceleration, roll, pitch, yaw, dust clouds, sound, smoke and flames, etc.) a CGF must generate the behavior that results in these changes in appearance. This often means simulating the human decision making that would be controlling these entities were they produced by manned simulators.

The aspects of the individual that must be considered here include environmental sensing, reactions to stimuli, the generation and following of short and long term plans, communication and cooperation with other entities, and vulnerability to events.

The ability of an entity to correlate and evaluate various aspects of its surroundings and its internal states may be called **Situational Awareness.** The data required for this includes information about the terrain and other entities in the vicinity. The mechanisms by which a decision making entity interprets the physical characteristics of its environment may be called **Terrain Reasoning.** This includes analysis of such features of the terrain as contours, obstacles, cover and concealment, and intervisibility. Short and long term planning will make extensive use of this facility.

Situational Awareness must also incorporate the ability to detect, track, and interpret the actions of other simulated entities. Location of threats, tracking of moving objects, and detection of emissions from other simulated entities requires an ability to monitor the actions (expressed in their PDUs) of other players. This facility may be called **Environmental Monitoring.**

Collection of data describing the environment does not, by itself, generate behavior. The formation of sequences of actions directed toward goals is by far the most complicated and difficult portion of the task.

Simulation of the thought processes performed by one or more players is a complex problem requiring the coordination of multiple cooperating (and sometimes conflicting) efforts. In order to gain a better understanding of its operation, one may attempt to partition human decision making into subcomponents. When this is done, it may appear that a number of different kinds of behavior are involved and these different tasks may require unique models for description and simulation.

Some aspects of behavior, such as control of a vehicle by a driver, may be most easily described as a series of states with definite transitions caused by certain events or conditions. Plans generated to carry out a mission might be expressed as combinations of lists of tasks to be accomplished serially and lists of tasks to be performed simultaneously. Rules for making decisions about the relative values of threats or choices of actions to take may sometimes be expressed as collections of conditional statements.

Whatever the results of attempts to categorize and understand decision making requirements, there will be a requirement for various **Behavioral Modeling** facilities which will require information in some form to guide their operation. The static information used by a Behavioral Modeling component will be referred to as a **Behavioral Database** to differentiate it from the dynamic information derived from Environmental Monitoring and Terrain Reasoning.

A CGF must interact with other simulation elements via a communications medium. Whether the simulated medium channels visual information, radio traffic, auditory cues, or information describing physical contact, some mechanism is required to coordinate transmission and reception of PDUs which encode the information. This is referred to as the **Network Management** facility because computer network facilities are the prime mechanism at present.

Finally, the CGF must be built on some sort of software structure that provides the computational resources, access to behavioral databases, sequencing, and other tools required to perform a simulation. This may be performed by a computer's operating system or by a special purpose **Executive** or **Monitor** program.

1.6 PROBLEMS IN BUILDING AUTOMATED FORCES

Software implementation of any one of the components mentioned in the previous section could probably be accomplished in a multitude of ways. Numerous algorithms could be devised to track moving entities and various commercially available expert system shells might be capable of generating decisions. However, some basic information requirements must be met to supply any of these specific solutions with data.

1.6.1 COORDINATED TERRAIN REASONING

Terrain Reasoning requires a Terrain Database arranged in a fashion that allows rapid and efficient access. This database must also correlate closely with the databases used by manned simulators to generate their own terrain interaction and with

visual databases used by Computer Image Generators to depict the environment.

Terrain data has many uses. Rapid updates of elevation and ground slope data are used to allow a land vehicle to generate accurate orientation cues. Less often, the terrain data is required for obstacle avoidance and route planning. Data is required at different levels. Route planning may require information for a relatively small area or may require access to data structures describing entire networks of roads or rivers. The means by which a terrain database (or databases) is structured determines its suitability and convenience for various purposes. An excellent discussion of some of these problems is to be found in [27].

Lack of correlation has the potential to lead to the appearance of unrealistic behavior on the part of CGF entities when they are viewed by individuals on other kinds of simulation nodes.

1.6.2 VEHICLE DYNAMICS AND STABILITY

Vehicle dynamics models must generate motion and attitude information of sufficient quality that the entities are not distinguishable from manned simulators by observers in the DIS. However, it appears that CGF vehicles do not always need to model everything with the same level of detail required by manned simulators, especially those incorporating motion platforms. They must, however, account for problems inherent in dynamic control systems, especially as there is no human feedback incorporated. Update rates and their consequent effects on stability must be considered.

1.6.3 PARTITIONING, INTERFACE, AND MODIFICATIONS

Partitioning of tasks between hardware resources (multiple CPUs) or software modules creates a requirement to specify their interfaces. A balance must be maintained between the specification of the details of the interfaces and the resultant effects on the functions and capabilities of the modules implementing the different categories. The interfaces must be developed together with the functions of the modules that use them. A judicious placement of interfaces eases the inevitable problems arising from later modifications to functionality.

1.6.4 EXPANDABILITY

A desirable characteristic of a CGF would be the notion of expandability. This concept of the potential for increased functionality after initial development may be viewed both horizontally and vertically.

Development of "behavior" for a CGF is largely a process of **Knowledge Engineering**. Extraction of rules and descriptions of behavior from **Domain Experts** is necessary in order to gain some confidence that the behavior to be generated will be correct. If it were possible to reuse behavioral subfunctions that were developed for, say, an armored vehicle, to develop the capability to simulate an unarmored wheeled scout vehicle, the potential to save immense amounts of work would be realized. For this reason, an **Object-Oriented** design is likely to be a wise approach to the overall design philosophy. Whether this would be best implemented using an "Object-Oriented" language remains to be seen. However, the capability to expand horizontally, by adding new types of entities and new functions to preexisting entities is necessary and entirely possible using good software engineering techniques.

In the vertical dimension, however, expandability means grouping lower level organizational units into higher levels and being able to construct new higher order behavior on top of robust and complete lower level capabilities. In order that a squad may be simulated faithfully, it may be necessary to simulate each individual or team within the squad to some level of detail and with some level of autonomy.

While it may be possible to impose some organizational control on each individual element of a group via an artificial external control module, such mechanisms may break down when complicated interactions are required. It is more likely that coordinated behavior would be successfully generated when individual instantiations of the components execute autonomous behavior with cooperative goals included.

This may be best illustrated by examples. First, take the case of maneuvering in formation. For a vehicle to maneuver to its assigned station in a moving formation it must determine a course and speed to generate the necessary relative movement within some period of time. This may be done independently by each individually simulated entity or it could be performed by a coordination algorithm that directed all vehicles within a formation. The first method would require a vehicle to make decisions based on its own observations of the behavior of surrounding entities. The second method would require a rather "omniscient" point of view. Both cases are complex and may be very difficult to solve in a manner that could avoid collisions. It is likely, however, that solutions based on the limited local knowledge of each entity would be more likely to

generate realistic "human" behavior than solutions which had the benefit of complete knowledge of all parties' intentions.

