


# DISTRIBUTED INTERACTIVE SIMULATION VIRTUAL CASSETTE RECORDER (DIS VCR) A DATALOGGER WITH VARIABLE-SPEED REPLAY

THESIS

Jonathan L. Fortner, Captain, USAF

AFIT/GE/ENG/94D-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/94D-10

# DISTRIBUTED INTERACTIVE SIMULATION VIRTUAL CASSETTE RECORDER (DIS VCR) A DATALOGGER WITH VARIABLE-SPEED REPLAY

THESIS

Jonathan L. Fortner, Captain, USAF

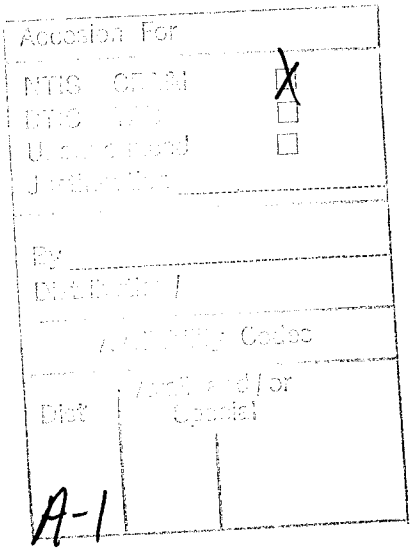
AFIT/GE/ENG/94D-10



Approved for public release; distribution unlimited

*Disclaimer*

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.



AFIT/GE/ENG/94D-10

# DISTRIBUTED INTERACTIVE SIMULATION VIRTUAL CASSETTE RECORDER (DIS VCR) A DATALOGGER WITH VARIABLE-SPEED REPLAY

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Jonathan L. Fortner, B.S.E.E.

Captain, USAF

December 1994

*Acknowledgments*

Thanks to my advisor Captain Keith Shomper for his insightful suggestions that helped me through some of more difficult passages in this thesis. My appreciation goes to all of the graphics students for their helpful instructions on a wide variety of topics. Special thanks to Lieutenant Jimmie Rohrer for his help with producing sound on an Indigo and Captain John Vanderburgh for his expert assistance in understanding performer-based applications.

My deepest appreciation goes to Mr. Steven Sheasby. Without him, this thesis would not have been possible. He provided significant information on the workings of the network and expert insight on the ins-and-outs of Distributed Interactive Simulation (DIS). I was able to adapt much of his source code to the completion of the software portion of this thesis effort.

Thanks to Bruce Clay for helping me understand his network daemons and for the help in understanding DIS audio.

I also appreciate the reviewing contributions of thesis committee members Lieutenant Colonel Martin Stytz and Lieutenant Colonel Patricia Lawlis.

*Table of Contents*

*List of Figures*

Figure Page

*List of Tables*

Table Page

*Abstract*

The overall objective of the Distributed Interactive Simulation Virtual Cassette Recorder (DIS VCR) is to add a flexible replay capability to any DIS environment and specifically to the Remote Debriefing Tool (RDT). The DIS VCR's abilities include selective filtering of incoming DIS Protocol Data Units (PDUs), variable-speed replays, ability to pause, fast-forward, rewind, efficient data storage and retrieval, and an interface that simplifies the execution of those functions. The thesis includes a DIS VCR-compatible design for concurrent replay of audio extracted from signal PDUs and an extension to the replay design that supports unmodifiable rendering or receiving applications. For variable-speed replays, we created a scalable simulation clock and a new PDU (the Replay PDU). Applications modified for replays use the simulation clock to govern their dead reckoning algorithms while the DIS VCR uses it to control the timed release of stored PDUs. The Replay PDU communicates mode and speed changes between the DIS VCR and replay-modified applications. The DIS VCR's full functionality was successfully demonstrated at the 1994 AFA convention.

# DISTRIBUTED INTERACTIVE SIMULATION VIRTUAL CASSETTE RECORDER (DIS VCR) A DATALOGGER WITH VARIABLE-SPEED REPLAY

## *I. Introduction*

### *1.1. Overview*

The past few years have seen remarkable advances in the area of Distributed Interactive Simulation (DIS). The DIS environment allows military units from around the country to participate in mock training exercises without a large commitment of resources (4:1).

Recent developments have demonstrated the capability to "integrate live aircraft into a Distributed Interactive Simulation" (1:5). Gardner (1) developed a Remote Debriefing Tool (RDT) that emulates, and for some functions improves on, the Red Flag Measurement and Debriefing System (RFMDS). The proof-of-concept for RDT demonstrated for the first time that we can capture live range data from a military exercise, transmit it across a wide-area network, and view the mission in real-time using the RDT software. Since the RDT software suite can replay the scenarios from stored RFMDS data, pilots no longer have to land at Nellis AFB to get the benefit of RFMDS in "reconstructing the missions (1:3)," a critical part of the training that helps pilots and crews learn. However, RDT's replay capability lacks the flexibility of RFMDS, and requires too much disk storage. For remote debriefings there is a long arduous process to get the data and set it up at a remote location.

The Advanced Research Projects Agency (ARPA) has continued funding RDT to improve its mirroring of RFMDS capabilities. They envision replay functionality that is very similar to what

every home Video Cassette Recorder (VCR) has. Even Gardner recommends a VCR type application for DIS Protocol Data Units (PDUs) as an "area where additional research is warranted" (1:146).

Another important part of mission reconstruction is the replay of UHF communications. The RFMDS system receives eight UHF channels of Red Flag participant chatter during the exercises and stores it on magnetic tape to replay during mass debriefings. For future research in this area, Gardner says,

> The DIS standard can be employed to transmit and receive digitized voice data. Hardware and software packages exist on the market today which might be adapted for use with RDT. The ability to monitor radio communications would be a significant step forward for the remote debriefing concept. (1:146)

Audio significantly contributes to the realism of RFMDS mission replays, where realism is a key factor in the effectiveness of the debriefing.

In time, simulations will provide a relatively inexpensive, completely realistic virtual environment that can measurably enhance our war-fighting capability. As an example, imagine a purely synthetic battlefield environment that looks and feels like southern Iraq, including representative interactive enemy forces. Now immerse pilots in this artificial battle zone by putting them in virtual cockpits (3) and giving them realistic targets or defensive objectives. After completing their mission, we could replay the scenario and debrief the pilots on their effectiveness. After such training, the actual combat area is more likely to be familiar. This familiarization training could potentially save lives and aircraft that were previously lost because of pilot disorientation.

### *1.2. Thesis Statement*

Develop a flexible replay capability for the RDT. This tool must be designed and implemented independent from RDT, so that any DIS application can benefit from it. Since the DIS standard (2) provides a way to send audio across the network, the software's design can

encompass the capturing and timely replay of that audio. This must be accomplished using the same off-the-shelf hardware and software available to the RDT project.

The DIS Virtual Cassette Recorder (from here on known as the DIS VCR) concept is shown in Figure 1.1.

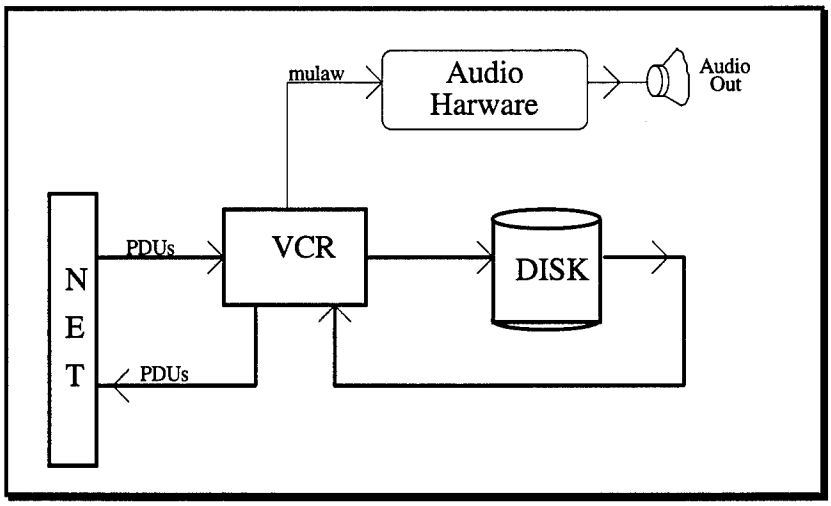



Figure 1.1. DIS VCR Basic Concept

A successful DIS VCR design and implementation will have capabilities as follows:

1. Capture DIS PDUs from the network and store them on disk.
2. Filter incoming PDUs to isolate a specific entity or range of entities, particular exercise, host, or site IDs, and specified PDU types.
3. Replay recorded PDUs at variable speeds, with the ability to pause during the replay.
4. Fast-forward and rewind through a DIS recording.
5. Provide an interface that reduces a user's learning curve and provides for easy identification and loading of DIS recordings.

### *1.3. Scope*

The scope of this thesis effort will include the research, design, and implementation necessary to deliver a system that selectively records and flexibly replays a limited number of DIS PDU types. The design will be extended to cover the modifications needed to include audio derived from DIS signal PDUs (2:74) .

### *1.4. Assumptions*

This section lists assumptions that were made in determining the full scope of this thesis project.

1. The DIS VCR user could be anyone in the DIS community.
2. The hardware and software development tools needed for the DIS VCR will not exceed that available to RDT (1:8).
3. Existing network daemons will be used.
4. The DIS VCR will be independent of any rendering scheme.
5. PDUs will be replayed in the same order in which they arrived.

### *1.5. Basic Approach*

The DIS VCR can be divided on the top level into two parts; record and play. Record is subdivided into getting filtered PDUs from the network and storing them to disk. Play is more complex in that it must be able to get PDUs from the disk, send them across the network, control the speed at which they are sent, pause and resume during replays, and allow the user to move quickly to some specified time in the data.

*1.5.1. Recording.* This component of the DIS VCR takes PDUs supplied by the network daemons, checks the PDUs to see if they fit the user specified parameters, then stores all matching PDUs to disk. As identified later, PDU storage to disk is the only real difficulty for recording.

*1.5.2. Play.* The network interface for play is nearly the same as record. When a replayed PDU is ready to send, it is handed off to the network daemon. Timing this hand-off is critical and is the main driver of the design for play. The next few sections break up play's functions and describe the basic approach to each.

*1.5.2.1. Speed.* Since DIS is a "real-time" environment, it is not equipped to deal with an other-than-real-time replay of a simulation. Therefore, it is necessary to create a clock that moves at the desired speeds making it easy to time-release PDUs. This design decision carries over to the receiving application since dead-reckoning is normally performed using real-time clocks.

*1.5.2.2. Fast-Forward and Rewind.* Unlike your typical home VCR, the DIS VCR only allows fast-forwarding or rewinding from the stop mode. To add fast-forward or rewind, there must be a way to determine where in the file a specific time falls. The difficulty is that PDUs are stamped with a time offset since the beginning of the hour (2:102); therefore, we need a way of marking time blocks.

*1.5.2.3. Pause.* To pause, the DIS VCR stops sending PDUs at the current point in the data stream. It resumes from there when pause is released. However, the application must not drop active entities due to a lack of PDUs (2:25). This requires modifications to the receiving application or a method for ensuring PDU updates are sent by the VCR at five second intervals (see section 3.7).

*1.5.2.4. File Selection.* For the DIS VCR, we need a convenient way to select a data file and some visual data that identifies which recording is selected and what it contains. In reality, record will need to store the file data that is used to identify it.

## *1.6. Design Philosophy*

Re-inventing the wheel takes considerable time and effort. It is better to make full use of existing code and high-level tools that will significantly increase productivity and allow time to concentrate on ease-of-use and functionality. The design and implementation of the DIS VCR will follow this philosophy.

## *1.7. Thesis Organization*

The remainder of this thesis is divided into five chapters and an appendix. Chapter II covers all relevant background material concerning DIS, simulations, and dataloggers. Chapter III discusses the system design and breaks it down into the components that make up the DIS VCR. The implementation of the design is found in Chapter IV which contains the details on how the solutions to replay were coded. In Chapter V, performance data and its relevance is discussed. Finally, Chapter VI summarizes this document, identifies the contributions of this work, and recommends future enhancements and research. To complete the thesis, a DIS VCR user's manual is included in Appendix A.

# *II. Background*

## *2.1. Introduction*

"Distributed Interactive Simulation (DIS) is a time and space coherent synthetic representation of world environments designed for linking the interactive, free play activities of people in operational exercises" (2:1). This chapter covers DIS concepts that directly apply to this thesis, applicable elements of the Remote Debriefing Tool (RDT), and current work on dataloggers.

## *2.2. Distributed Interactive Simulation*

The DIS standard (2) provides a way for geographically separated simulators to interact with each other via network communications. Distributed means that no single computer controls the simulation. Each locality is responsible for the rendering of entities based on incoming Protocol Data Units (PDUs) and local copies of common terrain and models (e.g. tanks, fighters and naval vessels). Local applications must send entity update PDUs when dead-reckoning thresholds are exceeded or five seconds have elapsed (1:14-15; 6:13-15).

The DIS standard defines 26 PDU types, but most applications only use a few. The initial design and implementation of the DIS VCR will support the entity state, fire, detonation, remove entity, and emissions PDUs. The following sections describe the PDU structures including the signal PDU that will be used for the audio design.

2.2.1. *A Common PDU Header.* The fields that are common to all PDUs are depicted in Figure 2.1. The first field identifies the DIS version the application is using. The DIS VCR uses version 2.0.3 and only captures PDUs with a three in this field. The exercise ID is used to identify the PDUs of participants from the same simulation. The participants agree in advance on the value

| | |
|---|---|
| PROTOCOL VERSION | 8-bit enumeration |
| EXERCISE ID | 8-bit unsigned integer |
| PDU TYPE | 8-bit enumeration |
| PROTOCOL FAMILY | 8-bit enumeration |
| TIME STAMP | 32-bit unsigned integer |
| LENGTH | 16-bit unsigned integer |
| PADDING | 16 bits unused |

Figure 2.1. Common PDU Header Fields

for this number. DIS applications must be able to filter out PDUs which do not have the proper exercise ID. The next two fields, PDU type and protocol family, identify the portion of the PDU following the header. They can be used to select only certain PDU types for a particular recording or replay. The time stamp holds the time past the hour when the data in the PDU is valid (2:102). It is represented as the number of DIS time units, roughly equal to 1.676 microseconds each, since the beginning of the hour. This value is shifted into the upper 31 bits, and must be shifted back before using it. The length field is the total number of bytes for the PDU, including the header. This key field is used in applications to copy the right amount of data to and from the network daemons. The length field is central to the PDU storage and retrieval scheme for the DIS VCR. The last field is padding that allows for future growth.

*2.2.2. Entity State, Fire, and Detonation PDUs.* The entity state, fire, and detonation PDUs are the three most important PDU types for most DIS applications. However, the DIS VCR does not perform any rendering, making the full details of these PDUs unimportant. Nevertheless, the VCR must account for the differences in the PDU lengths shown in Figure 2.2. Although the

| **PDU TYPE** | **PDU LENGTH** |
|---|---|
| Entity State | 144 bytes + articulation parameters |
| Fire | 96 bytes |
| Detonation | 108 bytes |

Figure 2.2. PDU Lengths by Type

layout for each of these PDU types differs significantly, they have identical entity ID fields immediately following the header, as shown in Figure 2.3. For fire and detonation PDUs, these fields identify the firing entity, not the weapon (2:112-114).

| PDU Header | See Fig 2.1 |
|---|---|
| Entity ID | |
| - Site | 16-bit unsigned integer |
| - Application number | 16-bit unsigned integer |
| - Entity | 16-bit unsigned integer |

Figure 2.3. PDU Entity Identification Fields

*2.2.3. The Signal PDU.* This PDU has the same header and entity ID fields as the others. However, the entity ID "may represent just the radio or an entity containing the radio" (2:167). The field mapping for the signal PDU is shown in Figure 2.4.

| FIELD SIZE (bits) | FIELD NAME | DATA TYPE |
|---|---|---|
| 96 | PDU HEADER | See Figure 2.1 |
| 48 | ENTITY ID | See Figure 2.3 |
| 16 | Radio ID | 16-bit unsigned integer |
| 16 | Encoding Scheme | 16-bit enumeration |
| 16 | TDL Type | 16-bit enumeration |
| 32 | Sample Rate | 32-bit integer |
| 16 | Data Length | 16-bit integer |
| 16 | Samples | 16-bit integer |
| 8 | Data #0 | 8-bit unsigned integer |
| | ⋮ | |
| 8 | Data #n | 8-bit unsigned integer |

Figure 2.4. Signal PDU Fields

The DIS standard specifies that "8-bit mulaw encoding, sampled at 8Khz, shall be supported by all DIS radio simulators" (2:74). To simplify the audio design, mulaw encoding is assumed. Therefore, the encoding scheme field is set to the mulaw enumeration.

## *2.3. RDT Versus RFMDS*

In the last chapter, it was pointed out that RDT's replay ability lacked flexibility and was a long, arduous, space hungry process. In this section, that replay process is examined and compared to the RFMDS capability.

*2.3.1. Data Conversion for Remote Sites.* The process of archiving, transferring, and converting an RFMDS recording is illustrated in Figure 2.5. Since live mission data, including audio, is stored to hard disk (4:Sec 3, 8), the first step is to archive it to a 9-track tape for shipment to the debriefing site. At the remote site, the raw data must be transferred to a hard disk using

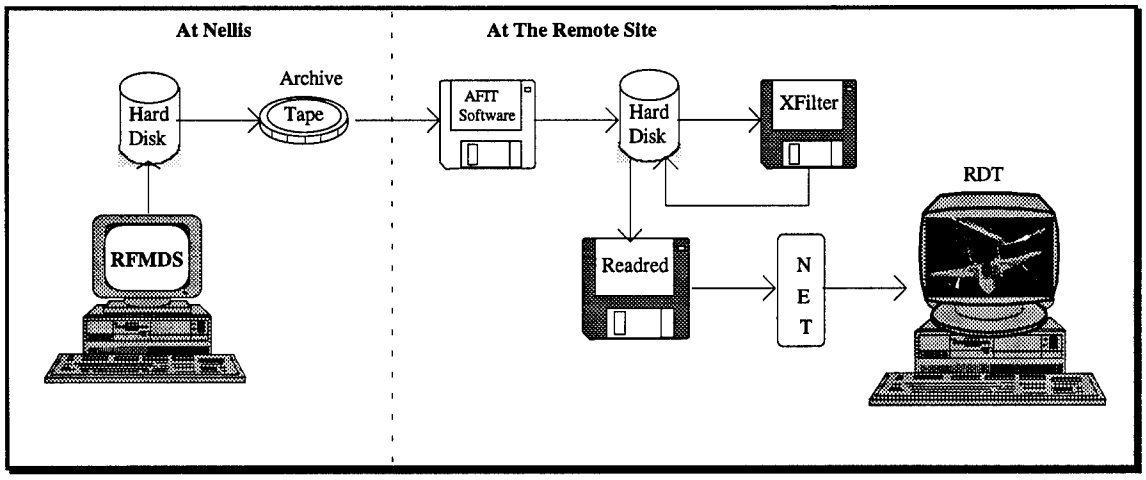



Figure 2.5. Getting RFMDS Data to a Remote Site

special AFIT software. The next step uses the RDT *Xfilter* program to trim off unused RFMDS messages (1:133). The new file can now be used as input for the RDT *readred* software which puts PDUs onto the network.

Table 2.1 shows the disk space requirements for raw and filtered data and lists each form's PDU count. Gardner discovered that he could use DIS dead reckoning (position predicting) algorithms to reduce the *readred* PDU count to 184,078 using a high-fidelity movement threshold of one meter and to 46,596 using a 10 meter threshold (1:18,133). He also noted that 38% of

Table 2.1. RFMDS Converted Data Sizes and PDU Counts

| LENGTH | RAW | XFILTERED | RAW PDU COUNT | 1 METER DEAD-RECK PDU COUNT | 10 METER DEAD-RECK PDU COUNT |
|---|---|---|---|---|---|
| 44.65 mins | 184MB | 123MB | 797,182 | 184,078 | 46,596 |

the CPU time was spent converting RFMDS floating point to the IEEE format used in DIS (1:94). The overhead severely limited *readred*'s maximum speed and caused it to lag behind range time on occasion. Our design for the DIS VCR reduces the disk space requirements for remote debriefing sites and significantly increases the speed and functionality for replays.

*2.3.2. Readred's Replay.* RDT tests with an RFMDS user revealed that the two most significant capabilities missing from RDT were UHF communications and flexible replay. Even Gardner states that "a variable-speed replay and pause capability are the backbone of the debriefings that follow the Red Flag missions" (1:141-143).

*Readred* replay was designed to run at one speed from start to finish. If stopped, it has to be restarted from the beginning. Mission analysts need the ability to slow the action down and pause at critical moments to provide an effective analysis. Fully functional replay software, combined with RDT's capabilities, will provide the necessary tools for a thorough mission evaluation.

## 2.4. *Datalogger Functions, Formats, and Uses*

Since the start of this thesis effort, many papers have appeared on dataloggers, their design, and their uses. This section describes the most commonly suggested datalogger functions and compares them to our concept of a DIS VCR.

*2.4.1. Datalogger Uses.* In some forms of training, it is more desirable to place the trainee into mission scenarios that can be repeated exactly, rather than injecting the interactive variability of DIS simulations. In any simulation, reactions of participants can vary, making accurate measurements of performance impossible. Guckenberger, and others, suggest using a "logger-playback module" to set up specific training scenarios. A single person could create the missions by recording the actions of one simulator, playing that logged simulation back while running another vehicle simulation that interacts with the first, and recording the two together. Each iteration adds another entity until the desired result is obtained for the training event. "The logger [or a DIS VCR] may be superior for simple interactive scripted training tasks" (7:263). More interesting is Guckenberger's suggestion to "convert JSTARS or AWACS data directly to a DIS logger file format." Analysts can then use a fast-scan replay mode and watch a "God's eye view of the battlefield" to detect patterns in enemy movements (7:267). Upon completion, the DIS VCR and RDT will provide the fast-scan mode and view for this purpose.

Burke and Lyou are interested in using a DIS VCR-like tool "to evaluate the network component's implementation of the DIS protocol." They want to know if applications are correctly following the DIS standard and if data integrity is maintained across the network (8:D58-D59). They want to isolate low-level network interface problems like corrupt data packets and perform high-level testing on applications for DIS compliance. They even suggest adding a packet editor so they can selectively alter the data to test application tolerances to certain errors.

The main use of the DIS VCR is remote debriefings. The logged Red Flag exercise will be replayed while mission situations are analyzed using RDT as explained in section 1.1.

*2.4.2. Datalogger Functions.* This section lists and describes all datalogger functions that are needed in RDT or were suggested by others. Table 2.2 summarizes the DIS VCR functions for RDT, as described in section 1.5.

Table 2.2. DIS VCR Functions for RDT

| FUNCTION | PURPOSE |
|---|---|
| Record | Capture DIS PDUs to disk |
| Record Filtering | Capture only PDUs that match certain parameters (e.g. Entity ID, Exercise ID, Site ID, ...) |
| Replay | See individual functions below |
| Pause | Stop action to analyze particular moment |
| Fast Replay | Quickly scan events to reach particular actions/events |
| Fast Forward & Rewind | Quickly move to a moment in time |
| Slow Replay | Slow the action down to observe frame-by-frame developments |
| Play Filter | Replay a on different exercise ID than recorded |
| Keep Header Data | Identify recordings. |

Guckenberger, and others, want a datalogger to detect the kill of a scripted target, then filter out any further PDUs from that entity or play a destroyed entity PDU on each update (7). This could be done while recording or replaying and would require much overhead in maintaining a state-of-the-world structure. This function is outside the scope of dataloggers and would be a separate application that can read the datalogger file format (see formats in section 2.4.3) and handle PDUs for probable kills.

Burke and Lyou want the same functions for logging, storage, playback, and flexible data subset selection as those for the DIS VCR (8:D-59). However, they also want a single-step mode that allows detailed examination of individual DIS packets for form and content and a way to measure the packet rates. They also said it is necessary to filter the replay output of PDUs to

allow analyses on specific sets of the logged data (8:D-60). Finally, they recommend packet editing tools that allow modification of the exercise ID, entity ID, or the introduction of specific errors in PDUs for application testing. Most of these new functions could easily be added to the DIS VCR design and playback filtering could be done using two VCRs. Although DIS VCR functions do not allow entity ID modifications, they do provide for exercise ID changes. Moreover, Burke and Lyou do not specify that the tools be built into one piece of software, and AFIT has other tools that can examine PDU content for data integrity.

The DIS VCR is designed with the functions needed for remote debriefings and could be adapted for other uses. Table 2.3 lists the additional functions not covered by the scope of this thesis.

Table 2.3. Datalogger Functions Suggested by Others

| FUNCTION | PURPOSE |
|---|---|
| Kill scripted targets | Realism in training with repeatable scenarios |
| Single-Step Replay | PDU packet analyses |
| PDU Editing | Change IDs or inject errors |
| Display PDU Rates | Network traffic versus data integrity analysis |

It is clear that datalogger designs are driven by the specific needs of the user. Therefore, a common datalogger file format, like those discussed in the next section, will allow DIS users to share data and tools.

*2.4.3. Datalogger Formats.* DIS development is being pursued by various organizations. Many tools for DIS analysis and visualization have been built, but they require the logged data to be in their specific format. To share tools that use PDU logged data, we need to develop a standard way of storing that data. This section examines two different approaches to standardization.

*2.4.3.1. Green's Standard Datalogger Format.* Green states that " a standard header, in conjunction with a standard data analysis file format, would facilitate the sharing of information and tools" (9:546). Table 2.4 contains the fields he recommends, the number of bytes each field uses, and a rationale for the fields. Green's standard header is not the minimum needed, which he suggests by identifying the bold-shaded fields as "mandatory" (9:551). Many of Green's fields are used when logging frames rather than DIS PDUs. Frames are captured to do network analysis or to record non-DIS data; actions which are not applicable to our DIS PDU datalogger.

After the header comes a block he calls the D-Record that contains D-Packets (9:553). Figure 2.6 shows the layout of this structure. The D-Structure allows the programmer to read a block of PDUs rather than one PDU at a time, thereby speeding up disk I/O. The structure could be read into memory, then parsed in a loop to send PDUs. The D-Packet allows for variable length PDUs, stopping when the D-Record is full. A PDU count field should be added to the D-Record header and used to set loop variables. Green suggests a D-Record block size of 240k-bytes which would hold about 1460 entity state PDUs.

| **D-RECORD** | | |
|---|---|---|
| Length | 4 bytes | Total fixed block (D-Record) size |
| Start Time - First D-Packet Time | 8 bytes | UNIX stamp |
| Stop Time - Last D-Packet Time | 8 bytes | UNIX stamp |
| D-Packet 1 | D-Packet Size | Multiple of 8 bytes |
| ⋮ | ⋮ | ⋮ |
| D-Packet n | D-Packet Size | Multiple of 8 bytes |
| Padding | m bytes | To next 8-byte boundary |
| **D-PACKET(s)** | | |
| Previous D-Packet | 4 bytes | File index pointer for reverse play |
| PDU Type | 2 bytes | Enumeration (i.e. DIS, SIMNET) |
| Time Stamp | 8 bytes | UNIX stamp |
| PDU Length | 4 bytes | From the PDU captured |
| The DIS PDU | PDU Length | Last field |
| Padding | Bytes | To an 8-byte boundary |

Figure 2.6. The D Structure for a PDU Block

Table 2.4. Green's Standard Header Format (9:547-551)

| FIELD | BYTES | TYPE | REASON |
|---|---|---|---|
| Logger Name Constant | 16 | Char | To locate lost data files with a find utility |
| Logger ID | 2 | Unsigned short | Identify internal sub-format |
| Logger Revision | 2 | Unsigned short | Identify internal sub-format |
| Logger Site ID | 2 | Unsigned short | To combine files from different sites. |
| Logger Host ID | 2 | Unsigned short | To combine files from different sites. |
| Header Length | 4 | Unsigned short | Header length in 8-byte double words |
| Size of Successive Records | 4 | Unsigned long | Fixed length of D-Records (see discussion) |
| Machine Name | 64 | Char-UNIX style | To combine files from different sites. |
| Logger Machine Name | 96 | Char | To combine files from different sites. |
| Logger Information | 2048 | Future use padding | Might be used later |
| Test ID | 128 | Char | ID which test was logged |
| Trial ID | 128 | Char | ID which trial of the test it was |
| TimeZone - Real World | 8 | UNIX tz structure | These fields identify the actual time of the recording and a different simulation time. |
| Start Date/Time - Real | 8 | UNIX timeval structure | |
| Stop Date/Time - Real | 8 | UNIX timeval structure | |
| TimeZone - Sim World | 8 | UNIX tz structure | |
| Start Date/Time - Sim | 8 | UNIX timeval structure | |
| Stop Date/Time - Sim | 8 | UNIX timeval structure | |
| Data Link Protocol | 1 | Unsigned char | Information about the data for the analyst |
| Network/Transport Protocol | 1 | Unsigned char | No reason given |
| Padding | 2 | Unsigned char | No reason given |
| Fixed Cell Size | 4 | Unsigned long | If using fixed-length cells instead of frames |
| Logger Data Link Address | 64 | Unsigned char | Ethernet address - no reason given |
| Logger Network Address | 64 | Unsigned char | IP address - no reason given |
| Transport Address | 4 | Unsigned long | UDP port - no reason given |
| Padding | 4 | Unsigned char | |
| Original File Name | 256 | Char | Hey it might be useful - no real reason |
| Command Invocation | 256 | Char | ID special settings on logger |
| Name of Terrain File | 256 | Char | So an application can use the same terrain for replay |
| Extra Terrain Information | 256 | Char | No reason given |
| Number of Dropped Packets | 4 | Unsigned long | Used when anomalies show up in analyses. Doesn't mention how to get this data. |
| Number of Collisions | 4 | Unsigned long | |
| Number of Giants | 4 | Unsigned long | |
| Number of Runts | 4 | Unsigned long | |
| Number of Auxiliary Files | 1 | Unsigned char | |
| Padding | 1 | Unsigned char | No reason given |
| Exercise ID | 2 | Signed short | A particular ID or -1 for all exercises |
| Padding | 4 | Unsigned char | No reason given |
| Each Aux. File Code | 1 | Unsigned char | For associated files like audio or the next volume if files are broken up. |
| Each Aux. File Name | 255 | Char | |
| Padding | To 24k | Bytes | Round header size to an 8-byte boundary. |

*2.4.3.2. Juliano's Standard Datalogger Format.* A simpler standard file header is suggested by Juliano and has the following general requirements:

- Portability and compatibility
- Files must be readily identifiable
- Support real-time playback
- Minimize file space
- Assure accurate analysis of DIS-based war fighting

The first one means a standard file format like UNIX or DOS with DIS compatible data values. To make files identifiable, he adds a character string of 256 bytes. Real-time playback is supported using a relative time stamp, which is the elapsed time since receipt of the previous PDU. He also adds date and time fields that would be useful to applications that display models according to seasons or time-of-day. To minimize the file size, variable length records are used to store PDUs. This adds a length field and a back pointer to the previous PDU record for bi-directional replays. Finally, he adds an IP address from the incoming packet to identify which machine the data originated from (10:128). The PDU record format is shown in Table 2.5.

Table 2.5. Juliano's PDU Record Format

| FIELD | DATA TYPE |
|---|---|
| Relative Time Stamp | 32-bits with microsecond resolution |
| Time/Date Stamp | 32-bit Ctime with date and time to the sec |
| Machine Address | 32-bit IP address |
| Previous PDU Length | 32-bit integer, acts as pointer |
| Current PDU Length | 32-bit integer |
| The PDU | Current PDU Length |

For some uses, this may not be enough information; however, it makes Juliano's datalogger independent of the architecture. In the last section, Green's logger stored the entire Ethernet frame and used it for low-level network analysis. Keeping a datalogger at the DIS level (just PDUs) allows network interface improvements independent of any DIS logger application (10:129).

The format of Table 2.5 is not efficient when it comes to disk I/O. It only allows programmers to read one PDU at a time. This means time for disk access is added as overhead for each replayed PDU.

## *2.5. Other-Than-Real-Time Simulations*

"As the breadth of distributed applications expands, there are sufficient reasons to suggest that execution of distributed simulators should be permitted to take place in either a faster than real time or slower than real time mode" (11:223). Miller is not the first to suggest this concept, but was the first to present a partial solution that is compatible with DIS standards.

The key to other-than-real-time simulations is time scaling using a simulation clock that has adjustable speeds. The scale is a ratio of elapsed simulation time over "wall clock time" (11:225). The scale is used in applications to adjust dead-reckoning algorithms and modify when entities should be dropped due to inactivity. The time scale must be communicated to other applications and Miller suggests using a new PDU scale field or a separate simulation management PDU. With the appropriate modifications to applications and a way to notify them of time scale changes, other-than-real-time execution is possible.

## *2.6. Summary*

With an appropriately designed DIS datalogger or DIS VCR, the replay needs of RDT can be satisfied. Files must be readily identifiable and organized so that data can be read in blocks to reduce overhead. Time scaling is a promising technique for controlling replay speed and requires the datalogger to communicate the scale to other applications (especially RDT).

Chapter III presents the design for the DIS VCR and shows how these elements are combined to achieve the objectives set forth in Chapter I.

# *III. Design*

## *3.1. Introduction*

Chapter II shows that there are many ways to approach the design of the DIS VCR. While the DIS VCR has an obligation to support remote debriefing using RDT, the design presented here is independent of RDT which also allows the DIS VCR to be used for many of the tasks discussed in section 2.4.1. This chapter starts by examining the top-level design issues that affected design decisions. Follow-on sections discuss VCR design details and other constraints. Although the thesis scope did not include implementing them, designs to support variable-speed replays in unmodified applications and audio in any replay, are presented in sections 3.7 and 3.8.

## *3.2. Motivation for the Initial DIS VCR Design*

This section lists and describes the design considerations that had a global impact on the DIS VCR design.

*3.2.1. Trade-off Analyses.* There were two approaches considered for implementing a datalogger that had slow and fast replay speeds; one requires the receiving application to be modified, the other does not. In the first approach, the application is modified to replace real-time with scaled-time in dead-reckoning algorithms and methods that determine when entities have dropped out of the simulation. The other method is to have the DIS VCR modify all PDU velocity and acceleration fields to fit the current time scale and ensure PDUs get sent out at proper intervals (see section 3.7). This second approach does not require the application to change, but it has some drawbacks. Applications that perform analyses based on entity velocity and acceleration vectors, would be useless since these values are artificially changed. This approach also adds considerable overhead to the play loop, limiting its maximum speed and entity capacity. For these reasons, we opted for the first approach and decided to modify applications that use the DIS VCR.

*3.2.2. Design Constraints.* Another factor limiting maximum speed and entity capacity, is the overhead associated with storing and retrieving PDU data to and from a hard disk. To be useful, the design must support PDU transmission rates for large numbers of entities. We could read or write one PDU at a time, or use blocks of PDUs to speed up disk I/O. If the block method is chosen, we must ensure the daemon buffers do not overflow while a disk access is occurring; otherwise, PDUs will be lost. Preliminary tests were performed which showed that reading and writing one PDU at a time is a disk I/O bottleneck (see 3.4.3.1); therefore, this method was not selected as the design for our storage and retrieval scheme. Section 3.4.3.2 shows how to prevent PDU losses when reading or writing blocks.

*3.2.3. The DIS VCR in a Network Simulation.* The primary function of the DIS VCR is to record and replay DIS simulations. This section covers its intended use and how these expectations affected our design.

*3.2.3.1. Interaction with RDT or Other Applications.* Sections 1.1 and 2.3, made it clear that the DIS VCR would have to support variable-speed replays and pause for RDT. Unfortunately, DIS applications like RDT were not built to handle pauses or speed variances. The RDT dead-reckoning algorithms assume the simulation is in real-time and have no way of handling variable playback speeds. In slow motion, the application would dead-reckon entities past their actual locations (entities would appear to jump backward on each PDU update) and it would drop entities due to the delay between updates. In fast motion, the real-time dead-reckoning algorithm would lag behind the replay and entities would leap forward on each PDU update. Therefore, it is necessary to modify the receiving application so it understands variable speed replays.

*3.2.3.2. DIS PDU Sources.* The PDU time stamp presents a new set of issues. Old replay software, like RDT's *readred*, may send PDUs with the time stamps based on the original

recording time. Therefore, when capturing PDUs from any exercise, host, or site ID, the time stamps can differ dramatically; even for PDUs received nearly simultaneously. For example, a real-time simulation fills the PDU time stamp with local time while *readred* stamps its PDUs with range time. This creates a problem when time stamps are used to schedule the release of recorded PDUs in the play loop.

One solution is to store a receipt time with every PDU and use it to time-release PDUs. This adds system calls to the overhead associated with every PDU and limits the DIS VCR's recording capacity. On the other hand, if the old replay application sends PDUs a little faster or slower than real-time, we need to use the PDU time stamps to control their release. Since this method would prevent the recording of a real-time interaction with a replayed scenario, the solution that stores the receipt time with each PDU is used in the DIS VCR design.

*3.2.3.3. Recording Demos Track-by-Track.* Setting up a complex DIS simulation demo requires a substantial commitment of personnel and computing resources. However, with a DIS VCR, you could record demos one track at a time. Each track could involve multiple simulators that would interact with a replay of previously recorded tracks. As suggested in the last section, we store the receipt time with each PDU and use it to time-release all recorded PDUs.

*3.2.4. Variable Speed with A Simulation Clock.* Section 2.5 discussed the idea of time scaling to control other-than-real-time simulations. In section 3.2.1 we identified the advantages of scaling time. To carry out this idea, we introduce a variable speed simulation clock. We control our simulation clock with the system clock to maintain an accurate scaling factor. To solve the dead-reckoning problem above, the rendering application and the DIS VCR each use a simulation clock. The time from the simulation clock is integrated into the rendering application's dead-reckoning algorithms to correct the lead-lag errors. To use the same time scale as the DIS VCR,

there must be some way of communicating the current scale value when it changes. Section 3.5.2 describes how this is accomplished.

## *3.3. Simulation Clock Design*

This section discusses the functions of a simulation clock and what is required to make time scaling work.

*3.3.1. Functions a Simulation Clock Must Support.* To support applications that render a scene based on the date, time of year, or time of day, the simulation clock must maintain and utilize date and time fields. As simulation fidelity matures, these will be needed to properly render lighting conditions, seasonal terrain appearance, and weather factors. Since the replay requirement includes variable speeds, the simulation clock has to have a way to scale the speed of the clock. Finally, we need a way to format the time for display in applications or the DIS VCR. Table 3.1 lists and describes the basic simulation clock functions.

Table 3.1. Simulation Clock Functions

| FUNCTION | DESCRIPTION / USAGE |
|---|---|
| Set Simulation Time | Allows the simulation clock to start from any time. |
| Get Simulation Time | Returns the scaled simulation time. If scale is set to 1.0, it acts like a normal clock. |
| Set Simulation Date | Allows applications to use any date within the limits of the operating system. For UNIX, this is 1 Jan. 1970 to the year 2038. |
| Get Simulation Date | Returns the date set with Set Simulation Date. |
| Change Time Scale | Varies the speed of the clock |

*3.3.2. Time Since the Beginning of the First Simulation Day.* To facilitate the design of scaled time, it is best to represent simulation time as time since the beginning of the first simulation day. This provides a way to easily gauge how much time has past between calls to the clock. More importantly, it allows us to set the clock to any date and time, and advance it by scaling the time difference between clock calls. Section 3.5 discusses how scaled time is used.

## *3.4. Recording*

The user typically only gets one shot at recording a complex simulation exercise. This makes it important to design the record function so that no PDUs are missed. Therefore our goal was to shift as much overhead as possible to the replay function. The following sections describe the essentials of the recording design with emphasis on maintaining an efficient record loop.

*3.4.1. Net Interface Design.* To record DIS PDUs, it is necessary to communicate with the network. The DIS VCR uses standard AFIT network daemons to retrieve PDUs. The daemons extract the PDUs from the Ethernet frames and deliver them in buffers. The VCR copies each PDU, stores it, and marks the buffer empty so the daemon can fill it with another PDU.