A second example involves organizational affiliation and control. If every entity maintains a concept of its chain of command and its own position within it, it may be more likely to model attrition realistically, including the confusion that may arise when a link is removed from the chain. Control should pass in a defined manner when a leader is eliminated and an entity should still be able to carry out its standing orders in the event it becomes leaderless.

2 CONTRACT REQUIREMENTS

Workplan Number 2, dated 13 November 1989 states that the goal of Task 3 is:

"to develop, evaluate, and demonstrate a testbed based on off-the-shelf advanced microcomputer and coprocessor technology as a viable implementation strategy for simulated opposing forces."

The baseline testbed concept is to employ ASAT (Perceptronics PC-based **Avionics Situational Awareness Simulator**) technology as a framework to investigate applications and utilities of such technologies as off-the-shelf advanced 80386 multitasking personal computers, low cost CIGs, rule-based expert systems and neural net simulators.

The testbed is to be capable of implementing a "narrow-slice simulated forces demonstration" interfaced to SIMNET.

Subtask 3c requires the acquisition and development of hardware and software for the testbed, including schedulers and executives, man-machine interfaces, basic simulation models, intelligence modules and preliminary databases, displays, and network interfaces and various combinations of coprocessors. A prime concern is the capability of the testbed to operate as part of a SIMNET network at the end of the project.

Subtask Deliverables include:

1. A demonstration of the basic testbed capabilities. (Completed at the quarterly review on 15 May 1990 at IST)
2. A technical report describing the configuration, parameters and results of the demonstration.
3. A final report documenting the capabilities of the various combinations of coprocessors to simulate opposing forces.

This report is submitted as item 2 in the list above.

3 TESTBED REQUIREMENTS

The workplan requires a testbed that provides a flexible, reconfigurable tool for experimentation with, and evaluation of, hardware and software solutions to problems arising in the development of a CGF.

There are many elements which must be investigated. The following is a partial list:

- Simulation of vehicle dynamics
- Simulation of human decision making
- Route and Path planning and following
- Simulation of communications traffic
- Algorithms for packet filtering by Intelligent Gateways
- Intervisibility calculations
- Terrain reasoning and navigation
- Simulation of command, control, and coordination of multiple units
- Determination of the capability of different types of coprocessors and computer architectures to perform these tasks
- Development and optimization of interfaces for the runtime control of the simulated forces
- Development of efficient tools for the creation of behavioral databases
- Correlation of Visual and Navigational aspects of terrain databases

With so many unknowns and so many possible avenues, an extremely flexible approach is required so that experimentation in one area may proceed with minimal side-effects in others.

The testbed was designed to satisfy the following requirements:

- To initially provide a working simulation capability for at least two cooperating entities (vehicles, dismounted infantry squads, etc.) with the capability to accept and carry out one or two limited missions, and to operate and maneuver together.

- To use off-the-shelf hardware.

- To be flexible in overall system design, allowing new modules and functionality to be added when they are discovered to be necessary, or allowing variations to be substituted for currently existing modules, all the while minimizing the side-effects on other modules.

- To allow different commercially available software packages (such as expert systems shells) to be integrated in different areas.

- To minimize the constraints imposed by hardware and system software on the design of the portions which do the actual simulation.

- To be compatible with SIMNET.

4 IST'S APPROACH TO THE PROJECT

In laying out a strategy for the design of the testbed, a number of requirements were considered. These ranged from firm specifications, such as the requirement for interfacing the system to SIMNET, through less well defined areas such as implementing a "narrow-slice Semi-Automated Force", to requirements that were determined by the researchers to be implied by the uses to which a "testbed" might be put (e.g. it must be flexible, reconfigurable, generic, portable, etc.)

As defined earlier, the term "Semi-Automated Force" refers to a CGF wherein the option is provided for replacement by direct human intervention of functions available via software at levels below the highest echelon being simulated .

This intervention may be useful for various purposes:

- simulation of the decision making capabilities of a human is extremely complex and it would be very computationally expensive to simulate some functions faithfully.

- selective human intervention would allow runtime modification of behavior with no modification of the software.

- some training may be gained by the individuals performing the intervention, especially when the opportunity arises to employ them at differing tasks.

However, to implement this sort of intervention would require complicated I/O interfaces to various levels of the simulation as well as a simulation which was already mostly automatic to provide a "pool" of facilities from which certain elements could be removed and replaced by the combination of interfaces and human assistance. In other words, to build a "Semi - Automated" System, one would first need to build an "Automated" System, then add facilities which would remove and replace selected parts and provided data formatting services to make the I/O interfaces act like the original interfaces between the software modules.

Putting first things first, the project has focused on:

- Determining which functions or modules are necessary to implement a robust automated system
- Specifying the minimal requirements of these modules
- Implementing the modules to verify the completeness of the scheme and verifying its capacity to generate a credible simulation.

The intent of the researchers is to develop a system with the potential for expansion in horizontal as well as vertical directions. Initial development, however, will keep these dimensions small. Horizontally, efforts will be concentrated on developing only a few types of entities, such as an M1A1 tank and possibly dismounted infantry. Vertically, efforts will be limited to squads and possibly platoons.

Development efforts will focus on structure. Models, such as those for vehicle dynamics, will be kept as simple as possible and possibly even rather artificial. The feeling is that more detail can be added later where necessary, if a robust and capable structure exists from the beginning.

Behavioral capabilities for a few relatively simple missions will be developed to determine fundamental requirements for individual entities. An example from the Combat Instruction Sets developed by Perceptronics (see [25]) will be implemented as a guideline. The concept of a "Narrow-Slice" will be applied both horizontally and vertically.

The workplan stated that the testbed would be based on ASAT technology. This will be true in the sense that PC/AT architectures are used. The ASAT software design, however, was found to be unusable for a number of reasons: documentation is non-existent, and the code is uncommented, poorly structured, and very tightly integrated to the point that modifications and enhancements are impractical. As is explained below, a flexible, expandable software architecture has been developed from the ground up to provide the facilities for research.

5 PLANNED CAPABILITIES OF THE TESTBED

This section presents a description of the functionality to be implemented by the testbed at the end of the development process. Currently, a subset of this functionality has been implemented. Those modules already operable will be noted.

5.1 NUMBER OF ENTITIES

The testbed will simulate the maneuvering, environmental sensing, communications, and decision making of two or more entities and will generate and receive SIMNET compatible messages to communicate with other nodes on the network.