*3.4.2. File Header Information.* One of the general datalogger requirements from section 2.4.3.2, is that files be readily identifiable. To satisfy that requirement, Juliano (10) used a 256 byte character field. However, the user may fail to enter sufficient information in the field. Also, if the DIS VCR is modified to support multiple versions of the DIS standard, then a separate field must be included so replay can select the appropriate algorithms for that version. We include fields for the user name, exercise ID, a title, DIS version, date, flags for each PDU type captured, and start and stop times.

*3.4.3. The Storage and Retrieval Scheme.* Designing a record function that is fast enough to prevent the loss of PDUs while providing an efficient way for the replay routine to retrieve the data and control it for replay functions, was a challenging task. This section identifies the difficulties and presents solutions to them.

*3.4.3.1. One PDU at a Time Design.* To conserve disk space, the best storage approach is to write PDUs one after the other following the file header. Then on replay, the PDU length field is used to determine the correct amount of data to read from disk and to maintain the file pointer position. However, for a disk with a 12 millisecond access time, an 8 millisecond rotational latency, and a 500k-byte per second transfer rate, the storage of 1000 PDUs takes 20.4 seconds or 20.4 milliseconds each. This rate is clearly unacceptable in our environment where typical PDU rates exceed 500 per second.

*3.4.3.2. The PDU Block Approach.* Handling a single PDU at a time is easier to implement, but the disk bottleneck almost guarantees that data will be lost. A faster approach for I/O is needed. Therefore, we chose to read and write blocks of PDUs. To simplify the block design, each block is structured with a start and stop time, an array of PDU pointers, the PDU array, and a PDU count field. The rest of this section outlines the block I/O approach and presents a comparison of it with the single PDU method.

The DIS VCR marks the daemon buffers empty when they are transferred to the block structure. While the VCR writes a full block to disk, it is possible for all of the daemon buffers to become full, resulting in PDUs loses. To prevent this from happening, two blocks are used in the producer-consumer parallel scheme shown in Figure 3.1. In the producer-consumer relationship, PDUs are inserted into a block until the count reaches a preset maximum. The block is

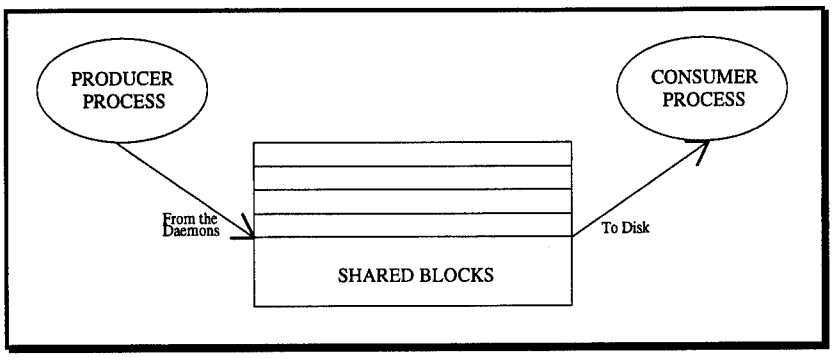



Figure 3.1. Record's Producer-Consumer Parallel Process Threads

marked full (a semaphore flag) and blocks are swapped so one can be written while the other is filled. This significantly increases the number of entities the record loop can handle.

As an example of the efficiency we can achieve with blocked I/O, consider the following calculation. A block of 1000 PDUs is about 150k-bytes and for a disk with a 12 millisecond access time, an 8 millisecond rotational latency, and a 500k-byte per second transfer rate, the write will take 0.32 seconds; 64 times faster than the single PDU approach. Furthermore, without a spawned consumer process, under a load of 1000 PDUs per second, 100 daemon buffers will overflow in 0.1 seconds or less (some buffers may already be filled) resulting in the loss of over 200 PDUs for each write to disk. If the disk is located on the network, instead of residing locally, the losses will increase.

*3.4.4. Filtering the Input.* There are times when the user wants to isolate a specific exercise, entity, or range of entities. Additionally, the user may only want to record PDUs from a particular site or host. In some applications, it is desirable to capture only one or two PDU types. For example, the user may want to see only the fire and detonation pairs or needs to isolate the audio. The DIS VCR design was built to accommodate the filtering of input PDUs based on any of the above criteria.

*3.4.5. Record Pseudo Code.* The record function that takes PDUs from the network and stores them to disk is shown below in pseudo code.

**Pre-processing Steps**

*Spawn the two-thread producer-consumer process*
*Get information for the file header*
*Start the simulation clock*

**Recording Loop**

*While not done*

*Get a PDU from the daemon and store it if keeping*
*Store the receipt time with the PDU and write a value to the pointer array*
*Update the display*
*Mark the daemon buffer empty*
*If the block's count = MaxCount or the block has been filled, mark it full and swap blocks*
*(the consumer detects the full condition and writes the block to disk, then marks it empty)*

*End While*

**End Record Loop**

*Write the last block to disk*
*Post-process the file*

**End Record**

The record loop uses the simulation clock to update the displayed clock so that two formatting routines are not needed; one for record using a real-time clock and another for replay which must use the simulation clock. This also makes it convenient for storing simulation time in file headers and block structures.

## *3.5. Playback*

*3.5.1. Net Interface Design.* To replay DIS PDUs, it is necessary to communicate with the network. The DIS VCR uses standard AFIT network daemons to send the PDUs. The daemons insert the PDUs into the Ethernet frames and deliver them to the network. The VCR copies each PDU from the block structure to a daemon buffer and marks the buffer full so the daemon will send it.

*3.5.2. Replay With a Simulation Clock.* The simulation clock is used to control replay functions by means of scaling. When the scale of the simulation clock changes, that change must be communicated to all modified applications using the DIS VCR for replays. The following sections discuss the mechanism for communicating scale changes and how a change in scale affects each replay function.

*3.5.2.1. A New PDU for Replays.* Since the DIS standard was developed for real-time interactive simulations, it does not provide a means for controlling variable speed replays. Our design introduces a Replay PDU for this purpose. This PDU supports the following actions:

- Switch mode to PLAY, PAUSE, or STOP
- Update the simulation clock time or date
- Change the time scale
- Initialize, start, or stop the replay

Whenever the DIS VCR changes mode, time or speed, it transmits a Replay PDU on the network to report the change to receiving applications. The Replay PDU uses the same header described in section 2.2.1 and the fields necessary to carry-out the actions listed above. These actions are captured in the replay state diagram of Figure 3.2 and are described in subsequent sections. As

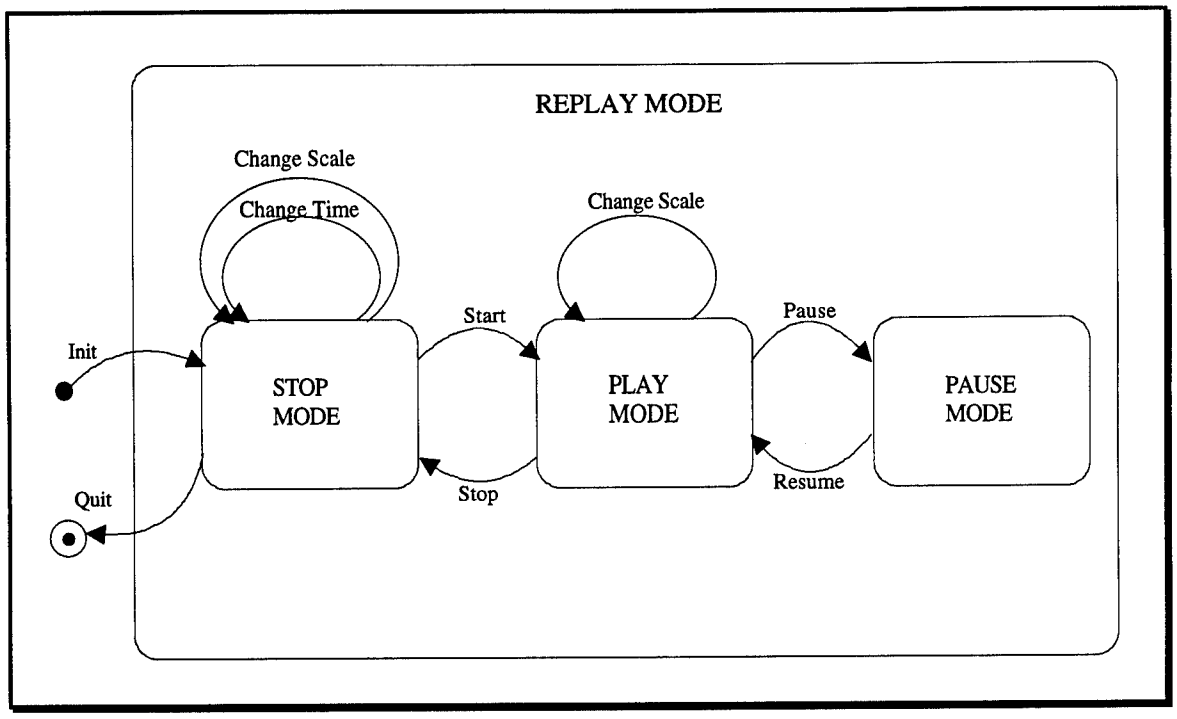



Figure 3.2. Replay State Diagram

depicted, replay initialization is what signals the application to start using its simulation clock instead of the system clock. The terminal condition (a circle with a dot in it) causes the application to revert back to using the system clock.

*3.5.2.2. Variable Speeds.* When the user selects a new speed, the DIS VCR sends out a Replay PDU to change the time scale. This can be done from the STOP and PLAY modes as shown in Figure 3.2. The VCR displays the new speed, sets its simulation clock speed to the new value, and begins releasing PDUs accordingly. The receiving application's dead-reckoning and entity time-out algorithms use the new scale to control their activities.

*3.5.2.3. Pause.* When pause mode is selected, a Reply PDU is sent to signal a mode change. The DIS VCR sets the simulation clock scale to zero which freezes the PDU stream at the

current point in the PDU block. The receiving application is responsible for keeping entities in its current player list alive and in their last position until a resume Replay PDU is received.

*3.5.2.4. Applications and Replays.* Because there is always some network delay before the application receives the Replay PDU that sets the initial clock time or changes the time scale, the DIS VCR replay and the receiving application will be out-of-sync. To illustrate, assume the DIS VCR and modified application are at ten times normal speed when the user decides to change speed to one tenth of standard (slow motion). The VCR sends out a Replay PDU to inform the application of the change, then responds immediately to the change and stretches the time between PDU releases 100-fold. The application continues to run at the faster speed until it gets the Replay PDU, entities leap backward on updates and begin dropping off because hundreds of PDUs were expected and only a few were received. Unfortunately, there does not seem to be a workable solution to this problem.

*3.5.3. Play Loop Pseudo Code.* The play function that takes PDUs from the disk and re-transmits them across the network, is shown below in pseudo code. The user may change speeds and pause while inside play's PDU block loop.

**Pre-processing Steps**

*Select a file*
*Set the simulation clock with file header*
*Spawn the two-thread producer-consumer process that gets data from disk*
*Start the DIS VCR simulation clock*
*Fill and send an initialization Replay PDU*

**End Pre-processing Steps**

**Outer loop**

*While Not stopping and Not end-of-file*
*Hold for full PDU block*

**PDU Block Loop**

*While Not stopping and Not end-of-block or end-of-file*

*If in PLAY mode*

*While PDU time Not equal to simulation clock time*
*Check user input, respond to STOP, PAUSE, and speed changes*
*End While PDU time Not equal to simulation clock time*
*Send current PDU*
*Update display*

*End if PLAY mode*

*Else PAUSED*

*If resume then*
*Send mode-change Replay PDU*
*Restart simulation clock*
*End if resume*

*End Else PAUSED*

*End While Not stopping and Not end-of-block or end-of-file*

**End PDU Block Loop**

*Swap PDU blocks*
*End While Not stopping and Not end-of-file*

**End Outer loop**

**Post-processing Steps**

*Kill the producer process, it is no longer needed*
*Send STOP mode Replay PDU*
*Reset data file*

**End Post-processing steps**

### *3.6. The DIS VCR Interface Design*

A well thought-out human-computer interface can make a substantial difference in a user's learning curve, performance, error rate and satisfaction. The first step in good human-computer interface design is to determine the required functionality. Exactly what are the tasks and sub-tasks that need to be done? The most frequent mistake by new designers is to add too much functionality. This just increases program complexity and clutter which, among other things, increases the learning curve and decreases user satisfaction. The second step is to ensure reliability, security, and integrity (13:57-164). The functionality of the DIS VCR was determined by examining the RFMDS features the VCR is suppose to emulate as discussed in section 2.4.2. The DIS VCR design addresses reliability, security, and integrity by using an interface design that limits many of the errors the user could make.

There are also some underlying design principals to consider for most interactive systems and especially when novice users are involved. First of all, the interface should maintain consistency by using the same prompts, menu styles, and help screens, all of which should keep a similar layout. The interface gives users informative feedback for every user action; even simple tasks get a nominal message that shows completion (13:61). The DIS VCR interface design is structured for a consistent approach to communicating with the user.

To keep from overwhelming the user, we have avoided packing too much information or too many actions into a single screen. The DIS VCR interface design breaks up tasks into several sequenced, well organized screens that keep a similar layout between them. This strategy reduces user search time and increases productivity and task speed (see section 3.6.2).

*3.6.1. DIS VCR Interface Goals.* The main theme for the DIS VCR interface is simplicity. Its basic functions should be as easy to use as the common home VCR that plays VHS tapes. If the user has to remember a lot of command line parameters to perform a simple replay, the interface design is a failure. The following list summarizes the design goals for the DIS VCR:

- Reduce the interface learning curve
- Maximize task speed
- Minimize the user error rate
- Easy access to help
- Easy Recording Identification

*3.6.2. Interface Styles.* Choice of interface interaction components depends on the target user and the goals we set for the interface. The list below, shows some of the primary interaction styles. Table 3.2 lists these style types and describes good and bad aspects of each.

- Menu selection
- Form fill-in
- Command language
- Natural language
- Direct manipulation (pointing device)

Table 3.2. Interface Style Types, Pros and Cons (13:57-60)

| TYPE | ADVANTAGES | DISADVANTAGES |
|---|---|---|
| Menu selection | Shortens learning, structures decisions & easy error handling | Slow, takes screen space |
| Form fill-in | Simplifies data entry, little training needed | Takes screen space |
| Command language | Flexible, appeals to Power users, can use the power of macros | Poor error handling, requires a lot of training and memorization. |
| Natural language | Known syntax | A lot of keystrokes to enter cmds and its behavior is unpredictable. |
| Direct manipulation | Visually presents task concept, easy to learn & retain, high satisfaction | Hard to program & more hardware needed. |

Based on the interface design goals of section 3.6.1 and the advantages listed in Table 3.2, the best combination for the DIS VCR interface is a pseudo-menu selection type form that is operated with direct manipulation and where data is entered via the form fill-in style. The pseudo-menu selection form functions can primarily be represented by buttons that can be pushed with a mouse. As suggested by Foley (17:457) and Shneiderman (13:71), these forms are broken into several sequenced, well organized screens as shown in Figure 3.3. The larger ones overlay the main VCR screen to focus the user's attention to a particular task. The message forms are dialogs that pop-up to give the user relevant feedback like task completion or information on detected errors. What the user manipulates on these screens or forms is the subject of the next few sections.

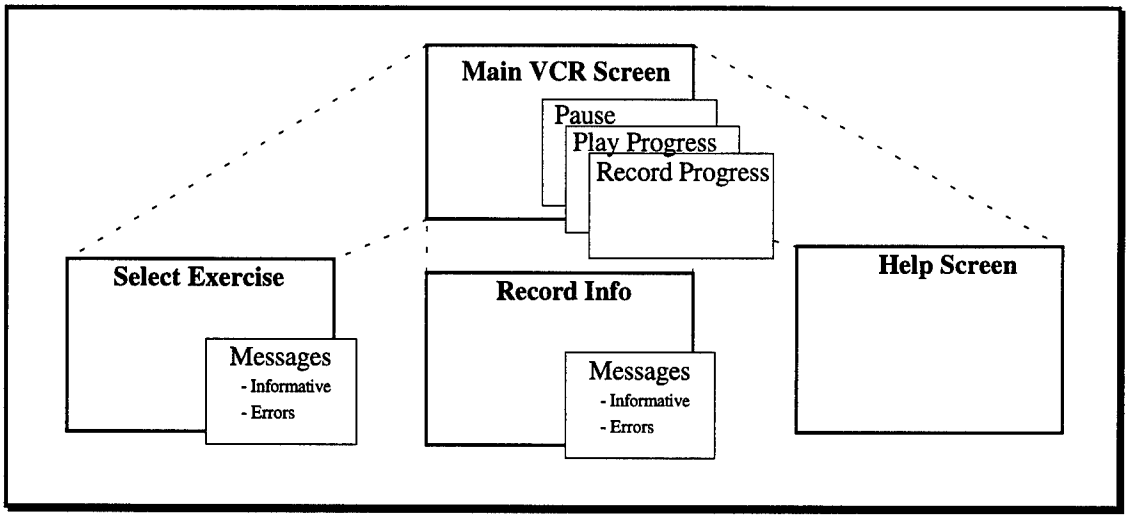



Figure 3.3. DIS VCR Screen Layouts

*3.6.3. Using High-level Interface Building Tools as a Basis.* Table 3.2 points out that direct manipulation is difficult to program. We intend to use high-level interface tools that reduce the difficulties to a manageable level for the DIS VCR interface. Users benefit since high-level tool developers take a standardized, rational approach to screen design that allows applications to acquire a familiar look and feel. These tools usually benefit from years of experience and usability testing (14:3).

*3.6.3.1. A High-Level Interface Tool Set.* The following sections describe elements of high-level interface tools that are used to design the DIS VCR's interface.

*3.6.3.1.1. Basic Tools the VCR Needs.* The building blocks of the interface are 3-D buttons, value adjusters like sliders, counters, input fields and text. These simple objects, shown in Figure 3.4, can be combined to create a powerful, easy to use interface.

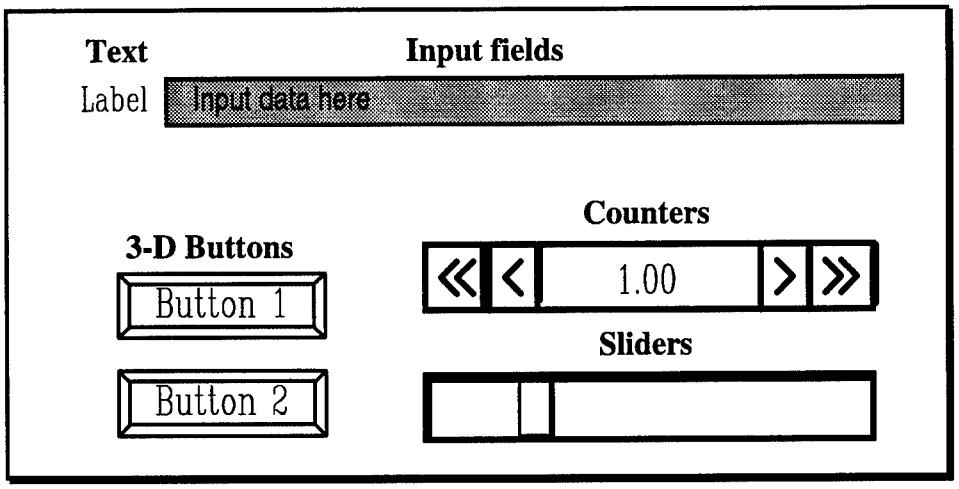



Figure 3.4. Interface Building Blocks

*3.6.3.1.2. Special Objects.* As effective as they are, the building blocks of Figure 3.4 are not enough by themselves for the DIS VCR interface. To reduce the manipulations for specifying which file to replay, a selection pick-list is convenient. With this type of object, the user is presented with a list of the items of interest and only needs to point and click to select one. Figure 3.5 shows a basic pick-list with one item selected. Pick-lists can be programmed to select one or many items and can list the item in any specified order.

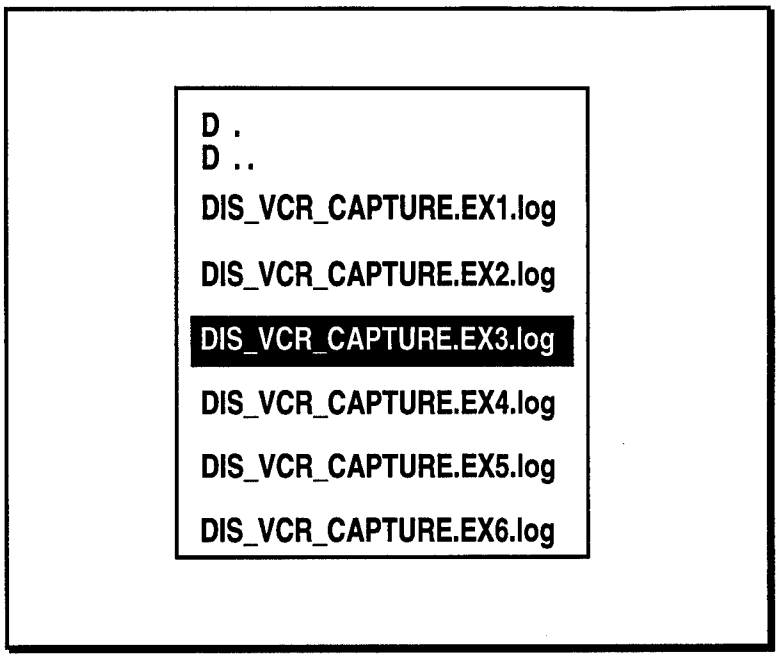



Figure 3.5. A Basic Pick-List

*3.6.3.1.3. Events.* Manipulation of the interface objects should create user events that can be used by the program. These events tell the application which object was used so the application can choose the appropriate action to take. The event detection process must not interrupt the program so that active tasks can continue their functions.

*3.6.4. The Main VCR Form Design.* The main screen must provide access to all of the DIS VCR's functions shown in Figure 3.3. Therefore, it must contain the interface objects needed to start a recording, initiate a replay, or control the current replay. To meet the interface goals of reducing the learning curve and easy access to help, the main DIS VCR screen design should be modeled after a typical home VCR (that interface is familiar to many users) and should provide a single-step access object for help.

*3.6.4.1. Speed Controls.* Standard speed settings give the user a convenient way to choose from among several common choices. This can be accomplished with special objects called

radio buttons. These buttons are grouped together in one cluster where the user can select only one of them. An example of this object group is shown in Figure 3.6. To capture speeds between the discrete values, we can use slider and counter objects like those in Figure 3.4.

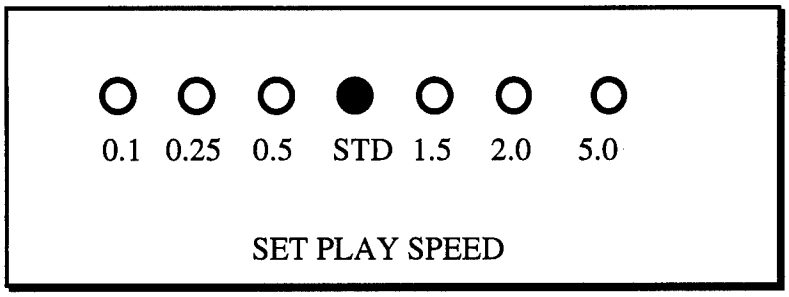



Figure 3.6. Radio Button Object Group

*3.6.4.2. Displayed Information.* The main DIS VCR screen should display information about the current recording to the user. The information must allow the user to determine which PDUs are being replayed, the date and time range when it was recorded, and what the recording is of. The main screen must also provide user feedback that indicates replay progress and clearly identifies where they are in the playback.

*3.6.5. Record Form Design.* To simplify data entry and reduce the user's error rate, the file header information should be captured via the form fill-in style. Record form entries must include user name, exercise ID, a title, a file name, and which PDUs to capture. Error checking should ensure that the user specified a file name and at least one PDU type to capture. The form should allow users to enter an exercise ID to specify the simulation they wish to record. The form must include a way for users to cancel the record process and return to the main screen. In addition, the form should provide a single-step access object for help.

*3.6.6. Select Exercise Form.* The selection form should give users an easy convenient way to select and load data files for replay. For file selection, an object like the pick-list in Figure 3.5 can be used. When the user selects a file, its information should be displayed so the user can identify the recording. The information should be identical to that entered in the record form. After selecting a file, the user must have some way to either delete the file or load it for replay. Error checking should ensure that the user chooses a file created by the DIS VCR. Like record, this screen must also have direct access to the on-line help.

*3.6.7. Other Screens.* Other screens include messages, questions, pause/resume, play progress, and record progress. Messages, questions, and progress dialogs provide informative feedback to the user and signal task completion. For example, when the user pauses a replay, a pause dialog should pop-up and provide an interface object that lets the user resume the replay.

*3.6.8. Interface Summary.* Now that the DIS VCR functionality, screen layout, and interface components have been specified, interface implementation can proceed. The implementation details are found in Chapter IV along with a discussion of our choice of high-level interface programming tools and their applicability to our design.

## *3.7. Supporting Unmodified Applications*

This section presents design changes for the DIS VCR that will allow it to support applications that are not modified to use a simulation clock and Replay PDUs. It may be that the application is built into hardware which cannot easily be modified, but could use the services of a variable speed replay application like the DIS VCR.

*3.7.1. Special Design Considerations.* A typical DIS rendering application maintains a State-Of-The-World (STOW) for all active entities in the simulation. The current state is used on each frame to correctly position and orient the entities based on PDU updates and dead-reckoning algorithms. The state is maintained with calculations that use the local system clock and the entity's velocity and acceleration vectors to predict where the entity should be on each iteration. Therefore, to get the appropriate response to a slow or fast motion replay without an adjustable simulation clock, adjustments to entity velocity and acceleration vectors are required.

*3.7.2. Slow-motion Replay or Pause.* Most DIS applications (i.e. RDT) are not built to handle pauses or speed variances. The dead-reckoning algorithms they use assume the simulation is in real-time and have no way of handling variable playback speeds. In slow motion, the application would dead-reckon entities past their actual locations (entities would appear to jump backward on each PDU update) and it would drop entities from the simulation due to the delay between updates.

To prevent entities from dropping out of the simulation in slow motion or pause mode, the DIS VCR must ensure PDU updates are sent out every five seconds as prescribed by the DIS standard (2:25). This requires the VCR to maintain its own dead-reckoned STOW.

*3.7.3. Maintaining State-Of-The-World.* DIS applications that maintain a STOW do not keep all of the information from the PDUs since they generally do not have to re-transmit them onto the network. The DIS VCR must keep all of the PDU information since it does transmit it back onto the network. The structure shown in Figure 3.7 holds the PDU, the elapsed simulation time since its last update, and the elapsed system time since the last time a PDU was broadcast for this entity. The elapsed simulation time is used to determine when an entity should be dropped from the active STOW list. The following sections discuss the DIS VCR actions in fast and slow replay modes.

| FIELD | DESCRIPTION |
|---|---|
| PDU | A full PDU |
| Elapsed Simulation Time | Simulation clock time |
| Elapsed System Time | Real-time |

Figure 3.7. STOW PDU Record

*3.7.3.1. Slow-motion Loop.* On each slow-motion loop iteration, the STOW PDUs are updated by dead-reckoning them with the simulation clock time and the entity velocity and acceleration vectors are modified to match the simulation clock time scale. Also, any PDU that has exceeded five seconds of elapsed real time, is broadcast onto the network. These actions are integrated into the play loop as shown in the pseudo code in section 3.7.4.

*3.7.3.2. Fast-motion Loop.* The fast-motion loop ignores the STOW since PDU updates are sent at faster than normal rates. The only thing the fast-motion loop does is modify the entity velocity and acceleration vectors to match the simulation clock time scale. However, special actions occur on each speed change for both slow and fast replay, as discussed in the next section, that requires the DIS VCR to keep an active entity structure even for fast-motion replays.

*3.7.3.3. Speed Changes.* If the user changes speeds, the DIS VCR must send a burst of all active entity PDUs with the new scale adjustment. If this is not done, application dead-reckoning will not match the current scale and will provide inaccurate results. Since the burst of adjusted PDUs is not instantaneous, there will be some noticeable lag in the application. Also, the burst will interrupt the sending loop and PDU updates will need to catch up to the simulation clock. After this brief settling time, the application rendering will appear smooth again. The modified play loop in section 3.7.4 shows how speed changes are handled.

An alternative to the PDU burst in fast-motion replay, is to let the entity's velocity and acceleration vectors be updated as they arrive in the input stream. The application settling time

will vary with the DIS VCR speed and total active entity count. During this settling time, relative velocities between updated and non-updated entities will be greatly distorted.

*3.7.4. The Modified Play Loop.* The following pseudo code is a reiteration of the play loop with the new modified portions appearing in italic bold print.

**Pre-processing Steps**

*Select a file*
*Set the simulation clock*
*Spawn the two-thread producer-consumer process that gets data from disk*
*Start the DIS VCR simulation clock*
*Fill and send an initialization Replay PDU*

**End Pre-processing Steps**

**Outer loop**

*While Not stopping and Not end-of-file*
*Hold for full PDU block*

**PDU Block Loop**

*While Not stopping and Not end-of-block or end-of-file*

*If in PLAY mode*
***Update or add entity to the STOW***
***Multiply PDU velocity and acceleration vectors by scalar***
***If slow-motion***
***Broadcast adjusted PDUs for entities that have not had an update within 5 seconds of system time***
***End If***

*While PDU time Not equal to simulation clock time*

*Check user input, respond to STOP, PAUSE, and speed changes*
*STOP: Set stop variable to drop out of loops*
*PAUSE:*
*Begin*
*Stop clock, set mode=PAUSE*
***Send PDU burst, velocities and accelerations set to zero***
*End*

*SPEED:*
*Begin*
*Change DIS VCR clock scale*
- ***Traverse STOW and send PDU burst with adjusted velocity and acceleration vectors.***
- ***send PDUs from the stream as fast a possible until even with the simulation clock.***

*End*
***Update whole STOW with dead-reckoning***
*End While PDU time Not equal to simulation clock time*

*Send PDU*
*Update display*
*End if PLAY mode*

*Else PAUSED*
*If resume then*
*Send mode-change Replay PDU*
*Restart simulation clock*
***Send PDU burst with adjusted velocity and accelerations***
***Send PDUs from the stream as fast a possible until even with the simulation clock***
*End if resume*
*End Else PAUSED*
*End While Not stopping and Not end-of-block or end-of-file*

**End PDU Block Loop**

*Swap to next PDU block*
*End While Not stopping and Not end-of-file*

**End Outer loop**

**Post-processing Steps**
*Kill the producer process, it is no longer needed*
*Send STOP mode Replay PDU*
*Reset data file*
*End else*

**End Post-processing steps**

*3.7.5. Summary.* Supporting unmodified applications takes considerable effort and adds an enormous amount of overhead to the replay loop. For large entity counts, replays other than slow-motion may not be possible. Also, analyses based on entity velocity and accelerations will not be valid at speeds other than normal.

### *3.8. DIS VCR Integrated Audio*

The DIS VCR already includes a way to turn on the capture of signal PDUs that carry DIS audio. However, the current play loop implementation does not provide a means to play the audio. Section 1.1 stressed the importance of audio for realism in remote debriefings with RDT and said that the ability to monitor radio communications would be a significant step forward for the remote debriefing concept. This section outlines the design changes needed to provide real-time audio in DIS VCR replays.

*3.8.1. Design Drivers.* The biggest design issues are the length of time it takes to play digitized audio and the size of the data. We have demonstrated that dedicated audio hardware can capture Red Flag UHF audio, convert it to DIS signal PDUs, and broadcast it on a wide-area network. The same hardware can read signal PDUs off the network and play the audio using its built-in sound board. Therefore, our audio design is not concerned with how signal PDUs get to the DIS VCR.

*3.8.2. Signal PDU Length and the Current VCR Design.* The signal PDU data fields can expand its size to 1500 bytes; the limit set by the Ethernet frame (1:24). The length is much higher than the other PDUs of interest which are 144 bytes or less as was shown in Figure 2.2. This is not a problem since the DIS VCR PDU block design accommodates PDUs of any size and the interface allows users to select signal PDUs as one to be captured.

*3.8.3. A Separate Process for Audio.* The size of digitized audio data in a signal PDU makes it imperative to separate its replay from the replay of other PDUs. We can use the same producer-consumer relationship that we used for our PDU block I/O approach. When a signal PDU is encountered in the data stream from disk, it can be handed off to a parallel process that handles conversions and replays the extracted audio data while the DIS VCR continues sending other

PDUs. The DIS VCR fills an audio structure with the raw data and marks it full. The audio process detects a full condition and starts processing the data for replay. This process may include decompression of the data and formatting it for the audio program to use. This process is shown in Figure 3.8.

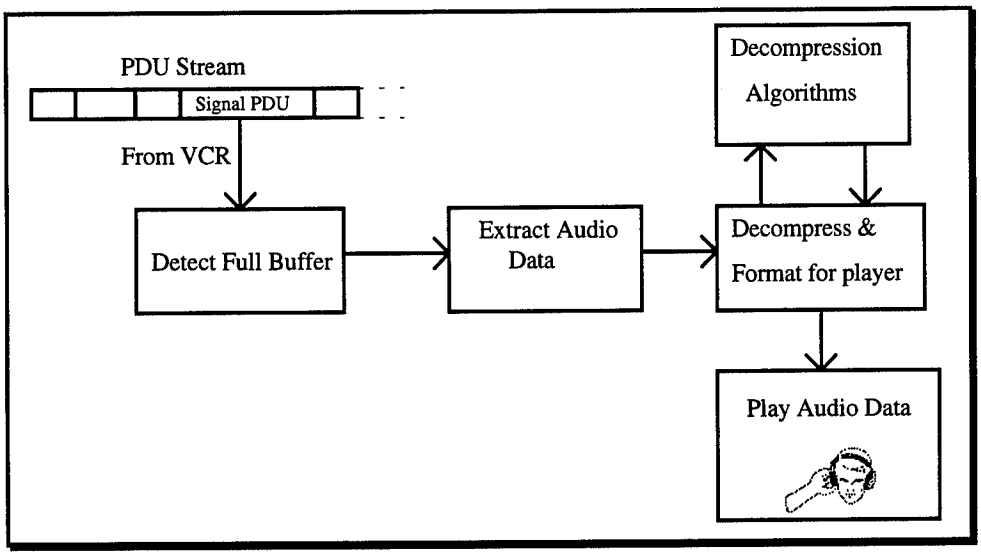



Figure 3.8. Audio Conversion Process

*3.8.4. Audio Design Summary.* Except for the decompression algorithms and formatting for a particular audio program, Figure 3.8 represents a complete design. The DIS VCR is already designed to capture signal PDUs and, with time, could be modified to include audio replay.

# *IV. Implementation*

## *4.1. Introduction*

Chapter III presented our design for record, variable speed replay, and the user interface that controls the DIS VCR system. This chapter examines the implementation of the DIS VCR functions and user interface design.

## *4.2. The Tools of Implementation*

The standard AFIT network daemons are implemented in C and all AFIT applications using replay are written in C++. Also, the AFIT C and C++ development environment has mature programming tools including the public domain high-level Forms interface libraries selected for the DIS VCR implementation. Therefore, C and C++ were natural choices for programming the DIS VCR. This is especially true for the simulation clock which had to be integrated into the C++-based Remote Debriefing Tool (RDT) and Synthetic Battle Bridge (SBB) applications.

## *4.3. Simulation Clock Implementation*

The simulation clock was on the critical path for DIS VCR implementation since the control of replays depended on it. The following sections describe how the clock was implemented to provide the needed functionality for both the DIS VCR and replay-modified applications.

*4.3.1. Simulation Clock Functions.* Table 4.1 lists and describes the simulation clock functions. These were coded as methods in a C++ class called Simulation_Clock. When an instance of Simulation_Clock is created, the clock date is initialized to 1 January 1970 and the time is set to midnight. These values are the lower limit imposed by the UNIX operating system. Although the C++ language provides UNIX compatible data types for the real time and date, we

needed special types to keep track of the simulation date and scaled time. The resulting time and date structures are shown in Table 4.1. The sim_time_t type is used for reporting time with a

Table 4.1. Simulation Clock Functions

| FUNCTION | DESCRIPTION / USAGE |
|---|---|
| set_sim_time | Allows the simulation clock to start from any time using struct of type<br><br>sim_time_t {<br>unsigned short sim_hour;<br>unsigned short sim_minute;<br>unsigned short sim_second;} |
| get_sim_time | Returns the scaled simulation time. If scale is set to 1.0, it acts like a normal clock. |
| set_sim_time_double | Uses a double to set time in seconds since first simulation day |
| get_sim_time_double | Returns a double of seconds since first simulation day |
| set_sim_date | A date between 1 Jan. 1970 and 2038 represented by a struct of type<br><br>sim_date_t {<br>int sim_month;<br>unsigned short sim_day;<br>unsigned short sim_year;}<br><br>Represents the first simulation day |
| get_sim_date | Returns the date set with Set Simulation Date. |
| change_time_scale | Uses a float to vary the speed of the clock |
| get_time_scale | Returns the time scale as a float |

resolution of seconds. Since microsecond resolution is used in DIS application dead-reckoning and in determining the time release of DIS PDU packets, we use a double that represents the number of seconds ( with 15 digit precision) since the beginning of the first simulation day. The precision of a 32-bit float is only accurate for six digits of resolution, so a double was required (15). The time since the beginning of the first simulation day, provides a method for determining when a recording crosses midnight and in calculating scaled time.

*4.3.2. Scaled Time* The simulation time is updated on every call to get_sim_time or get_sim_time_double by the following calculation:

$$sim\_time = last\_sim\_time + real\text{-}time\ since\ last\ call * time\_scale$$

where the real-time since last call is a system clock difference between the two calls. For example, if the simulation time is 12:33:59, time_scale is 10, and real-time passage was 1.0 seconds, then the new sim_time is 12:34:09.

## *4.4. Recording*

The following sections describe the implementation of the recording design with emphasis on maintaining an efficient record loop.

*4.4.1. File Header Information.* The file header information that includes all of the fields specified in section 3.4.1 is shown in Table 4.2.

Table 4.2. File Header Information

| FIELD NAME | SIZE (bytes) | TYPE / DESCRIPTION |
|---|---|---|
| simdate | 8 | A sim_date_t type (see Table 4.1) |
| name | 26 | A 26 character User Name |
| title | 30 | A 30 character title |
| PDU_flags | 20 | Array of 5 integer flags, shows captured ON/OFF |
| ex_id | 4 | An integer up to 65535 |
| site | 4 | An integer up to 65535 |
| DIS_version | 4 | An short int value of 3 indicating DIS Version 2.0.3 |
| start_time | 8 | A double of seconds since 1st sim day |
| stop_time | 8 | A double of seconds since 1st sim day |
| PDU_count | 4 | Total PDUs in file, up to 4294967296 for unsigned int |
| num_blocks | 4 | Total number of PDU blocks stored |

The DIS VCR writes a partially filled header to disk to reserve room for it. When recording stops, the program writes an updated header to that space.

*4.4.2. The Storage and Retrieval Scheme.* The storage and retrieval scheme implementation for the PDU block approach, was accomplished using the block structure shown in Figure 4.1.

| FIELD NAME | SIZE (bytes) | TYPE / DESCRIPTION |
|---|---|---|
| start_time | 8 | A double of seconds since 1st sim day |
| stop_time | 8 | A double of seconds since 1st sim day |
| empty | 4 | Variable for communicating empty/full condition |
| version | 4 | Set to 1 for this DIS VCR version |
| count | 4 | Number of PDUs in this block |
| offset | 4004 | Array of pointers into the data char array for up to 1000 PDUs |
| data | 152,000 | Data array, can hold up to 1000 entity state, fire, and detonation PDUs |

Figure 4.1. PDU Block Structure

There are actually 1001 offset pointers used in the block traversal algorithm. The extra pointer marks the end of the PDU character array when the maximum PDU count is reached and is used to determine the length of the last PDU as follows:

$$Length = offset[i+1] - offset[i]$$

The algorithm for traversal could have used the PDU length field to move forward through the character array, but this does not provide for future DIS VCR versions that may want to play the data in the reverse direction.