5.2 HUMAN CONTROL INTERFACE

The testbed will communicate with an operator who will perform the tasks of the next higher level in the chain of command above the highest level simulated by the testbed. In other words, the operator will assign missions or tasks only to the highest echelon simulated and will receive feedback via text messages, visually, or computer synthesized speech.

5.3 AUTONOMOUS BEHAVIOR

The simulated entities will employ aspects of terrain reasoning to navigate through a locale described by a Terrain Database. Entities will plan routes which lead to a destination, avoid obstacles in the terrain, avoid other maneuvering entities, and may, in some cases, employ features of the terrain for cover or concealment. The terrain database will be compatible with that used by SIMNET. It may be one of the versions of the SIMNET terrain databases (such as that used by the MCC) or may be transformed from a version encoded according to SDIS [12] into a more usable version.

Simulated entities will receive orders specifying missions, interpret those orders, generate their own internal plans to carry out those missions, follow their plans, and replan when necessary.

5.4 SCOPE OF THE DEFINED BEHAVIOR

A limited set of behavioral modules or contexts will be developed, probably using one or more of the Combat Instruction Sets listed in [25] as models.

5.5 IMPLEMENTATION METHODS AND CONSIDERATIONS

Simulated entities will generate their "behavior" using a variety of mechanisms. These may include:

1. Algorithmic solutions to procedures that must be done repetitively and often.

   Example: Periodic computation of vehicle speed and location.

Example: Relative motion problems such as station keeping or interception trajectories may be solved using Maneuvering Board methods (vector manipulation techniques), see [15].

2. Finite State Machine solutions to procedures best modeled as sequences of discrete states.

   Example: Control of vehicle turning and acceleration parameters to come to a specified heading, brake to a stop, or pass through a given location.

3. Production Systems for solutions best modeled as IF-THEN rules.

   Example: If GREATLY_OUTNUMBERED and FUEL_RANGE > 10_MILES THEN BEGIN RETREAT ELSE BEGIN HASTY_ATTACK.

4. Inference Engines for solutions to problems involving partial data, levels of uncertainty, and requiring educated guesses concerning the environment.

   Example: ENEMY REPORTED IN REGION AND UNIDENTIFIED CONTACT SIGHTED AT EXTREME RANGE IMPLIES CONTACT IS ENEMY WITH CONFIDENCE FACTOR OF 75%.

5. Lists and Tables for operations such as mission planning and route planning.

   Example: A Reconnaissance Mission might be compiled into a structure consisting of several lists; A list of intermediate destinations to be reached sequentially, a list of activities to be performed continuously during the mission, and a list of activities to be triggered by events.

The testbed software is structured to permit integration of a wide variety of tools such as those listed above.

6 DESCRIPTION OF THE TESTBED

The design and configuration of the testbed hardware and software is described in this section.

6.1 SYSTEM ARRANGEMENT

The Testbed hardware located at IST is arranged as a Local Area Network with two nodes. One node consists of a SIMNET M1A1 tank simulator running in standalone mode to provide a visual display of CGF entities or an entity for the CGF to play with or against. The other node is a single processor

host computer used to generate the CGF. The nodes are linked using thin ETHERNET cable (RG-58).

6.2 CGF HARDWARE

The CGF processor is a Hewlett-Packard RS/25C VECTRA computer. It is a PC/AT machine with an Intel 80386 processor running at 25MHz., an 80387 math coprocessor, 4 Megabytes of main memory, a 304 MegaByte hard disk, a color VGA monitor, keyboard, and a 3COM Etherlink-II Ethernet interface board.

The single processor architecture was used because it was sufficiently powerful to execute an initial prototype whose primary purpose was to determine fundamental requirements. From the beginning, however, it was suspected that the processing required to simulate a single vehicle would quickly overwhelm the capabilities of one 80386. When sufficient experience has been gained with the prototype, it will be possible to reassign various software modules to different hardware platforms suited to particular types of computation.

In addition to the above equipment, Three RGB video monitors (borrowed from another contract) are used to provide a remote view of the M1A1 driver's vision blocks. The monitors were located in the same room as the VECTRA to provide a convenient mechanism for observing the behavior of the computer generated entities.

Two SIMNET STEALTH vehicles were installed at IST in mid-summer 1990. They were, however, not ready for use at the time of the demonstration, so the monitors described above were substituted. The STEALTH vehicles provide a much better tool for observation and are being used extensively at this time.

6.3 CGF SOFTWARE

The software running on SIMNET equipment is BBN's Version 6.0. It is not described in this report, which is intended to describe the CGF software developed at IST.
The CGF Software is grouped into two primary categories. **OFFLINE** software is used to develop behavioral databases, to link those databases with specific entity types in order to build simulation modules, and to create specific starting scenarios. **ONLINE** software generates CGF behavior in realtime and communicates with the SIMNET equipment.

Development of online software has outpaced that of the offline portion because the offline tools must be tailored to accommodate the specific mechanisms used online to generate behavior. It appears at this point in the development of the

project that offline database creation tools will have to be developed after the mechanisms, such as state-machine executors, inference engines, and production rule interpreters, have been assigned to different tasks and have been tested using "hand-crafted" databases.

The online software is described in the next section. The section on offline tools follows and is primarily a discussion of ideas that have been discussed, likely target areas for their implementation, and brief descriptions of likely implementations.

6.3.1 ONLINE SOFTWARE

The CGF runtime software is described in the following sections.

6.3.1.1 DESIGN AND DEVELOPMENT PHILOSOPHY

A software development strategy was adopted which provides a logical path from determination of requirements through specification of design to validation and optimization. Rapid prototyping methods were used to provide a working framework which permitted experimentation and testing at a very early stage. With this tool, a number of ideas were tried and a minimal list of required functions was compiled. Some of these were built and added to the framework until the capabilities discussed later in this report were realized.

A software design strategy was adopted which uses an Object-Oriented structure coupled with a message passing mechanism. For prototyping, this was implemented using a simple executive (or scheduler) written in C, running as a single software process to run on a single processor PC/AT type architecture under DOS 3.3.

The Object-Oriented design strategy was chosen so that once the functions of the modules were determined and prototyped, they could be rehosted or "ported" with minimal change to a hardware base with a true multitasking operating system which would provide a more efficient framework for execution. Later they could be rehosted on a multi-processor system where shared busses, shared memory, a local area network, or some other mechanism could be used for message passing internal to the CGF and various processors could be chosen for the specific requirements of different modules.

The overall plan was to develop various functional modules, to integrate them, determine the required communications and data requirements, determine the minimal set of functions required

to implement a "narrow-slice" SAFOR, and then to determine where performance and capability might be enhanced by means of specific hardware devices such as advanced coprocessors, TRANSPUTERS, signal processors, pattern matchers, communications chips, neural nets, etc.