The start and stop times are used in the play loop to schedule the timely release of PDUs. The empty field is used to ensure that each block is ready to be written or filled, so a memory collision

does not occur. The record loop (the producer) uses a spin and wait loop to ensure the block has been fully written to disk by the consumer process before it begins using it as shown in the following pseudo code:

**Producer loop**

*While not done recording*

*While current block is not empty, do nothing*
*End while not empty*
*Fill the current block*
*Switch to other block*

*End while not done recording*

**End producer loop**

The process that writes a full block to disk, uses the opposite value of the empty field to know when a block is ready to write. Its spin loop is as follows:

**Consumer loop**

*While process active*

*While current block is empty, do nothing*
*End while empty*
*write full block*
*switch to other block*

*End while process active*

**End consumer loop**

This two-threaded parallel consumer-producer process forms the basis of the DIS VCR storage and retrieval scheme.

The version identifier shown in Figure 4.1 will allow future DIS VCR versions to select replay algorithms based on its number. The total number of PDUs stored in a block is entered in the count field and is used to delimit the block play loop. The offset array values point to PDU

locations in the data array. The offsets were required for storing and accessing variable length PDUs. Offsets are calculated as follows:

$$current\ offset + PDU\ length$$

Finally, the data array is a large character array for storing PDUs back-to-back. Getting PDUs in and out of the array is analogous to reading and writing one PDU at a time, but it is read into memory as a block and parsed using the PDU count and offsets.

*4.4.3. Filtering the Input.* In section 3.4.4 we stated that the DIS VCR design was built to accommodate certain filtering of input PDUs. The programming methods chosen to isolate a specific exercise, site, host, entity ID, range of entity IDs, and capturing of certain PDU types, are shown in Table 4.3. Since the exercise ID and PDU types are likely to be used the most, they are included in the interface component for recordings.

Table 4.3. DIS VCR Filters

| FLAG | SET BY |
|---|---|
| Site ID | Command line switch |
| Host ID | Command line switch |
| Entity ID | Command line switch |
| Entity ID Range | Command line switch |
| Exercise ID | Command line switch or Interface |
| PDUs to Capture | Command line switch or Interface |

*4.4.4. Limiting the Overhead.* We wanted the record loop to be as efficient as possible, but still have it provide feedback to the user on its progress. Progress updates require expensive screen I/O and could cause the loss of PDUs when entity counts are high. The solution to this problem is to limit the number of updates. Our record loop implementation uses a modulus function to limit

those updates to every 50th PDU. However, the user will look for progress immediately, therefore progress is also updated on the first PDU.

*4.4.5. Record Loop.* The record function that takes PDUs from the network and stores them to disk is shown below in pseudo code.

**Pre-processing Steps**

*Spawn the two-thread producer-consumer process*
*Get information for the file header from command line switches and the interface*
*Resolve any input errors*
*Start the simulation clock with local time at standard speed*

**Recording Loop**

*While not done*
    *If no PDU in five seconds, show message*
    *Get a buffered PDU from the daemon*
    *If PDU fits all parameters*
        *Copy it to the PDU block structure*
        *If first PDU, set file start time*
    *End If*
    *Store the receipt time with the PDU and write a value to the pointer array*
    *Update the displayed clock with simulation time*
    *Mark the daemon buffer empty*
    *If the block's count = MaxCount or block is full, mark it full and swap blocks*
    *(the consumer detects the full condition, writes the block to disk, then marks it empty)*
    *If PDU count is one or on a fifty boundary, show progress update to user*
*End While*

**End Record Loop**

*If the user stopped before a MaxCount or full count, mark last block full*
*Set file stop time to time of last PDU received*
*(the consumer detects the full condition and writes the last block to disk)*
*When write is complete, write header, close file, and set it up for immediate replay*
*Kill the consumer process, it is no longer needed*

**End Record**

Note that the code allows for feedback to the user, but not on every PDU. This overhead saving strategy puts 49 *if* statement executions in place of 49 graphics screen updates.

## *4.5. Playback*

*4.5.1. Reading and Using Header Information.* The file header shown in Table 4.2, is read when a file is first selected (see Appendix A on how to select a file). The DIS VCR uses the simdate and start_time header fields to set the simulation clock's initial time and date. They are also used with the rest of the data to populate the identifying information on the right side of the main VCR screen shown in Figure 4.2.

*4.5.2. Replay With a Simulation Clock.* The simulation clock is used to control replay functions by means of scaling. When the scale of the simulation clock changes, that change must be communicated to all modified applications using the DIS VCR for replays to prevent the problems discussed in section 3.2.3.1. The following sections discuss the mechanism for communicating scale changes and how a change in scale affects each replay function.

*4.5.2.1. A New PDU for DIS Replays.* The Replay PDU supports the following actions:

- Switch mode to PLAY, PAUSE, or STOP
- Update the simulation clock time or date
- Change the time scale
- Initialize, start, or stop the replay

Whenever the DIS VCR changes mode, time or speed, it transmits a Replay PDU on the network to report the change to receiving applications. The functional actions are captured in the replay state diagram of Figure 4.3 and are discussed in subsequent sections.

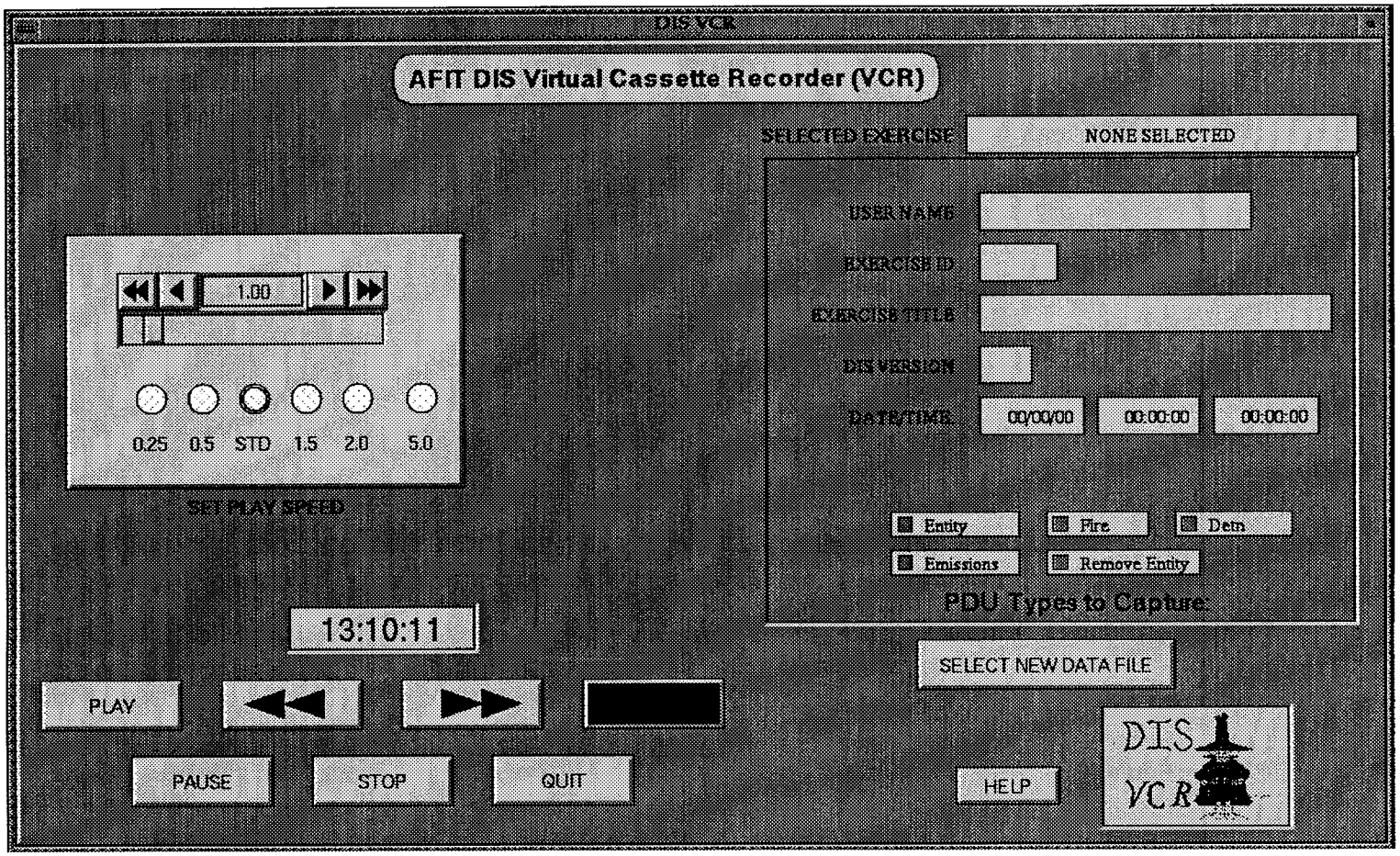



Figure 4.2. Main VCR Screen

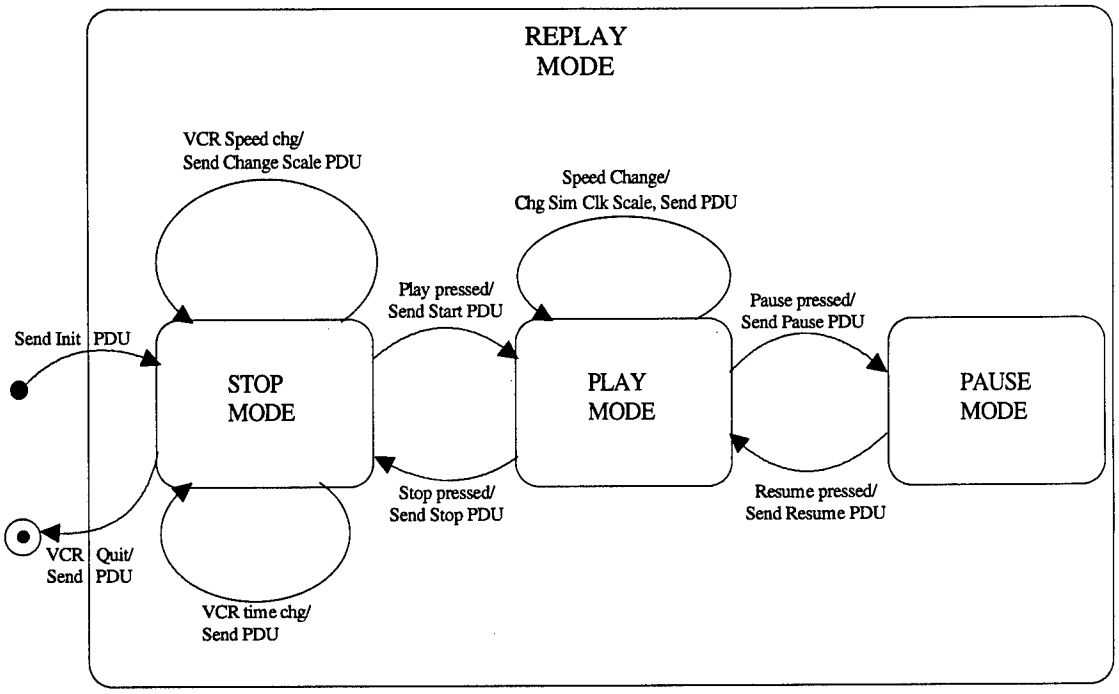



Figure 4.3. Replay State Diagram

As depicted, replay initialization is what signals the application to start using its simulation clock instead of the system clock. The terminal condition (a circle with a dot in it) causes the application to revert back to using the system clock. Each state transition generates a Replay PDU formatted as shown in Figure 4.4. The shaded area is the common PDU header described in section 2.2.1. The final seven fields are used to represent replay actions and variables.

The Replay PDU is patterned after other DIS standard PDUs to maintain compatibility with them and for submission as an addition to that standard. For example, the simulation time field is identical to the simulation management PDU time stamp (2:102) and the enumerations all begin with other as specified in (12).

The Request ID contains a value of zero on the broadcast of the initialization Replay PDU. Every broadcast of a Replay PDU increments the value of the Request ID. When the value of

Request ID reaches 256, the value is reset to one. These values let the receiving application sequence the incoming Replay PDUs.

| Size-bits | | Replay PDU Fields |
|---|---|---|
| 96 | *PDU Header* | Protocol Version - 8 bit enumeration |
| | | Exercise ID - 8 bit unsigned integer |
| | | PDU Type - 8 bit enumeration |
| | | Protocol Family - 8 bit enumeration |
| | | Time Stamp - 32 bit unsigned integer |
| | | Length - 16 bit unsigned integer |
| | | Padding - 16 bits unused |
| 48 | *Originating Entity ID* | Site - 16 bit unsigned integer |
| | | Application - 16 bit unsigned integer |
| | | Host - 16 bit unsigned integer |
| 48 | *Receiving Entity ID* | Site - 16 bit unsigned integer |
| | | Application - 16 bit unsigned integer |
| | | Host - 16 bit unsigned integer |
| 64 | *Simulation Time* | Hour - 32 bit integer |
| | | Time Past Hour - 32 bit unsigned integer |
| 32 | *Time Scale* | 32 bit float - 0.1 to 10.0 for the VCR |
| 8 | *Request ID* | 8 bit integer |
| 8 | *Mode* | 8 bit enumeration (12)<br>- 0 Other<br>- 1 Stop<br>- 2 Pause<br>- 3 Play |
| 8 | *Direction* | 8 bit enumeration (12)<br>- 0 Other<br>- 1 Forward<br>- 2 Reverse |
| 8 | *Reason* | 8 bit enumeration (12)<br>- 0 Other<br>- 1 Initialize<br>- 2 Start<br>- 3 Pause<br>- 4 Resume<br>- 5 Stop<br>- 6 Quit<br>- 7 Change Time<br>- 8 Change Time Scale<br>- 9 Change Direction |

Figure 4.4. 320-bit Replay PDU

*4.5.2.2. The Speed Change Process.* When the user selects a new speed, the DIS VCR sends out a Replay PDU to change the time scale. The simulation time, time scale, and reason are all updated to reflect the new state and the request ID is incremented. The VCR displays the new speed, sets its simulation clock speed to the new value, and begins releasing PDUs accordingly. Upon receiving the Replay PDU, the receiving application's dead-reckoning and entity time-out algorithms use the new scale to control their activities.

*4.5.2.3. Speed Limits.* The DIS VCR's slowest and fastest replay speeds are one tenth and ten times that of standard wall clock time. Although speeds outside this range are possible, we chose the same values typically used in home VCRs. Realistically, the maximum speed is a limit imposed by the speed of the replay loop which is governed in part by the network interface and the hardware.

*4.5.2.4. Pause.* When pause mode is selected, a Reply PDU is prepared with the current simulation time with mode and reason set to pause. The DIS VCR sets the simulation clock scale to zero which freezes the PDU stream at the current point in the PDU block. The receiving application is responsible for keeping entities in its current player list alive until a resume Replay PDU is received.

*4.5.2.5. Applications and Replays.* The synchronization problem discussed in section 3.5.2.4 does not have a perfect solution in the absence of a shared real-time clock. One compromise is to have the DIS VCR send the time-scale Replay PDU to the receiving application and wait for some set amount of time (e.g. one second) before changing the scale on its simulation clock. The amount of time to wait is dependent on the network's efficiency, routing, and current load and therefore makes this a non-viable solution. A second way to handle DIS VCR and

receiving application synchronization, is to add a field to the Replay PDU for the local time when the change becomes effective. Neither program would change the simulation clock until the local time matched the time specified for the change. This solution only works if the local clock times are within milliseconds or less of each other. Thus, the sending and receiving sites would need some way of ensuring the clocks match. Finally, the synchronization could be ignored if the user does not mind a little settling time after speed changes. This is what the DIS VCR does since any other solution would be out of the scope of this thesis effort.

*4.5.3. Replay Overhead.* An overhead reducing strategy similar to the one in the record loop is also used in replays for progress feedback. However, the same efficiency could not be achieved because the play loop performs repeated clock comparisons on every PDU to time their release. The play loop also checks for speed and pause changes during a replay; something record does not need to do (see section 4.5.4). These differences create more branch conditions to test which slows down the play loop. See Chapter V for data on replay maximum speed versus entity counts.

*4.5.4. Play Loop Pseudo Code.* The play function that takes PDUs from the disk and re-transmits them across the network, is shown below in pseudo code. The user may change speeds, pause, and stop while inside play's PDU block loop.

**Pre-processing Steps**

*Select a file if one is not loaded*
*Read the header, set the simulation clock, and populate info on VCR main screen*
*Hold the simulation clock until the loop begins*
*Spawn the two-thread producer-consumer process that gets data from disk*
*Read first two blocks*
*Start the DIS VCR simulation clock*
*Fill and send an initialization Replay PDU*

**End Pre-processing Steps**

**Outer loop**

*While Not stopping and Not end-of-file*

*Hold for full PDU block*

**PDU Block Loop**

*While Not stopping and Not end-of-block or end-of-file*

*Capture user inputs*

*If in PLAY mode*

*Check for speed changes, set if changed by user*

*Get daemon buffer and copy current PDU to it*

*While PDU time Not equal to simulation clock time*

*Check user input, respond to STOP, PAUSE, and speed changes*

*STOP: Set stop variable to drop out of loops*

*PAUSE: Stop clock, set mode=PAUSE*

*SPEED: Change clock scale and continue this loop*

*End While PDU time Not equal to simulation clock time*

*Mark the daemon buffer full so it will send it*

*Update displayed clock and progress indicators*

*End if PLAY mode*

*Else PAUSED*

*If resume then*

*Set mode=PLAY*

*Send mode-change Replay PDU*

*Restart simulation clock*

*End if resume*

*End Else PAUSED*

*End While Not stopping and Not end-of-block or end-of-file*

**End PDU Block Loop**

*Swap to next PDU block*

*End While Not stopping and Not end-of-file*

**End Outer loop**

**Post-processing Steps**

*Kill the producer process, it is no longer needed*

*Send STOP mode Replay PDU*

*If end-of-file (eof)*

*Reset data file to the beginning*

*End if eof*

*Else*

*Reset data file to beginning of current block*

*End else*

**End Post-processing steps**

*4.6. The DIS VCR Interface Implementation*

The main theme for the DIS VCR interface design was a simple emulation of the home VCR with goals to:

- Reduce the interface learning curve
- Maximize task speed
- Minimize the user error rate
- Supply easy access to help
- Provide easy recording or data file identification

Since our primary user is the same as RDT's, we satisfy the first goal by leveraging our user's familiarity with RDT's interface by reusing the same high-level interface tools as in RDT. This makes the Forms Library (16) the tool of choice for building the DIS VCR interface. The Forms Library was built to reduce the time consuming process of constructing the user interface on Silicon Graphics Workstations. The main theme of an interface built with the Forms Toolkit, is a form (a window-like object) that is populated with interaction objects that signal events or issue direct call-backs to user defined functions. The interaction objects include all those identified in section 3.6.3.1.1 and several others. Each of the following sections shows the form that was created for each screen shown in Figure 3.3 and discusses its actions and contribution to satisfying the interface goals.

*4.6.1. The Main VCR Form.* The main DIS VCR form gives a user access to all of the DIS VCR's functionality (see Figure 4.5). From here, users can initiate a replay, recording, or actions to direct the current replay. The main form is divided into four functional groups which are geographically separated. In the upper left corner of the form, there is a cluster of three objects used to control the replay speed. On top is a counter object with two increment buttons on each side. The outside buttons change the speed by ± 0.1 and the inside buttons modify it in

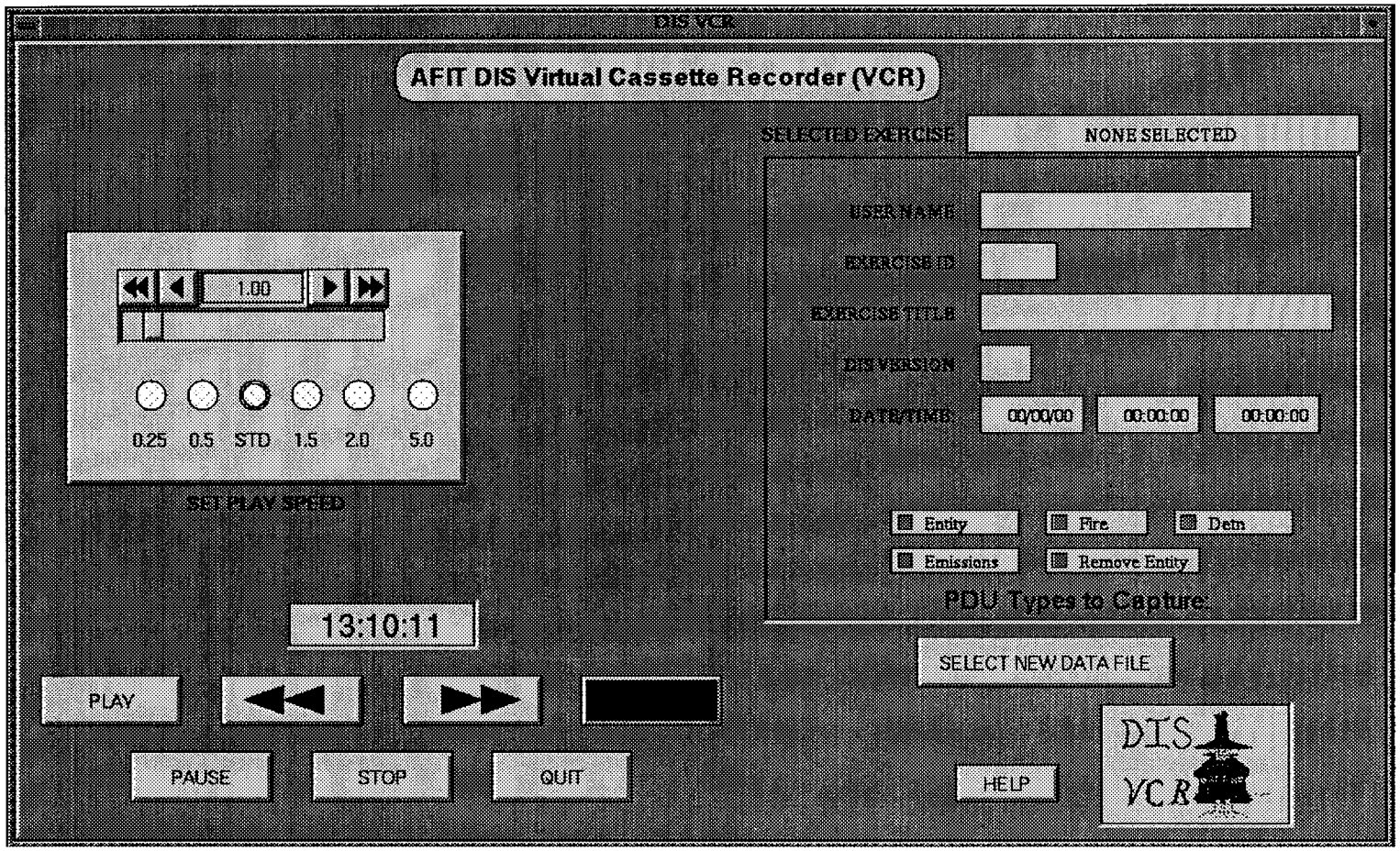



Figure 4.5. Main VCR Screen

± 0.01 increments. The center object slides from 0.1 to 10.0 in steps of 0.01 and returns its value when released. Positioned below the slider, there is a group of radio buttons allowing the user to select one of six typical speeds. This final object helps satisfy the goal of minimizing task speed since a single mouse click will change the replay speed. Also, since the user never enters a speed value directly, an error cannot be made.

The button controls emulating the typical VCR are clustered below the speed group. The meaning of the labels are identical in scope and function to a Video Cassette Recorder which helps to reduce the learning curve even further. Just above these buttons is the simulation clock display with time shown as military time.

The identifying information about the current recording is grouped together in the middle-right side of the form. Nothing is selectable in this group, it is an "information only" copy of pertinent file header information from Table 4.2. This information group contributes to meeting the easy identification goal. Furthermore, the information reminds the user of which PDUs are included in the recording, the duration of the replay, and when it will stop.

Finally, one of the two buttons on the lower right provide access to the on-line help while the other lets the user start a new file selection process (see section 4.6.3).

Once a recording has started, a lack of incoming PDUs for five seconds triggers the message shown in Figure 4.6. This lets the user know that either the simulation has not begun, the daemons were not started properly, or that something is wrong with the way the record filters were specified.


Figure 4.6. Time-out Message for PDUs

*4.6.2. Record Form.* The record information is entered using the form shown in Figure 4.7. The input fields on the left simplify data entry and reduce the user's error rate by restricting the type of information a user may enter (characters or numbers). The buttons on the right allow the user to select which PDUs to capture. These buttons light up when the PDU is selected for capture (user feedback). Pressing the done button starts the record process, or the user may select the cancel button to gracefully exit the record form and return to the previous state. In addition, there is a button for help on the record form.

The DIS VCR looks for certain errors after the done button is selected. First, it ensures that the file name field was not left blank. If it was, the user is informed with the message shown in Figure 4.8 and is returned to the record information form when OK is pressed. Next, the DIS VCR

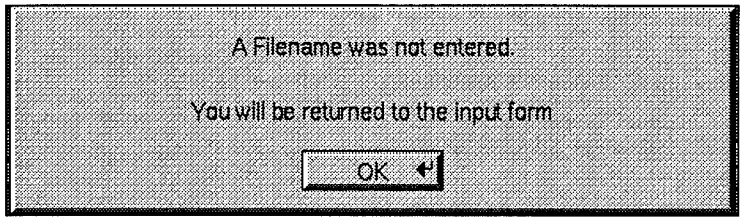



Figure 4.8. File Name not Entered Message

checks to see if a duplicate file name already exists in the directory. If the file name has previously been used, the user is asked for overwrite confirmation using the form in Figure 4.9. A

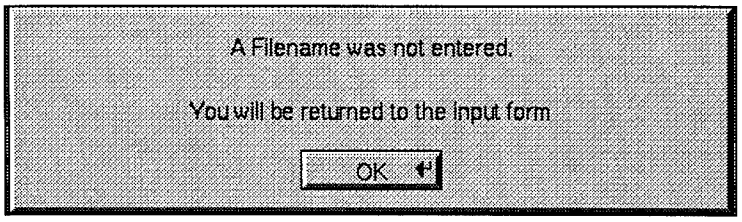



Figure 4.9. File Name Exists Message

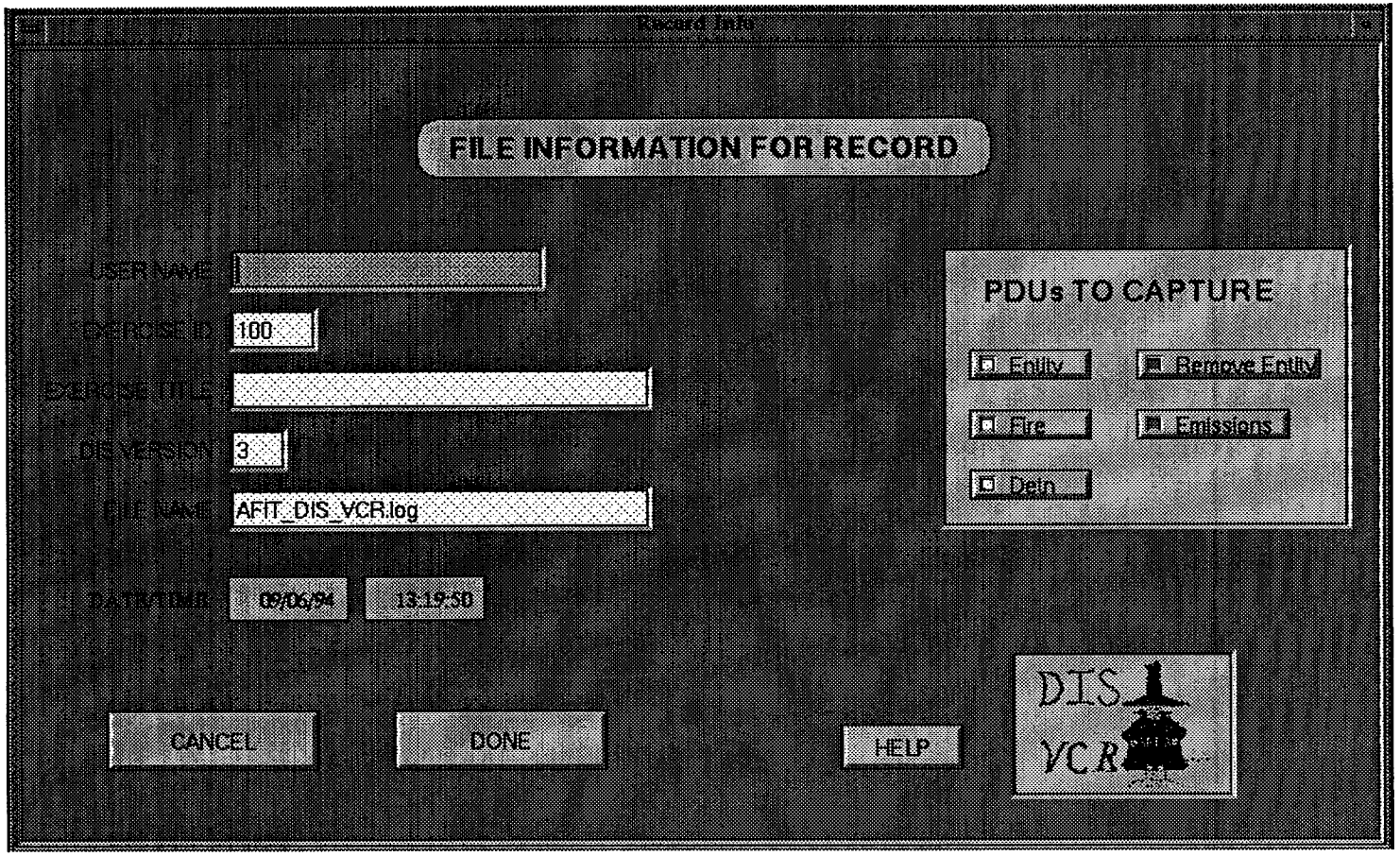



Figure 4.7. Record Form

yes answer overwrites the existing file with the new recording; otherwise, a no answer returns the user to the record form to allow modification to the file name.

When all the errors have been cleared, recording begins and the progress form in Figure 4.10 is displayed. The form uses a red bar to indicate progress; it also displays an accumulated count of received PDUs. These indicators provide nominal feedback to keep the user aware of the recording's progress.

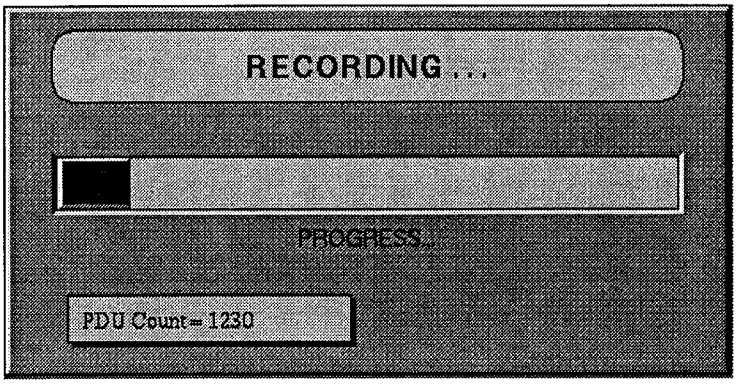



Figure 4.10. Progress Form

*4.6.3. Select Exercise Form.* The selection form gives users an easy convenient way to select and load data files for replay. The key selection object is the pick-list on the left side of the form as shown in Figure 4.11. When the user picks a file from the list, its identifying information is displayed in the object group on the right. After selecting a file, the user can delete it or load it for replay using the corresponding buttons. Like the record and main DIS VCR forms, this screen has direct access to on-line help. The cancel button lets the user gracefully exit the selection form and return to the previous state.

If the selected file is not a DIS VCR recording, the user is informed with the form shown in Figure 4.12 and returns to the selection form. If the user selects the load button prior to choosing a valid file, the message in Figure 4.13 is displayed after the file selection form is closed.

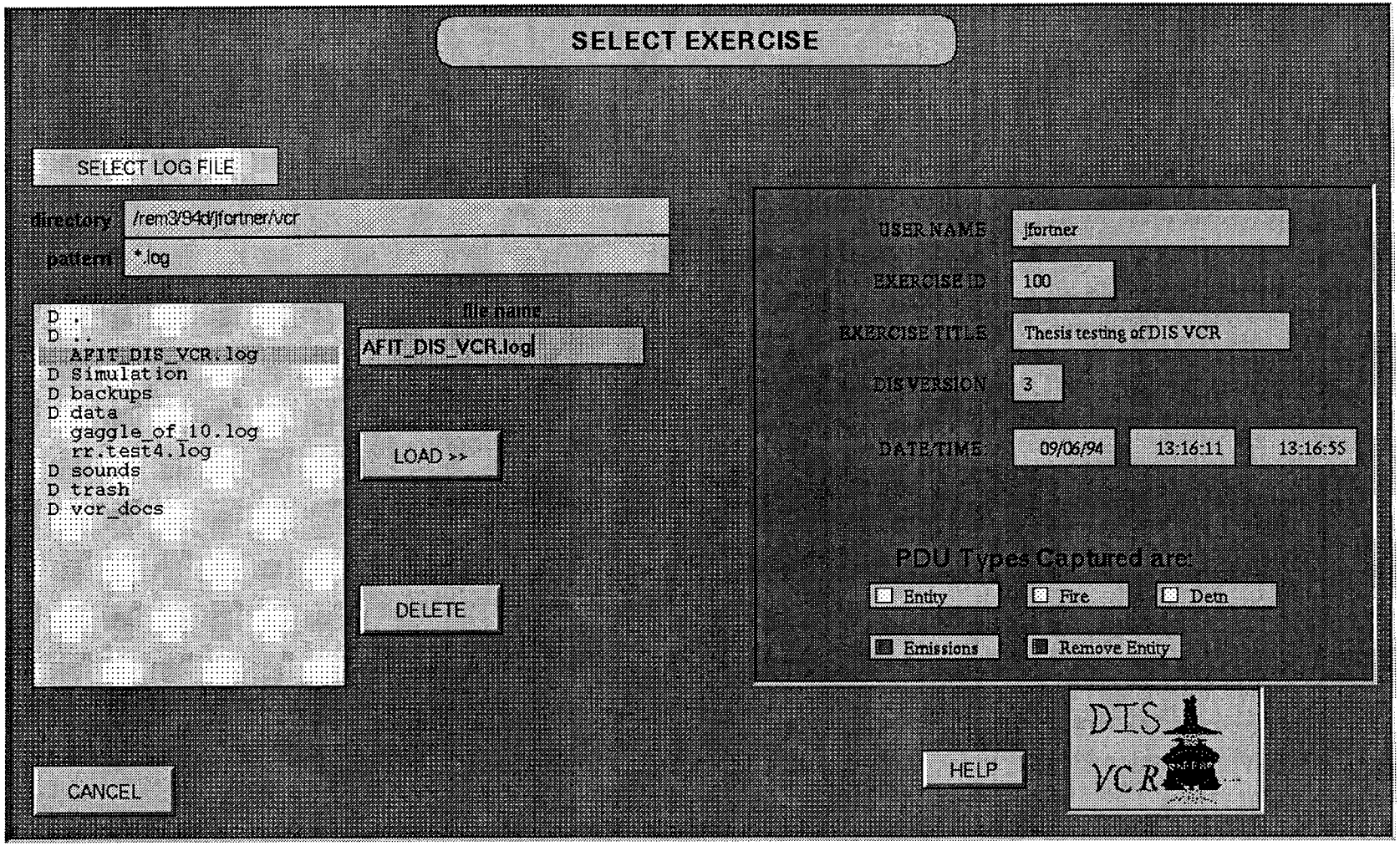



Figure 4.11. Select Exercise Form

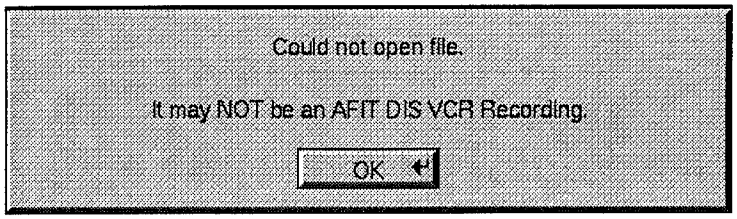



Figure 4.12. File is not a DIS VCR Recording Message

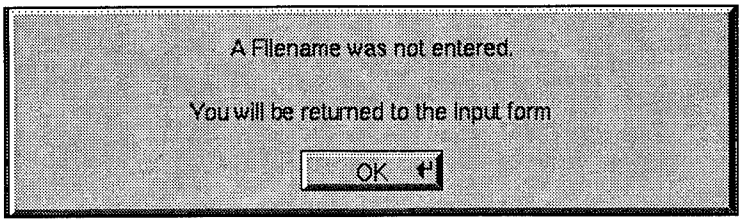



Figure 4.13. File was not Selected Message

These messages let the user know when something has gone wrong and needs to be corrected. They also play an important role in satisfying our interface goal of reducing user errors.

Once a file is successfully loaded, play begins and a progress form similar to the one for record is displayed. The only difference is that the progress bar is blue and the title reads play progress (see Figure 4.10).

### *4.7. Implementation Summary*

In this chapter we presented our DIS VCR implementation which included a simulation clock with scalable time and a Replay PDU that communicates time and speed changes to applications. The clock and Replay PDU events were incorporated in the record and play loops and illustrated using detailed pseudo code of those processes. To complete the building of the DIS VCR, a Forms-style interface was added that met the goals set forth in Chapter III. The design and implementation phases included some concurrent and post testing of DIS VCR performance. The test results are found in Chapter V which discusses replay speed limits, disk utilization, and network statistics.

## *5.1. Introduction*

Throughout design and implementation, various aspects of the DIS VCR were analyzed and tested. Early experiments with record and playback affected the selection of the storage and retrieval scheme and indicted areas for subsequent testing. The DIS VCR's performance is directly linked to disk I/O and network performance prompting extended investigations of these areas. This chapter presents DIS VCR performance data gathered in testing and analyzes some anomalies that occur in replays.

## *5.2. Network Performance*

Part of the early experiments with the play loop clearly demonstrated the limits of the standard AFIT network daemons. The play loop was set up with minimal overhead so that it could send PDUs out at maximum speed. Even the most efficient loop, which did not depend on disk I/O, could only send 900-1100 PDUs per second. This prompted the insertion of time calls before and after the daemon interface routines to measure their speed. The time measurements identified the daemons as a bottle-neck. The author of the daemon code, Bruce Clay, said that it was not uncommon to only get 350 PDUs per second per processor and cited 1500 PDUs per second as the maximum attainable performance using a four processor ONYX.

Replays can lose PDUs due to the relaxed UDP/IP protocols employed by DIS. This effect would be cumulative when using the DIS VCR as a demonstration building tool. This prompted us to check the robustness of the UDP/IP protocol on our local network. The test used the DIS VCR to transmit a predetermined number of PDUs onto the network and a PDU monitor who's only task was to count the number of PDUs received. Of the 45,000 PDUs sent in each run, two PDUs were

lost. This means that 0.00044% of PDUs transmitted may not make it to local receiving applications. As the error accumulates, it could cause DIS players to momentarily drop out of the simulation and be reloaded on the next update.

### *5.3. Storage and Retrieval Performance*

We also inserted code to measure disk I/O performance for the single and block PDU storage and retrieval schemes. This test demonstrated that the block approach could handle almost 10 times as many PDUs as the single PDU method. Hard disk transfer rates varied from 0.55-3.6MB per second when reading or writing one PDU at a time and from 2.68-22.34MB per second for blocks of 1000 PDUs. The average transfer rate for a 1000 PDU block was 13.1MB per second which supports 85962 PDUs per second. Unfortunately, the daemons can only handle a fraction of that and, even if they could handle it all, the Ethernet bandwidth could not. According to the system administrator, the local Ethernet has a bandwidth of 1.25MB per second with only about half of that available for user applications. If no other applications are using the network, the Ethernet can only handle about 3125 PDUs per second with an average packet size of 200 bytes (18:394). Therefore, it is safe to predict that the DIS VCR PDU block I/O performance will not require improvement for some time and will not be fully utilized until substantial enhancements to the network and daemons occur.