By partitioning functions into objects from the beginning we felt that we increased our chances for making optimal assignments to specialized architectures. That is, we decided to determine what was necessary before we decided what resources we would acquire to accomplish the tasks.

6.3.1.2 ORGANIZATION

6.3.1.2.1 EXECUTIVE MODULES

The first prototype is implemented using a very simple executive mechanism to schedule the execution of short term tasks for each of the functional modules within one CGF simulation node. The executive, along with most of the rest of the simulation, is written in C. The executive is described in more detail in [9].

The executive runs as a single process under DOS 3.3. Following its own initialization and that of some of the other modules of the simulator, the EXECUTIVE repetitively services three main functions in a loop. It locates and dispatches the highest priority message that has been queued for transmission, then checks a queue of items that have been assigned various timeout value, and then checks for external input via the console keyboard.

The primary mechanism for scheduling task execution and for communication between modules is via message. Messages are placed on one of a number of priority queues by an executive service on behalf of the sender. Messages are serviced in a first in first out manner within a priority queue, but no message is serviced until all messages of a higher priority have been serviced. The priority queues are doubly-linked lists, as are all queues currently implemented.

6.3.1.2.1.1 SCHEDULING AND DISPATCH

Modules which must be able to receive messages or have tasks scheduled are assigned data structures called control blocks. A control block contains data describing the state of the module at some time and also contains the addresses of routines which may be called by some other facility to request that module to perform some action without knowing the internal details about how that module will perform the

requested action. These routines are usually used to interpret the contents of a message directed to that module.

The EXECUTIVE maintains a list of all the control blocks in the system and their addresses. When the EXECUTIVE locates the highest priority message in the queues, it unlinks it and dispatches it to its intended destination by doing the following:

1. It locates an ID for the destination in a specified field in a header at the beginning of the message.

2. It uses the ID to index into its list of control blocks and obtains the address of the destination module's control block.

3. It locates a specified field in the control block where the address of a service routine is located, calls that routine, and passes the address of the message to the routine as a parameter.

The service routine called by the EXECUTIVE is responsible for performing any actions necessary on receipt of the message and returning control to the EXECUTIVE when finished. The control block must be used to save state variables between scheduled operations.

6.3.1.2.1.2 MESSAGE PASSING

Within a CGF simulation node, messages are passed between modules by a combination of EXECUTIVE services. Messages may be of variable size and are usually created dynamically, although they may be reused if the originator and destination agree not to release the buffer.

A module sending a message provides the address of the buffer and a requested time delay to the executive service **send_msg()**. If the delay is zero, this service appends the message buffer to the priority message queue corresponding to the priority field in the message header. If the delay is other than zero, it specifies the number of hundredths of seconds the message should be delayed before being queued on a priority message queue. In that case, it will be inserted in the special sorted queue called the **timeout chain**.

When the EXECUTIVE is ready to check the priority queues for a message to dispatch, it calls the service **msg_scan()** which searches the queues and dispatches the highest priority message queued, if any.

6.3.1.2.1.3 TIMERS AND DELAYS

Periodic scheduling of actions at regular intervals, scheduling of operations after computed delays, and any other events that should take place after some minimum time period, is performed via the timeout chain which is a doubly-linked list. Messages queued on the chain are placed into the chain in chronological order, based on the time they are queued plus the delay parameter located in the message header.

When the EXECUTIVE is ready to check the TIMEOUT CHAIN for messages to forward, it checks the first item to see if its expiration time has already passed. If it has, it unlinks that element and queues it to the proper priority message queue for future dispatch. It then continues to drain the TIMEOUT CHAIN until it encounters a message that has not yet expired or, when it has traversed the entire chain.

6.3.1.2.1.4 INITIALIZATION

Initialization of the EXECUTIVE involves initializing all of its queues to an empty condition. Other modules, such as the UPDATE MANAGER and the MOVING OBJECTS MANAGER are initialized by initializing their queues and sending them UPDATE messages (which they continue to reuse throughout the simulation). It is possible at this time to automatically create other entities and to begin their simulation, either by specifying other routines or by reading and interpreting an initialization file.

Once the simulation has started, messages may be sent to the INITIALIZATION MANAGER to request the creation of control blocks for simulated entities and their subsequent activation.

6.3.1.2.2 SIMULATOR MODULES

6.3.1.2.2.1 DISPLAY

A simplified PLANVIEW display is implemented by a DISPLAY MANAGER. This consists of services to draw an aerial view of a part of the arena, services to locate the center of the area to be depicted and the scale at which items will be drawn, and routines to erase and redraw icons representing moving entities.

The PLANVIEW DISPLAY currently draws entities being simulated in different colors. Entities being simulated locally are drawn YELLOW. Entities being tracked via the MOVING OBJECTS MANAGER are drawn RED when a Vehicle Appearance PDU provides appearance information and in GREEN whenever a dead reckoned position is computed.

Orthogonal grid lines are drawn to aid in determining positions of objects and terrain features such as contour

polygons, trees, treelines, tree canopies, buildings, roads, and rivers are drawn in the currently active **regions** (for a definition of **region** see the discussion on the TERRAIN DATABASE MANAGER below) of the arena. For debugging purposes, intended tracks of vehicles may be sketched in as well as some intermediate information computed during terrain reasoning operations.

6.3.1.2.2.2 CONSOLE

For debugging purposes, and to allow requests to be entered manually, a mechanism is included to accept keyboard. This allows keystrokes to be buffered as they are entered so that other processing does not have to be suspended while awaiting a complete message.

The keyboard is polled in the main executive loop and when a character is available it is appended to a keystroke buffer or, if the character is a backspace, causes the last character to be removed from the end of the buffer. When a carriage return is entered, the entire keystroke buffer is processed.

Processing the keystroke buffer is a matter of locating destination, operation, and parameter fields in the buffer and using them to construct a message in a dynamically allocated message buffer. The new message is then sent via executive services and the keystroke buffer is cleared for reuse.

Console messages may be directed toward a number of destinations:

The DISPLAY MANAGER may be given an absolute or relative centering offset or a scale.

The MOVING OBJECTS MANAGER may be given an update rate at which it will periodically recompute the DR positions of all the objects it is tracking.

The UPDATE MANAGER may be given an update rate at which it will periodically execute the update routines of all the entities in its list. This varies the rate at which vehicle dynamics models are computed.

The INITIALIZATION MANAGER may be given a CREATE request to generate a new instance of an entity of some type at a given location with a given attitude.

Entities, such as vehicles or drivers may be given commands. Vehicles may be given commands to change speed to a given value or turn left or right at some rate. Drivers may be told to go to some location and stop or they may be told to go to and pass through a location at a given speed.