The DIS VCR can store 6562 entity state PDUs per megabyte using the block storage and retrieval scheme of section 4.4.2. The combination of *readred* and the DIS VCR lowers the storage requirement for remote debriefings by 4.4 to 17.5 times over the method depicted in Figure 2.5. Data storage comparisons are shown in Figure 5.1. The dead reckoning threshold *readred* uses (1 or 10 meters), affects the actual storage space needed for a DIS VCR recording as shown in the figure.

| DATA STORAGE MODE | STORAGE | LENGTH | 1m Dead-Reckoned *readred* PDU Output | 10m Dead-Reckoned *readred* PDU Output |
|---|---|---|---|---|
| ***RAW DATA*** | 184 MB | 44.65 mins | 184,078 PDUs | 46,596 PDUs |
| ***XFILTERED*** | 123 MB | 44.65 mins | | |
| ***DIS VCR*** | 7-28 MB | 44.65 mins | | |
| | | | | |
| ***RAW DATA*** | 21.40 MB | 5.18 mins | Unknown | 8,036 PDUs |
| ***XFILTERED*** | 12.66 MB | 5.18 mins | | |
| ***DIS VCR*** | 1.37 MB | 5.18 mins | | |

Figure 5.1. DIS VCR and *readred* Storage Comparison (1:133)

### *5.3. PDU Rates and Replay Speeds*

Since the daemons can only process 900-1100 PDUs per second, the maximum speed of the replay loop is limited by the entity count, or more precisely, the average number of PDUs received per second. Table 5.1 lists average PDU counts and shows the maximum replay speed for each.

Table 5.1. Average PDUs Per Second and Replay Speeds

| Low PDUs/sec | Avg. PDUs/sec | High PDUs/sec | DIS VCR Speed |
|---|---|---|---|
| 312 | 380 | 698 | 1.3 |
| 281 | 351 | 465 | 1.5 |
| 262 | 290 | 440 | 3.3 |
| 141 | 160 | 213 | 5.8 |
| 127 | 129 | 243 | 6.7 |
| 60 | 80 | 100 | 8.6 |
| 31 | 45 | 92 | 9.75 |

Table 5.2 lists PDU rate excerpts from the *10th DIS Workshop: Working Group Minutes* and numbers from a typical *readred* replay. The first two traces were 60 minutes long and consisted of 90-96% entity state PDUs and less than 0.3% fire and detonation PDUs (18:394). The average number of PDUs per second depends on the amount and type of each entity class. For the I/ITSEC DIS sample, the DIS VCR could handle 1059 entities at speeds up to 1.3.

Table 5.2. PDU Rates From Other Sources

| Source | Avg. PDUs/sec | #Entities |
|---|---|---|
| DEC Traces from I/ITSEC | 62.92 | 149 |
| scenario week. | 64.92 | 119 |
| I/ITSEC DIS Sample | 75.3 | 210 |
| readred replay | 25 | 57 |

### *5.4. RDT Replay Fidelity*

RDT replay fidelity was questioned when testing revealed that 13% of the stored PDUs were not in time stamp order for different entity IDs, even though PDUs for the same entity ID were (caused by the UDP/IP network protocol). We were concerned that RDT's dead reckoning algorithms would move the entity farther than it should have while waiting on the delayed PDU, causing the entity to jump backward on the update. The largest measured delay from the sample was 0.11 seconds. However, discontinuities in entity movement due to out of order PDUs is typically imperceptible. For example, consider an entity moving at 350mph in one of RDT's views.

Given:

Screen size: 1280 X 1024 over a 15.25in square

RDT 3D view scale: 6.00in = 15NM ; $\frac{6\,in}{15\,NM} = \frac{6\,in}{1094400\,in} = \frac{1}{182400}$

We can calculate how much the entity will move on screen:

**Horizontal pixels per inch:**

$$Hpixels = \frac{1280pixels}{15.25in} = 83.93\,pixels/in$$

**Actual distance the entity moved:**

$$Actual = 350\,mi/hr \cdot 0.11seconds = 677.6in$$

**3D View distance moved:**

$$3D\_move = \frac{677.6in}{182400} = 0.0037in$$

**3D view pixels moved:**

$$3D\_pixels = 0.0037in \cdot 83.93 pixels/in = 0.312 pixels$$

Similar calculations for the RDT plan view predict movement of 0.065 pixels in the view's unzoomed state. Obviously the numbers will not hold if we are zoomed on an entity moving at mach 2, but this is not the typical case. Therefore, PDUs stored out of order are of no immediate concern. However, the error could be cumulative if the DIS VCR is used for scenario building. To improve fidelity, the data could be post processed to correct the time ordering problem.

# *VI. Conclusions and Recommendations*

This chapter looks back at the thesis statement and measures success by comparing DIS VCR accomplishments with our original objectives, followed by a discussion of its contributions to the DIS community. The thesis closes with some recommendations for future work that will help enhance this thesis effort, the AFIT DIS environment, and the DIS world in general.

## *6.1. Thesis Objectives Revisited*

The primary objective of the DIS VCR was to add flexible replay capabilities to the Remote Debriefing Tool (RDT). Derived requirements from that objective included the selective filtering of incoming PDUs, variable replay speeds, ability to pause, fast-forward, rewind, efficient data storage and retrieval, and an interface that simplifies the execution of those functions. Secondary goals included a DIS VCR compatible design for concurrent UHF audio replay and an extension to the replay design that supports unmodifiable receiving applications.

To demonstrate the filtering capabilities of the DIS VCR, recordings were made using various combinations of PDU capture switches, exercise IDs, and entity IDs. These were replayed and viewed for correctness using other applications (i.e. RDT).

The various replay functions were put to the test when an RDT-DIS VCR demonstration was created for the Air Force Association Convention in Washington D.C. this year. The demonstration illustrated for the first time, the successful completion of variable speed replay, pause, fast-forward, and rewind. Also, the DIS VCR was used at the convention to replay a Red Flag mission used to demonstrate RDT's capabilities and an AFIT gaggle of 100 entities used to highlight the usefulness of the Synthetic Battle Bridge (SBB). The DIS VCR proved itself while serving literally millions of PDUs to using applications.

Chapter V showed the DIS VCR's storage and retrieval methods to be efficient enough to support the recording and playback of 100,000 entities; ARPA's 1997 goal for entities in a theater-

wide battle simulation of real and "computer generated forces" (4:8). With significant improvements in network and computer performance (about 30-50 times faster), the DIS VCR should be able to record and replay very large scale simulations.

To meet secondary goals, sections 3.7 and 3.8 presented designs for supporting unmodifiable applications and audio replays from signal PDUs. The designs were complete in concept and only need to be implemented.

## *6.2. Future Work*

This section suggests additions to the DIS VCR that will enhance its performance or add to its capabilities and recommends future areas of research and development.

*6.2.1. DIS VCR Additions.* The synchronization problem between the DIS VCR and replay-modified applications described in section 3.5.2.4 should be corrected using one of the methods described in section 4.5.2.5. The lack of synchronization detracts from the fidelity of DIS VCR replays during speed changes.

The DIS VCR should be modified to allow the user to reverse the replay direction. This would keep the user from having to guess how far to go back to see a particular event again. The Replay PDU is designed to handle the reverse direction, but the DIS VCR and simulation clock would need modified.

Since the DIS VCR includes a way to capture signal PDUs, functions should be added that implement the audio replay design presented in section 3.8. Being able to hear the UHF chatter from Red Flag exercises, would greatly enhance the realism of replays with the RDT-DIS VCR combination. Also, the DIS VCR was implemented to capture a small subset of the 26 possible PDU types. AFIT's projected expansion in this area would require DIS VCR modifications to add any new types we wish to support in replays (e.g. collisions, simulation management, etc.).

The analysis community would benefit from a single-step replay mode that displayed the PDU data and allowed the user to edit and save it. Analysts could use the mode to debug application and network anomalies and test the robustness of an applications tolerance to packet errors as suggested in Chapter 2. This idea could be expanded for use as a scripted demonstration creation tool. Users could plot out an entities adjusted path, edit the data to match, and replay the result. This may also include a way to insert new PDUs into the data stream to correct for network PDU losses or add to fidelity of certain maneuvers. Editing would allow analysts to mark entities as damaged or destroyed when their prediction models indicate a hit or kill.

The DIS VCR has many filters that select specific PDUs for recordings, but only one (exercise ID) that modifies the replay. Replay filtering can be accomplished by re-recording the data, but this ability would be more convenient if it were added to the replay interface.

*6.2.2. Other Recommendations.* If the DIS community is to meet the goals set by ARPA (4), the network bottleneck must improve. DIS VCR testing demonstrated the limits of the current system and research in its improvement is recommended. AFIT has already taken steps in this direction with research into the design of UDP/IP hardware outboard processing and plans to upgrade the local Ethernet. Research should include improvements to the DIS standard that reduces the amount of bandwidth required for DIS exercises (18).

## *6.3. Thesis Contribution*

The DIS VCR improves and extends the capabilities of RDT by providing one of the two most important functions missing from it, variable-speed replay (1:142). Created as a separate application, the DIS VCR also provides a way to produce Guckenburger's mission training scenarios with measurable performance (see section 2.4.1) and allows any DIS analytical group to use its capabilities. Finally, this thesis effort has proven the effectiveness of a new Replay PDU in orchestrating communications between the DIS VCR and any replay-modified application.

# *Appendix A. DIS VCR User's Guide*

This appendix contains a complete description of how to use the AFIT DIS VCR (from here on the VCR). It covers the details of setting up the equipment, followed by the VCR start-up procedures, then instructions for making a recording and playing it back, and finally a wrap-up section.

## *A.1. Getting Started*

This section explains the hardware and software setup needed prior to starting the VCR. You will also find a list of all applicable files and a description of what each does.

*A.1.1. The Hardware.* Most Silicon Graphics workstations can run the VCR. However, the VCR can handle more entities on faster CPUs and on multi-processor machines. The minimum recommended computer is a 100MHz Indigo.

The VCR code has a compilation switch for making audio available. An SGI Indigo, with its standard sound card, is sufficient to play the VCR sounds.

The VCR requires a connection to the network. It uses an Ethernet system and a specific set of network daemons written for AFIT (see A.1.2). These must be available for the software to function.

*A.1.2. The Software.* This section divides the required software list into VCR source code, executable and data files, and network daemon categories. This information is collected into Table A.1 for convenience. The references to directories will assume you are on a machine that has access to the WARBREAKER directories.

Table A.1. VCR Files, Their Functions, and Locations

| FILENAME | FUNCTION or USE | LOCATION |
|---|---|---|
| **EXECUTABLE/DATA** | | |
| vcr | Main executable | /usr/people/wb/bin/vcr |
| *.log | Recorded data files | /usr/people/wb/bin/vcr |
| vcr.logo.xbm | Bitmap displayed on the main FORM | /usr/people/wb/bin/vcr |
| **SOURCE CODE** | | |
| vcr.c | Main source code file | /usr/people/wb/src/vcr |
| vcr.h | Main include. Defines, VCR types, all externals used in VCR FORMs. | /usr/people/wb/src/vcr |
| vcr_forms.c | Code that builds all VCR forms | /usr/people/wb/src/vcr |
| sim_time.h | Types/defines for Simulation Clock | /usr/people/wb/src/vcr |
| Simulation_clock.cc | Complete source for all Sim Clock functions. | /usr/people/wb/src/vcr |
| Simulation_clock.h | Specification header for .cc | /usr/people/wb/src/vcr |
| **NETWORK** | | |
| sgi_recvd | Executable for the receive daemon | /rem4/afa94/bin |
| sgi_sendd | Executable for the send daemon | /rem4/afa94/bin |

In addition, the workstation must have access to G/L libraries and standard system calls; however, these are included in the latest release of IRIX. The listed source code is not needed for execution, it is only included for completeness.

*A.1.3. Before Starting the VCR.* The way you set up your machine before starting the VCR depends on how you intend to use it. First, determine the daemon setup using the following command line

*ps -elf | grep sgi*

This will locate any active daemons and show you most or all of the command line invocation. The *-p* and *-b* switches are the most important because they tell you which port you will send or receive on (usually 3000) and how many buffers (typically 100) each daemon is using, respectively. Also of interest is the *-o* switch. Typically, DIS applications both send and receive, so the daemons are

set up to prevent reception of PDUs sent on the same host. The *-o* switch overrides this default setting so you can run an application that sends only and one that receives only, on the same machine. In these cases, the VCR is used to send replayed PDUs or to receive and record them. Be cautious here, you DO NOT want two applications running on the same machine if they will compete for daemon buffers. For example, if both programs are actively receiving PDUs, one will get some and the other will get the rest. This results in applications losing PDUs and causing entities to randomly drop out of the simulation.

If the *ps* command does not show the ***-o*** switch, the correct port, or enough buffers, use the following commands to stop the daemons

*/rem4/afa94/bin/sgi_recvd -p#### -q*

*/rem4/afa94/bin/sgi_sendd -p#### -q*

where *####* is the port number shown by the *ps -elf | grep* call. Now restart the daemons with the settings you need. If you use the ***-o*** switch, be sure to list it before the others as shown in the following examples.

*/rem4/afa94/bin/sgi_recvd -o -p3000 -b100 &*

*/rem4/afa94/bin/sgi_sendd -p3000 -b100 &*

The *&* means to run this process in the background and will return you to the prompt after the daemons are setup.

IMPORTANT: When using different hosts in simulations, you must have all the hosts' daemons on the same port or the simulations will not communicate with each other.

*A.2. Starting the VCR*

Some applications need to receive an initialization replay PDU before they will switch to their simulation replay clock. So, be sure to start the VCR after the receiving program is loaded. This is only necessary if you intend to use speeds other than standard.

When starting the VCR, you may want to include some of the command line switches it supports. Table A.2 lists all the switches and their effect on recording and playback. For record, the switches specify the filters that limit which PDUs the VCR will capture and set the initial default values for the record form. The site, host and entity IDs are set only by the command line switches. If a particular filtering switch is not used, the VCR will assume you wish to capture all PDUs regardless of its ID. For example, if the VCR is started as follows

*vcr -e2*

you would be preparing to record entities from any site or host with an entity ID of 2. If you use *-e, -l,* and *-u* together, only the *-e* will be recognized and just the specified entity ID will get recorded, not a range.

Table A.2. VCR Command Line Switches and Effects

| SWITCH | RECORDING | PLAYBACK |
|---|---|---|
| -i | Set Default Exercise ID to record | Set Exercise ID for Replay |
| -s | PDU Site ID to record | No affect |
| -h | PDU Host ID to record | No affect |
| -e | Single Entity ID to record | Not Implemented |
| -l | Range starting Entity ID to record | Not Implemented |
| -u | Range ending Entity ID to record | Not Implemented |
| -L | No affect | Repeat Replay of Same File |
| -E | Set Default Flag to Capture Entity State PDUs | No affect |
| -F | Set Default Flag to Capture Fire PDUs | No affect |
| -D | Set Default Flag to Capture Detonation PDUs | No affect |
| -M | Set Default Flag to Capture Emissions PDUs | No affect |
| -R | Set Default Flag to Capture Remove Entity PDUs | No affect |

*A.3. Making a Recording*

Now that you are familiar with command line switches, we can look at the record process. After starting the VCR with any switches you wanted, you will see the main VCR form shown in Figure A.1. The Selected Exercise information will be blank, the VCR's clock is set to the local time, and the speed controls are all set to the standard speed of 1.0.

To begin a recording, simply press the RECORD button. This will bring up the record input form shown in Figure A.2. Table A.3 shows each input field on the form, whether it is an optional or mandatory entry, the type of data to enter, and any error checking on the field. All input fields respond to the commands shown in Table A.4. The input fields will have default data in them or the data entered from a previous recording. To readily identify a recording, you should enter your user ID and a title describing the simulation. Also, you must enter an exercise ID and a file name. The VCR will append a ".log" to the end of the file name you enter, to make it readily identifiable.

Table A.3. Record Form Input Field Parameters

| INPUT FIELD | REQUIREMENT | DATA | ERROR CHECKS |
|---|---|---|---|
| USER NAME | Optional | String of any characters | None |
| EXERCISE ID | Mandatory | Integer | Only accepts integers. |
| EXERCISE TITLE | Optional | String of any characters | None |
| DIS VERSION | Not modifiable | String of any characters | None |
| FILE NAME | Mandatory | String of any characters | Error if left blank. VCR checks for duplicate file names. |

Table A.4. Input Field Key Reference

| KEY USED | RESPONSE |
|---|---|
| TAB | Move to the next field |
| ESC | Clear the field |
| ENTER | Accept data and move to the next field |
| END | Move cursor to the end of any text |
| INSERT | Toggle insert/type-over mode |
| HOME | Move to the start of current field |