The CONSOLE and DISPLAY MANAGERs described above are intended primarily for debugging and development. They do not represent a viable design approach for an **OPERATOR INTERFACE.** A flexible, user friendly mechanism to allow a non-programmer to control a CGF must be developed but this has not become a priority at the current stage of the project.

6.3.1.2.2.3 UPDATE

The UPDATE MANAGER is a periodically scheduled service that traverses a list of all entities (vehicles) and executes a routine to update each entity (vehicle dynamics). It locates the address of the update routine in the entity's control block. If the entity is not initialized, and therefore not ready for updates, the value in the routine field is NULL and nothing is done. Specifying the update routine in the control block permits selection of routines tailored to the needs of different types of entities.

The rate at which the UPDATE MANAGER reschedules itself is variable and may be controlled via a message to the UPDATE MANAGER.

6.3.1.2.2.4 NETWORK

The modules permitting communication between one CGF simulation node and other nodes on a network form the NETWORK MANAGER. The services provided by this module include synchronous transmission and asynchronous reception of PDUs (via interrupt service routines) and byte swapping for incoming and outgoing packets.

In the current implementation, only VEHICLE_APPEARANCE_PDUs are generated for transmission or handled upon reception. Other PDU types are currently ignored when received.

Use of the network for other types of messages is anticipated. This could include simulation configuration and control, and simulated message traffic between entities such as unit commanders.

6.3.1.2.2.5 MOVING OBJECTS

The behavior of entities generated on nodes external to the CGF simulation node must be monitored to provide up to date information to the CGF. This is handled by the MOVING OBJECTS MANAGER (MOM).

The MOM receives a message from the NETWORK MANAGER corresponding to each VEHICLE APPEARANCE PDU received via the Network. The MOM creates a data structure for each entity the

first time it receives location or appearance data. Each time thereafter, it updates that data structure to reflect the last known "fix", or accurate position and attitude.

Periodically, at a rate that may be modified via the console interface, the MOM computes its best estimate of each entity's position and saves that in the data structure.

Other modules make use of the dead reckoning (DR) information by calling services provided by the MOM. The DISPLAY MANAGER may use that data to draw entities on a planview screen. Various modules may use the MOM's information to simulate the location via visual, aural, or electromagnetic means, of certain entities.

6.3.1.2.2.6 TERRAIN DATABASE

The TERRAIN DATABASE MANAGER provides information concerning the stable portion of the environment. It accesses a terrain database file containing terrain polygon information (including roads and rivers) and feature data such as the locations and sizes of trees, treelines, and buildings. The SIMNET MCC terrain database format is currently used. This is arranged in 500 meter square terrain patches.

The TERRAIN DATABASE MANAGER uses a caching strategy to provide quick access to data without requiring the entire database to be kept in memory. Patches are read from the file and kept in the patch cache area. This system attempts to reduce the number of disk accesses by using a "Least Recently Used" (LRU) algorithm. The LRU scheme retains patches that are active or that have been recently used, letting patches that have become stagnant (relatively) leave the cache to make room for new patches being read from disk.

6.3.1.2.2.7 TERRAIN DATABASE SERVICES

Access to patches is provided through the use of local terrain **regions.** A region is defined to be a square area consisting of nine patches. Each entity is assigned a region when it is initialized. The patch containing the entity's location is the central patch and the eight surrounding patches are determined and added to the region. Whenever an entity moves across a patch boundary, its new central patch is determined and the region is updated.

An entity can obtain terrain information from the patches within its region. Services are provided to extract specific data from a patch, such as a list of all the buildings or the location of the nearest tree. The elevation at an XY coordinate can be computed from the surface polygon

information. This is used by another service which computes the orientation of an object placed on the terrain surface.

6.3.1.2.3 SIMULATED ENTITIES

Entities, whether vehicles, individuals, or combinations are represented by a combination of data structures and routines. Information concerning the state of an entity is kept in its control block, as are the addresses of specific routines such as the update routine or the message servicing routine for the entity type.

6.3.1.2.3.1 SIMULATING VEHICLES

A simplified vehicle dynamics model has been implemented which provides mechanisms to move a vehicle about with enough fidelity to avoid serious breaches of reasonability.

Acceleration and deceleration is linear. Aerodynamic resistance is ignored. Braking to a stop is performed with a certain amount of cheating to avoid problems arising from variable update rates. A vehicle's speed does not change in a turn and, in fact, a vehicle may turn at any radius at any speed. Currently, vehicles may not be directed to a specific location in reverse, although they can be made to back up and to turn while backing.

A vehicle may be given a desired acceleration rate to be used until specifically modified. When the vehicle is then given a desired speed it will begin to accelerate or decelerate at the specified acceleration rate until it has reached the desired speed, which it will then maintain. Turns may be made at zero speed by pivoting.

6.3.1.2.3.2 SIMULATING INDIVIDUALS

The system has been developed to allow entities other than vehicles to be simulated but current emphasis has been on implementation of a tank and its driver. The primary maneuver command for a tank driver may be expressed as:

> GO TO, AND PASS THROUGH, LOCATION (X,Y) AT 'S' METERS PER SECOND

A special case of this uses 'S' equal to zero. This is equivalent to saying:

> GO TO, AND STOP AT, LOCATION (X,Y)

The required maneuvering is currently generated by a state machine that turns the vehicle toward the destination, accelerates or decelerates as required, and makes continual course and speed corrections until the conditions in the command have been satisfied.

In addition to the physical operations involved in vehicle control, a driver has a number of higher order functions. Obstacle Detection/Avoidance and Avoidance of Detection by others must be considered in short term route planning. Vehicle Status monitoring and some other tasks must be performed continuously. Some tasks will be performed as responses to specific events.

Other simulated entities, such as the commander of a tank may be implemented. Functions performed by a commander could include accepting a mission assignment, planning a mission, determining a route for travel, issuing orders to subunits, monitoring the environment, directing weapons usage, etc.

6.3.1.2.3.3 SIMULATING SITUATIONAL AWARENESS

For an entity to generate behavior appropriate to its circumstances it needs:

- information that describes the current state of its environment
- a means to organize and interpret that information to determine the "current situation"
- a body of knowledge concerning what to do in different situations.
- the capability to perform certain actions to effect the actions required
- a means to keep track of its own actions, states, and processes

Some categories of behavior considered in this project include maneuvering through terrain, avoidance or engagement of other entities, employment of weapons, and communications.

Paths that bypass obstacles such as buildings, areas of steep slope, or rivers, may be generated by a number of methods, usually involving the determination of all possible path segments and a determination of the lowest cost path using graph search methods [18][28][29][31]. Calculation of headings and speeds required to achieve a position relative to other moving entities may be done arithmetically. Determination of

the boundaries of regions shielded from sight by intervening obstacles may be performed using geometric methods.

All of these problems may be solved in different ways using algorithms suited to the various purposes. However, it may be quite difficult to relate the courses available that avoid collision with a building to regions where the chances of being detected visually are minimized, all the while attempting to make progress along a thrust line.

At this stage in the development of the project it appears that one of the most complicated aspects of the human cognitive processes that must be simulated involves the fusion of different kinds of environmental information into one cohesive internal model that can be used for a number of purposes, such as:

- Exploitation of Terrain Features for Route Planning and Obstacle Avoidance
- Determination of regions providing cover and/or concealment
- Monitoring and tracking threat entities
- Avoidance of collisions with other moving objects, especially when maneuvering in formation
- Determination of visibility of regions and entities

Models of subsets of the environment may be constructed based on any number of different data structures and algorithms, but advantages are to be gained by adopting a unifying structure to accommodate as many of the above mentioned purposes as possible.

If a single structure can accommodate information about the terrain and the entities located on it, then relationships between their positions may be more easily determined.

One approach currently being investigated constructs a **local map** or "region of cognizance" in memory to represent a geographical subset of the arena appropriate to the entity's situation. Information concerning terrain elevation and slope and the overlying features is obtained from the TERRAIN DATABASE MANAGER. Information concerning the locations and accelerations of entities, both friendly and threat, is obtained from the MOVING OBJECTS MANAGER and from other sources such as intelligence messages and sighting reports.

A **local map** may be constructed when needed, covering a specific area with the required level of detail and granularity. For initial mission planning, a large geographical area might be modeled with a coarse granularity using all the information available at the time relating to threat locations and desired destinations.

Portions of a pre-planned route might be periodically reevaluated during the execution of a mission or whenever new or more detailed information is made available concerning threats and sightings.

At the small end of the scale, frequent analyses could be made of the region directly surrounding a maneuvering entity in order that courses and speeds may be calculated to avoid obstacles, to arrive at the correct location in a formation, to avoid collisions, or to acquire and engage targets.

6.3.1.2.3.4 USE OF A GRID AS A LOCAL MAP

Preliminary work has used a local map implemented as a two dimensional grid composed of square regions of the terrain. Each square region is considered homogeneous and is represented by a data structure describing its characteristics, such as:

- Whether or not the area is passable to the entity
- Whether or not the area is visible to the entity or to others
- Other costs of traversing the area, such as fuel or wear values
- Whether the area contains a threat, and if so, its type

Before using the local map, its grid is populated with information from the Terrain Database and other sources. Once a grid is built, a number of operations may be performed on it to extract information.

- Paths may be derived which miss obstacles. The Lee-Moore algorithm [3][19] which uses an expanding wavefront function with selective backtracking can find paths through mazelike regions. With extensions, this has the capacity to generate paths that avoid collisions with multiple moving objects.

- Intersections of courses with threats, obstacles, and other entities may be extracted by "drawing" a line across the grid and reading the contents of the grid squares that are intersected.

- Costs of traversal of paths may be calculated by assigning weights to grid locations based on factors such as nearness of threats, screening by squares representing cover or concealment, and terrain surface type such as roadway or mud.

- Alternative paths may be generated by masking of regions or masking of certain grid information on all grid locations.

Paths are currently generated which miss obstacles. These are returned as linked lists of locations. These locations are supplied to the driver's maneuvering mechanisms and a successful traversal of terrain results. The route finding/following mechanism as currently implemented for a tank driver may be expressed as:

> FIND AND FOLLOW AN OBSTACLE AVOIDING ROUTE FROM YOUR CURRENT LOCATION TO LOCATION (X,Y) AND STOP AT THAT NEW (X,Y).

6.3.1.2.4 UTILITIES

6.3.1.2.4.1 LINKED-LIST TOOLS

The message services, timers, terrain database management, terrain reasoning and other modules all make extensive use of queues. A utility providing a variety of capabilities based on a doubly-linked list is included. Queues may be created and initialized. Items may be appended, prepended, or inserted into queues. Items may also be inserted in a sorted order and items may be located by value.

6.3.1.2.4.2 ARCHITECTURE SPECIFIC TOOLS

A number of utility routines are provided to perform data conversions. These are used to swap bytes for CPU's which use different ordering schemes and to generate rotation matrices and to extract angles from these matrices in order to be compatible with SIMNET Version 6.

6.3.1.2.4.3 MATHEMATICAL AND GEOMETRIC TOOLS

A number of tools used in geometrical and numeric operations and debugging are grouped together. They include functions which:

- determine angular differences between pairs of bearings
- dump various messages and structures to the screen for debugging
- build local maps using data from a terrain database and the MOVING OBJECTS MANAGER

- use local maps to locate routes and paths through obstacles

- perform matrix operations used in computation of attitude and location in 3-space.

6.3.2 OFFLINE SOFTWARE

Functions performed "offline" in support of the CGF fall into two primary categories:

1) Behavioral assemblages must be:
   - written
   - assigned to specific entity types
   - tested and validated
   - modified

2) A starting scenario must be defined for a simulation exercise. This may include:
   - specification of a terrain database
   - creation and placement of players at initial locations
   - assignment of players to forces and operational units
   - promulgation of initial operational orders and mission assignments

The implementation of these functions is intimately related to the structure of the online software. Behavioral databases must be created in formats usable by the entities at runtime. Scenarios must be specified in a manner allowing runtime software to dynamically load terrain databases and create entities.

6.3.2.1 GATHERING KNOWLEDGE FOR THE SYSTEM

As mentioned earlier, simulated entities will generate their "behavior" using a variety of mechanisms which may include:

- Algorithmic solutions
- Finite State Machines
- Production Systems
- Inference Engines
- Lists and Tables

Information from **Domain Experts** is usually required in order to construct behavioral models which exhibit reasonable levels of credibility. However, experts in tactical areas are rarely programmers and the traditional means of extracting information has been through use of a **Knowledge Engineer**. This is an individual with intimate

knowledge of the details of the simulation process who interviews the Domain Expert and generates databases to drive the simulation. This has proved to be time consuming and difficult and is one of the worst bottlenecks in the knowledge transfer process.

This project has investigated several ideas for the automation of part or all of the tasks performed by a Knowledge Engineer. Ideally, a system would be developed that would prompt a Domain Expert to provide the information appropriate and necessary to the behavior to be simulated and would convert that information to a form that could be utilized efficiently by the simulation. This does not appear to be practical for the areas best suited to efficient, algorithmic solutions. However, several approaches to the other areas are discussed below.

6.3.2.1.1 FINITE STATE MACHINES

A **Finite State Machine** (FSM), in the context of this investigation, refers to a collection of the states or conditions that an entity might be in at any time, the actions permitted or required while in each state, and the criteria for transition from one state to another.

An FSM may be represented by a series of statements, by a multidimensional table, or by a graphical means such as a diagram that uses:

- Named circles (bubbles) to represent states
- Directed lines (arrows) between states to indicate permissible transitions
- Rectangular blocks on the transition lines holding text to describe the criteria for transition
- Text within the named circles to describe the action(s) to be taken while in that state

A graphical approach would appear to be most suitable for non-programmers. An interface could be developed which would allow a domain expert to define the overall structure of the **state diagram** through the use of a **Graphics Editor** using a menu and a pointing device such as a mouse. This might include placing the state "bubbles" and drawing the transition lines.

Information describing the transition criteria and the state specific actions could be selected from a menu or could be entered as text. A system similar to this has been developed by Harris Corporation [24]. IST is currently examining this product, called AKATS, and is investigating ways to acquire it for integration and testing.

6.3.2.1.2 PRODUCTION RULE LISTS

One area of investigation has resulted in the development at IST of a tool to convert "English-like" rules into executable code.

Lists of **If..Then..** type statements, called **Production Rules**, are generated by a Domain Expert using a text editor. These lists are submitted to a **Preprocessor** which generates a file consisting of a series of C language subroutines and data structures relating them. The preprocessor's output format is compatible with an IST developed **Production Rule Interpreter** which efficiently executes these compiled lists of rules at runtime.

For a Domain Expert to be able to generate these production rules there must be a defined set of testable conditions from which the "if" portions of the rules are constructed, and a defined set of executable functions from which the "then" consequents are built. In order that the process may be convenient and efficient, some method must be available to present these options to the individual as he writes the rules. A multi-window interface would seem to be appropriate here, with testable items listed in one window, executable functions in another, and the set of rules being edited displayed in a third.

To ease the burden on the Domain Expert, a template should be provided to the user whenever he wishes to generate a new rule. The text editor should provide the expert the capability to modify or delete rules, save rulebases in files, merge files, etc.

6.3.2.1.3 INFERENCE ENGINES

An **Inference Engine** is a software facility for making judgements based on the presence or absence of "facts" and on rules of implication. A number of systems, such as CLIPS and NEXPERT are commercially available for generating and executing databases constructed for this purpose. IST is currently using CLIPS in an attempt to derive some generalized rules for tactical movement.

6.3.2.1.4 LISTS AND TABLES

Some aspects of CGF behavior involving the generation of sequential actions are suited to the use of data structures such as lists and tables. Mission planning and route specification are especially dependent upon the generation of

lists of actions which must take place serially or in parallel.

IST is developing the concept of **"Mission Scripts"** which may be implemented as groups of lists containing information about actions to be taken, areas to be traversed, or messages to be sent, etc. These lists may be statically defined, as for a standard mission such as a road march, but at times, dynamic analysis of situations may require modification or replacement of lists as conditions mandate changes in plans, routes, or responses to events.

One factor common to all of these knowledge acquisition and representation packages is that each must be tailored to the specific environment toward which the knowledge is targeted. Currently, no available system can accommodate all of the types of knowledge required by a CGF, although an attempt to integrate a number of approaches has been made by BBN in the development of KREME [1]. More work must be done defining the different types of required behavior before benefits will be gained by development of specific tools to create these different types of behavioral databases.

6.3.2.2 SCENARIO GENERATION

There are two reasons why an automated method for scenario creation is necessary. They are **complexity** and **repeatability.**

Creation and placement of individual entities within an arena involves making a large number of choices. Entities must:

- be selected for their type
- be placed reasonably on the terrain
- be assigned to operational units
- be assigned behavioral databases
- be assigned initial missions, standing orders, etc.

For training purposes as well as for debugging and validation, it is important to be able to recreate specific conditions so that behavioral sequences may be observed more than once, or from different vantage points.

A mechanism contributing to the satisfaction of both requirements might be implemented using a **scenario file** which could be built using an offline mechanism and would be read by the simulation at the beginning of execution.

Such a mechanism could use a series of screens and menus to guide the user through the creation of the scenario.

The user would be asked what type of entity he wished to create, where to locate the entity on the terrain, (x, y, heading, etc.) and possibly what production rule databases or state machine definitions to use for the behavior of that entity if a non-standard entity was being assigned. The user might also be asked to enter a mission plan for an entity or force. This could be done with an editor which would guide the user in filling in specific items needed for a mission plan.

A useful feature would be the ability to save a scenario under a unique name so that it might be reused. Another would be a feature to automatically generate and place units consisting of multiple entities.

6.3.2.3 CONTROLLING THE DEVELOPMENT PROCESS

Development of behavioral assemblages will likely require a Domain Expert to specify behavioral databases, test them by observing running simulations, and modify them as required. This cycle could be streamlined by use of a **Development Executive** which would allow the expert to specify whether he wanted to generate databases, assign them to entities, develop or modify a scenario, or select a scenario and invoke its simulation. Features could be developed such as the ability to "checkpoint" a running simulation, halt it, and restart the simulation at a named checkpoint. This would permit comparision of behavior resulting from different versions of a behavioral database.

6.3.2.4 INTERFACES

Standard and user friendly interfaces are desirable for all of the functions described above in order to remove the requirement for a user to learn multiple protocols. The interface could consist of a series of screens and menus. This interface would allow the calling of routines from a menu item to accomplish some task, such as file creation or field validation. With a totally menu driven system, all input from the user could be validated and unreasonable choices excluded.

The use of a mouse is anticipated so that the user will have only to point to a menu item and click it to activate that item. The operator would also benefit from inclusion of **"HELP"** screens. These text screens would be accessed via HELP menu items and would explain specific aspects of the system.

7 CURRENT CAPABILITIES AND LIMITATIONS

The testbed currently permits simulation of multiple vehicles but generation of more than one vehicle per VECTRA requires a slower than optimal update rate. This produces somewhat jerky movement. The highest possible update rate is determined by the number of entities being generated, the type of behavior specified for each, and the number of remotely generated entities being tracked by the MOVING OBJECTS MANAGER.

The behavior currently implemented simulates a driver's actions in driving across terrain, turning, accelerating, and braking to reach a specified location while avoiding collision with static obstacles. This component will be used as a building block by higher level modules such as mission executors.

Entities are created and placed via console input on the ground, at a given (x,y). This is an interim solution which will not be necessary once scenario generation is implemented.

Commands currently implemented include the following:

DISPLAY MANAGER
- Center the Display Offset at (x,y)
- Set the Display Scale to N meters per pixel
- Toggle the display of terrain polygons

UPDATE MANAGER
- Set the update rate for entities (vehicle dynamics) to N hundredths of a second

MOVING OBJECTS MANAGER
- Set the update rate used by the MOVING OBJECTS MANAGER for Dead-Reckoning to N hundredths of a second
  Toggle the display of the MOVING OBJECTS MANAGER tracking data

INITIALIZATION MANAGER
- Create and place a vehicle at (x,y) with given heading

INDIVIDUAL VEHICLES
- Command a vehicle to accelerate/decelerate to N meters per second
- Command a vehicle to use N meters per second per second as standard acceleration rate until further notice
- Command a vehicle to turn right or left
  Command a vehicle to go to (x,y) and stop
- Command a vehicle to go to and pass through (x,y) at N meters per second
- Command a vehicle to set its main gun elevation to N radians
- Command a vehicle to slew its turret azimuth to N radians relative to vehicle heading

- Command a vehicle to develop and follow an obstacle avoiding path to (x,y) and stop at that location

BIBLIOGRAPHY

1. Abrett, G. and Burstein M., "The KREME Knowledge Editing Environment (Revised)", Knowledge Based Systems Vol. 2, BBN Systems and Technologies Corporation Advanced Simulation Division, Academic Press Limited (October 1988).

2. Bailey, M. and Companion, M., State-of-the-Art Assessment for Simulated Forces, University of Central Florida Institute for Simulation and Training, Orlando, FL (November 22, 1989).

3. Bannister, D. and Mair, J., The Evaluation of Personal Constructs, Academic Press Inc., New York, NY (1968).

4. Bannister, D. (ed), Perspectives in Personal Construct Theory, Academic Press Inc., New York, NY (1970).

5. Brooks, R., Buchanan, B., Lenat, D., McKeown Jr., D., and Fletcher, J., Panel Review of the Semi-Automated Forces, Institute for Defense Analyses, IDA Document D-661 (September 1989).

6. Cadiz, J., Ouyang, R., and Thompson, J., Interfacing of the Silicon Graphics Networkable Flight Simulator with SIMNET, University of Central Florida Institute for Simulation and Training, Technical Report IST-TR-89-1 (October 1989).

7. Cadiz, J., Ouyang, R., and Thompson, J., The Simnet Rotation Matrix, University of Central Florida Institute for Simulation and Training, Technical Report IST-TR-89-4 (October 1989).

8. Ceranowicz, A., Downes-Martin, S., and Saffi M., Simnet Semi-Automated Force Version 3.0: A Functional Description (Revised), BBN Systems and Technologies Corporation Advanced Simulation Division, Report No. 6939 (October 1988).

9. Danisas, K., Smith, S. and Wood D., Sequencer/Executive for Modular Simulator Design, University of Central Florida Institute for Simulation and Training, Technical Report IST TR-90-1 (January 1990).

10. Downes-Martin, S., Replacing the Exercise Controller with the Enemy: The Simnet Semi-Automated Forces Approach, BBN Systems and Technologies Corporation Advanced Simulation Division, Report No. 7211 (December 1989).

11. Intelligent Simulated Forces: Evaluation and Exploration of Computational and Hardware Strategies, University of Central Florida Institute For Simulation and Training, Work Plan Report #2 CDRL A002 Contract N61339-89-C-0044 (November 13, 1989).

12. Lang, E. and Wever, P., SDIS Version 3.0 User's Guide: Interchange Specification, Class Definitions, Application Programmer's Interface, BBN Systems and Technologies Corporation Advanced Simulation Division (August 1990).

13. Lee, C., An Algorithm for Path Connection and Its Applications, IRE Transactions on Electronic Computing, Sept 1961, pp. 346-365.

14. Literature Review for Intelligent Simulated Forces, University of Central Florida Institute for Simulation and Training (October 1989).

15. Maneuvering Board Manual, Hydrographic Office Publication No. 217, U.S. Naval Oceanographic Office (1963).

16. Marcus, S. (ed), Automating Knowledge Acquisition for Expert Systems, Kluwer Academic Publishers, Norwell, MA (1988).

17. Military Standard (Draft) Protocol Data Units for Distributed Interactive Simulation, University of Central Florida Institute for Simulation and Training, Procedures Document IST-PD-90-2 (June 1990).

18. Mitchell, J., "An Algorithmic Approach to Some Problems in Terrain Navigation", Artificial Intelligence v37, December 1988, pp. 171-201.

19. Moore, E., "The Shortest Path Through a Maze", Proceedings of the International Symposium on Switching Theory, Harvard University Press, Vol. 1, 1959, pp. 285-292.

20. Novak, J. and Gowin D., Learning How to Learn, Cambridge University Press, New York, NY (1984).

21. Peterson, J., Petri Net Theory and the Modeling of Systems, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1981).

22. Pope, A., The Simnet Network and Protocols, BBN Systems and Technologies Corporation Advanced Simulation Division, Report No. 7102 (July 1989).

23. Rationale Document Protocol Data Units for Distributed Interactive Simulation, University of Central Florida Institute for Simulation and Training, Procedures Document IST-PD-90-1 (June 1990).

24. Sargeant, J. and Schuerger, J., "A Graphically Oriented Automated Knowledge Acquisition Tool", Proceedings of Third Florida Artificial Intelligence Research Symposium, Harris Corporation, GESD (April 3-6, 1990) 107-111.

25. Simnet Semi-Automated Forces (Version 3.X) Functional Specification, Perceptronics, Technical Report No. PTR-4043-15-0200-4/90 (April 1990).

26. Stahl, H., Ford, K., Adams-Webber J., and Novak, J., "ICONKAT: Integrated Constructivist Knowledge Acquisition Tool", Proceedings of Third Florida Artificial Intelligence Research Symposium (April 3-6, 1990) 107 - 111.

27. Stanzione, T., Terrain Reasoning in the SIMNET Semi Automated Forces System, BBN Systems and Technologies Corporation Advanced Simulation Division, Report No. 7140 (October 1989).

28. Thorpe, C. and Gowdy, J., Annotated Maps for Autonomous Land Vehicles, Unpublished, Carnegie Mellon University (1990).

29. Thorpe, C., Path Relaxation: Path Planning for Mobile Robot, Carnegie Mellon University Robotics Institute, Technical Report CMU-RI-TR-84-5 (April 1984).

30. Thorpe, J., The New Technology of large Scale Simulator Networking: Implications for Mastering the Art of Warfighting, Ninth Interservice Industry Training Systems Conference (November 1987).

31. Tarjan, R., Fast Algorithms for Solving Path Problems, JACM 28(3), July, 1981.