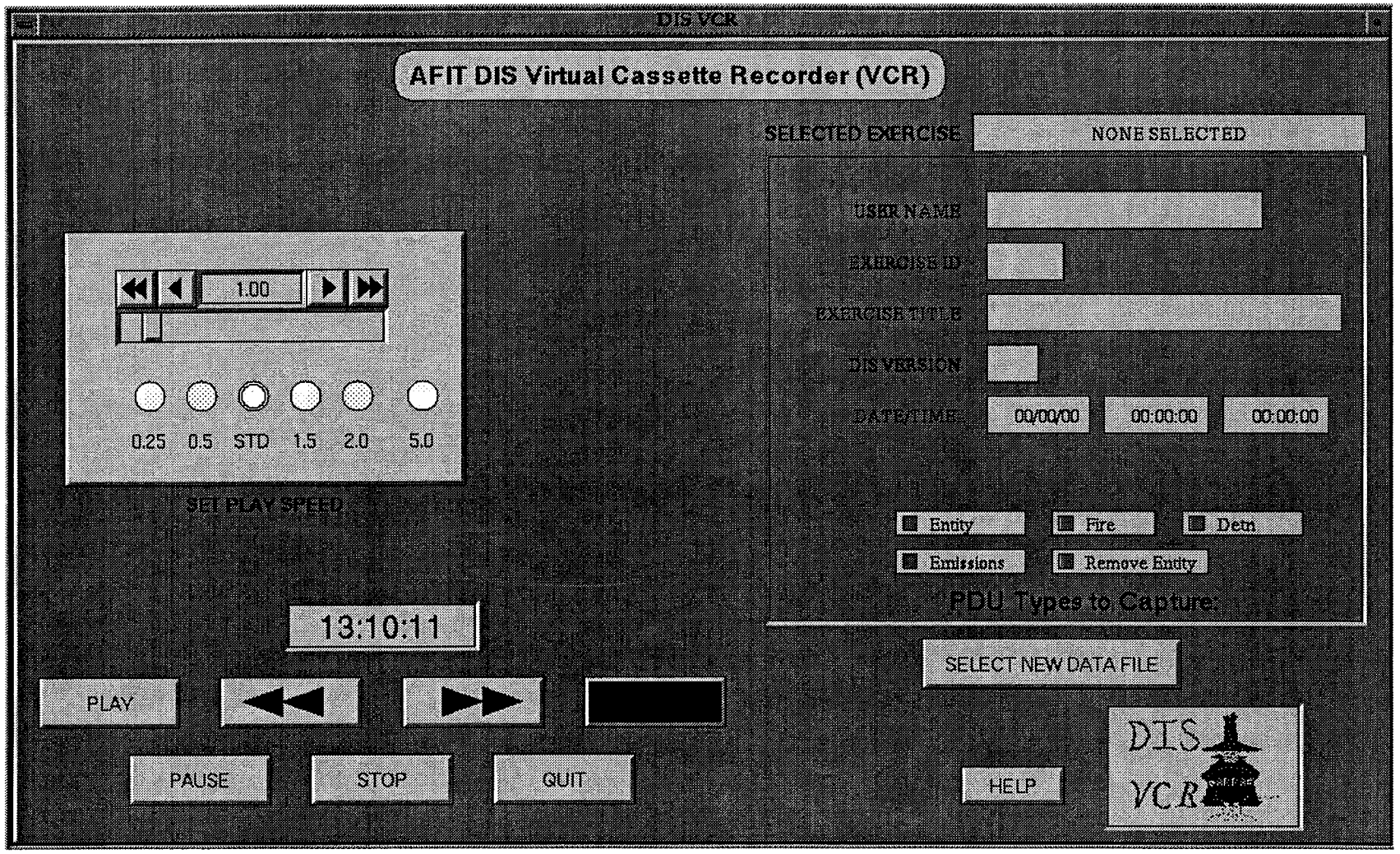



Figure A.1. Main VCR Form

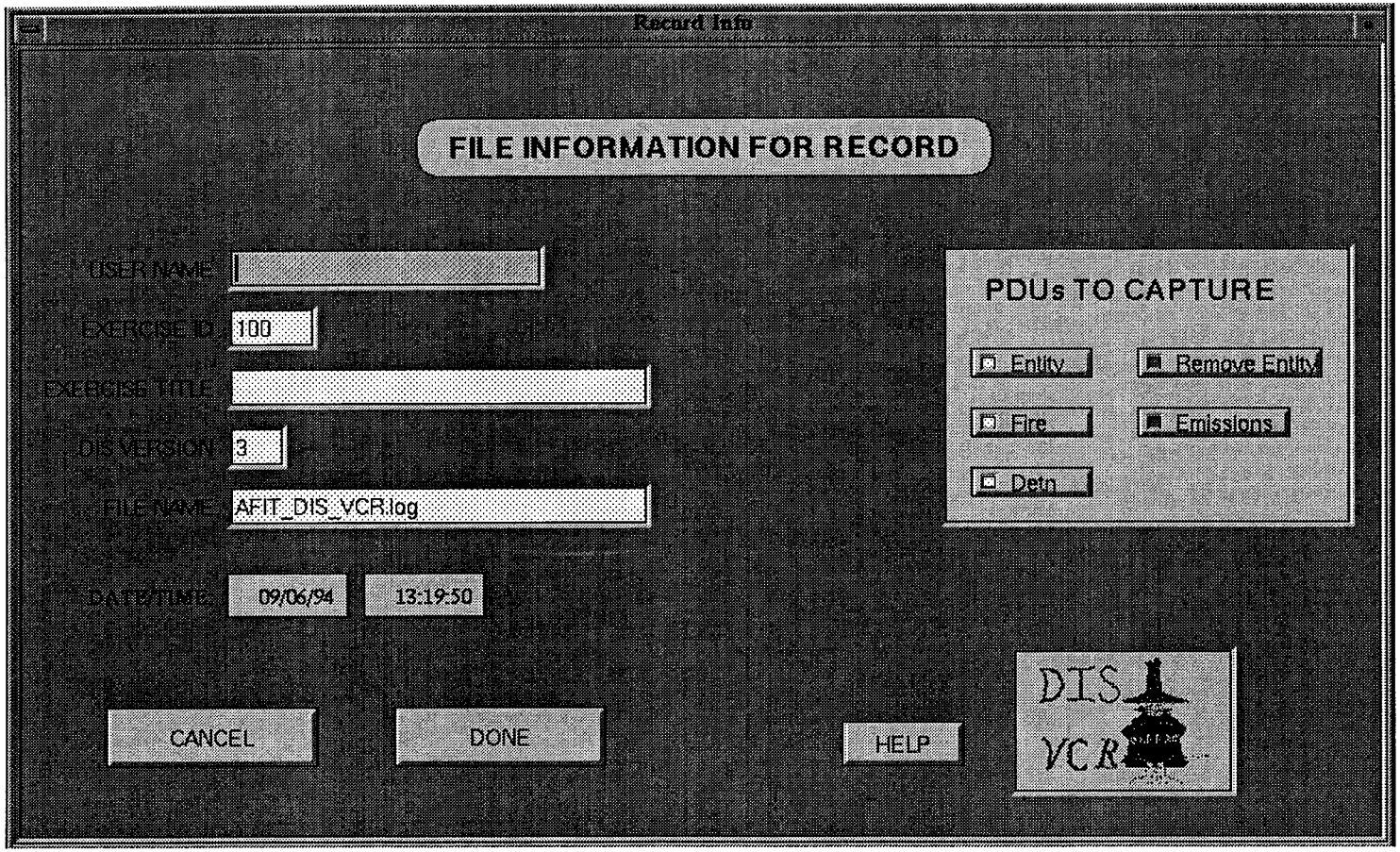



Figure A.2. Record Input Form

For initial recordings, the "PDUs TO CAPTURE" buttons will have the Entity, Fire, and Detn (Detonation) lit up. Use the mouse to select additional PDU types or to deselect those you do not want to capture. At least one PDU type should be selected, or the VCR will record nothing.

The VCR sets the site, host, and entity IDs to zero (meaning all) and the exercise ID to 100, unless you used the command line switches to change them. The record form does allow changes to the exercise ID but will initially display the one set by the command line or the default of 100. Set this value to the one agreed to by all simulation participants. To capture PDUs from any exercise ID, set it to zero.

To start recording, press the DONE button. The VCR will check for errors and inform you of any before proceeding. For example, if the file name already exists, it asks if you want to overwrite it. A no answer will return you to the form to change it. Once errors are resolved, the VCR closes the record information form and copies the new information to the main VCR form and PDU processing begins. The progress form, shown in Figure A.3, has a red progress bar and shows a PDU count. The progress bar and the VCR clock are updated on every 100th PDU and the count on every 10th. However, they all update on the first PDU so you know immediately if the VCR is receiving PDUs that match your parameters. If the record loop does not receive any matching PDUs in the first five seconds, a message dialog will warn you. Once you press the OK button, the loop will wait until a PDU comes in or STOP is selected.

To determine how long you have been recording, compare the start time shown in the information box to that of the VCR's clock. This is important because the VCR does not check for available space before trying to write data to disk. Check the available space before recording and monitor it periodically when disk space is limited. Table A.5 shows time versus disk space requirements for various entity counts to help determine if you have enough free space for recording.

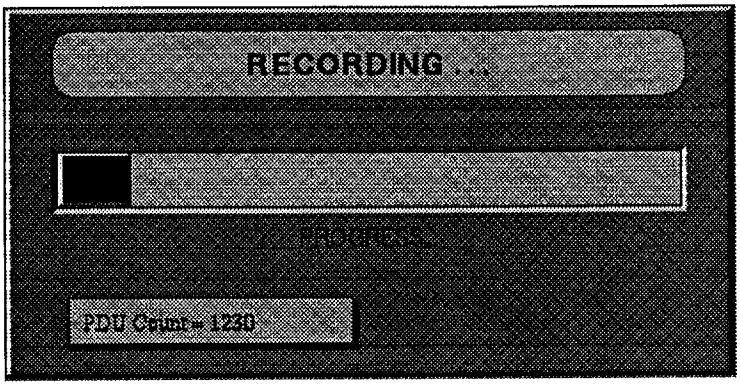



Figure A.3. Record Progress Form

Table A.5. Recording Time Versus Disk Space Requirements

| ENTITY COUNT | TIME | SPACE | PDU COUNT |
|---|---|---|---|
| 100 | 110 min. | 250 MBs | 1.5 Million |
| 500 | 10 min. | 21.5 MBs | 70,000 |

When the simulation is over, press the STOP button to end the recording. The final PDU block is written to disk and your new recording is rewound for immediate replay. From here you can exit the VCR or replay the data as described in the next section.

## *A.4. Replaying a Recording*

Unless you just finished recording the simulation exercise you want to replay, it will be necessary to select one. The next sections tell how selection is accomplished and walks you through the replay process, including the effects of speed controls.

*A.4.1. Select Exercise Form.* Before covering the selection procedure, it would be good to know what actions put you into the Select Exercise Form. There are two ways to invoke the select process. First, if you press PLAY and a file was not just recorded or selected, then the Select Exercise Form will automatically be displayed for you. Once a file has been loaded, the PLAY

button only affects that file. The second way to start a selection is to click on the SELECT EXERCISE button on the main VCR form.

Selection is accomplished by the form shown in Figure A.4. This form consists of a pick list, a file information box, various buttons, and two input fields used to set the data directory and file name filter or pattern. Each of these objects is covered in the following sections.

*A.4.1.1. The Pick List.* This list contains both directory and file names. If you click on a directory name (these have a "D" preceding them), the VCR switches to it and updates the pick list to reflect the change. If the items in the list do not fit in the window, a scrollbar is added to the side of the list to give you access to more entries. The scrollbar has a button you can drag to scroll the list or you can click on the bar above or below that button to page-up or page-down through the list.

When you select a file name, the file's information is displayed in the box on the right. This will help you identify which recording it is, when it was recorded, and what PDUs were originally captured.

*A.4.1.2. The Buttons and Inputs.* As you can see, there are four buttons included on this form. The CANCEL button allows you the exit the form and return to the previous state. Pressing LOAD means you are satisfied with your selection and want to play that file. If you entered this form from the press of the PLAY button, replay will begin immediately by displaying PRE-PROCESSING in a message. Otherwise, you are returned to the main VCR form and will have to select PLAY to begin. After the pre-processing message, a progress form, like that in record mode, pops-up and PDUs begin replaying. Just like record, updates to the progress form do not occur on every PDU. The frequency of updates will depend on the replay speed and the number of entities in a particular recording. Speed is discussed in detail in section A.4.2.

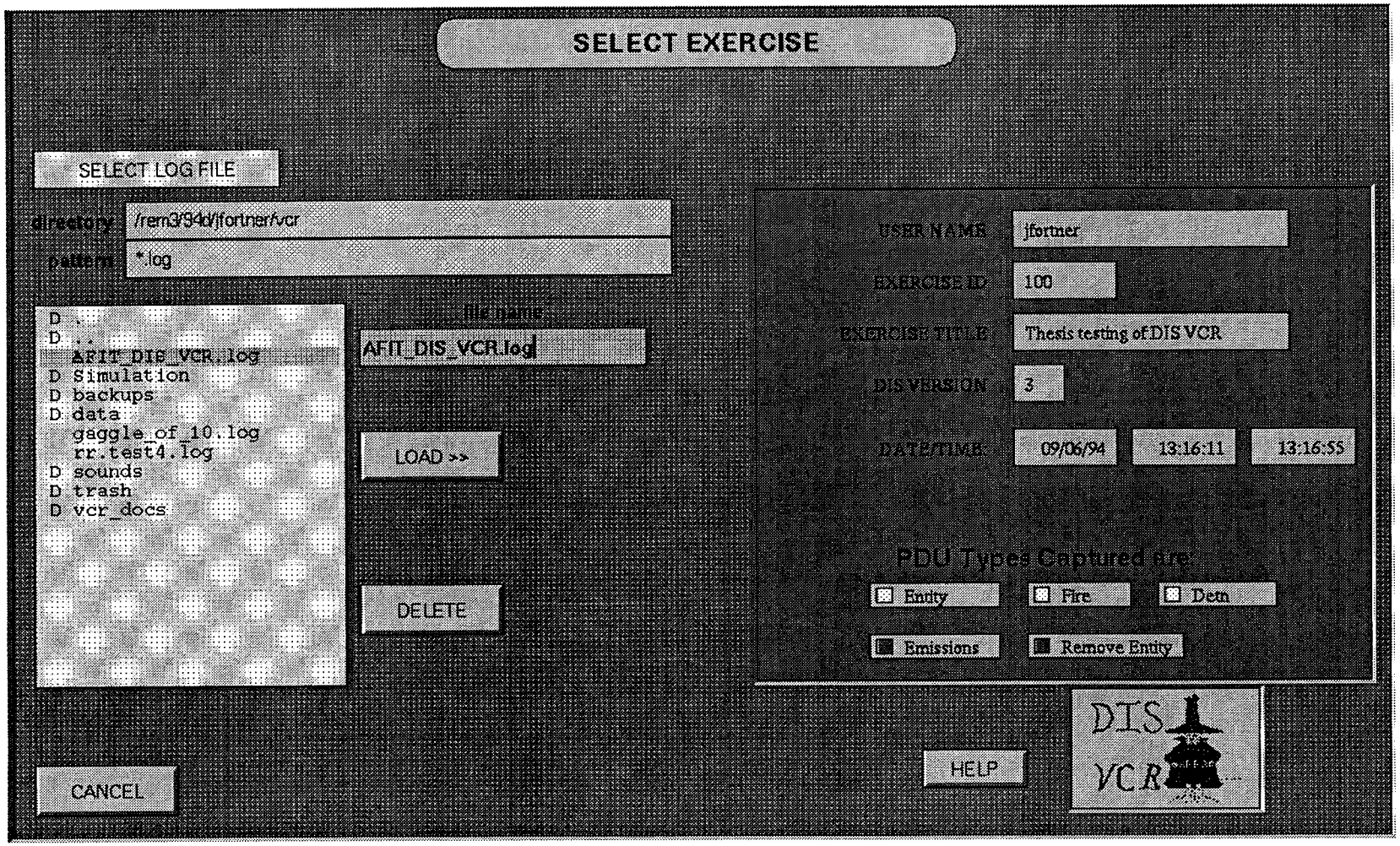



Figure A.4. Data Selection Form

The DELETE button allows you to purge old data from the disk. Periodically review all data files and backup or delete the ones you will no longer use, since VCR data files are typically large.

A press of the HELP button launches the form shown in Figure A.5, with specific help on the current form. This help system was borrowed from the Remote Debriefing Tool (RDT) and functions the same way. As with the pick list, if the information does not fit in the window, a scrollbar will give you access to the rest of the text. Buttons on the bottom act as an index to other help topics, press these buttons to load other topics. To exit help, choose the QUIT button.

Aside from clicking a directory name to change data directories, you can also type it in directly. Just click on the directory input field and enter the new one in the box that comes up. Limited error checking is done here, so try to be accurate. The file name pattern is changed the same way.

*A.4.2. Replay Speed.* When you selected a file, the VCR looked at the speed setting and started the replay at that speed. However, you can also change speeds during the replay as often as you choose. The receiving application's response to speed changes depends on whether or not it accepts and reacts to replay PDUs. What you will see in applications for slow or fast speeds and speed changes, is summed up in Table A.6. Typically, you will only want to use standard or fast speeds for applications that do not pay attention to replay PDUs.

Table A.6. Application Responses to Speed and Speed Changes

| SPEED OR CHANGE | APP. USES REPLAY PDU | NON-REPLAY APP. |
|---|---|---|
| Fast | Dead-reckons smoothly through PDU updates. | Entities take forward leaps. |
| Slow | Same | Entities drop off and reload. Dead-reckoning leads the entity. |
| Slow to Fast | Net delay on Replay PDU to App. causes some initial forward leaping but smoothes out. | Switches from dropping entities to forward leaps. |
| Fast to Slow | Same Net delay causes some entities to initially drop, but OK after reloading. | Switch from forward leaping to dropping off or reverse leaps. |

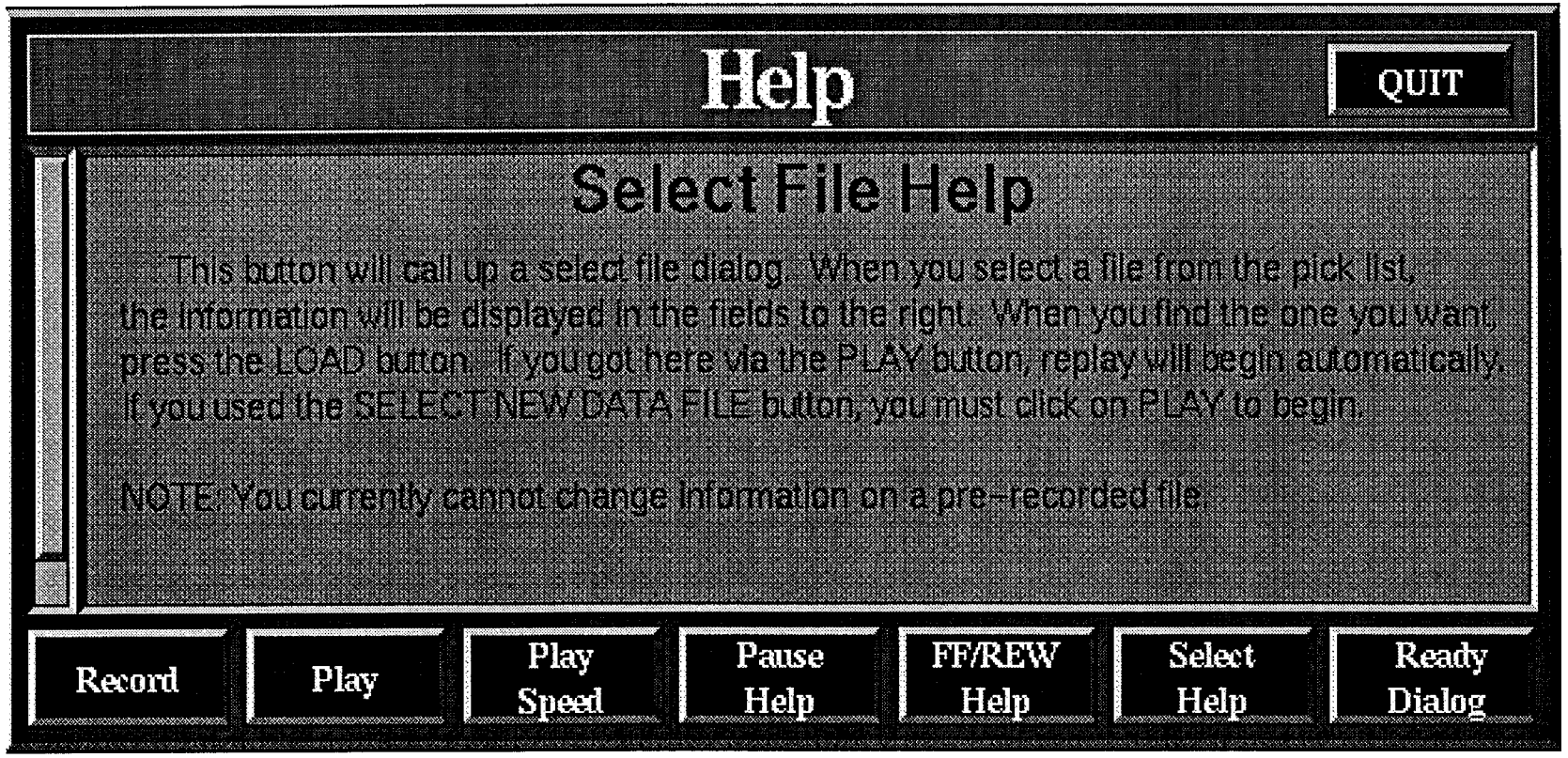



Figure A.5. Help Form

The radio buttons let you select from six fixed speed values ranging from 0.25 to 5.0. The selected button is lit up and its value displayed in the speed counter field. If the speed slider or counter buttons set the speed, the radio buttons are cleared and the new speed is displayed in the counter field.

*A.4.3. Fast Forward and Rewind.* Use the fast-forward (FF) or rewind (REW) buttons to move within the time range of the recording. To use these, a file must be loaded and stopped.

The FF button will advance the VCR's time until you press STOP or reach the stop time. If you did reach the end of the file, the VCR will place you at the start of the last PDU block and the time is set to the block's start time. The block will have at least one PDU and as many as 1000. When you stop in the middle of the recording, the VCR time will usually fall between the start and stop time of a block. When PLAY is selected again, PDUs are sent as fast as possible until PDU time catches the VCR clock, then play will continue at the selected speed.

The REW button is just the opposite of FF with one exception. When you rewind to the start or to a time within the first block, the file is pre-processed again before starting and the clock is set to the recording start time.

*A.4.4. Pause.* During a replay you may want to pause the VCR to examine a situation more closely. This is only possible for applications the respond to the replay PDU. When you select PAUSE, the VCR simply stops sending PDUs from the current block and transmits a replay PDU to inform the application. The application maintains the current simulation state and will not discard any entities from lack of PDUs. The pause form has a RESUME button that will restart the replay from where it paused.

*A.5. Wrapping it up*

Recording and replaying with the AFIT DIS VCR is nearly as simple as the home VCR. The real difference is the network setup and the selection of files instead of video tapes. Although the files use much disk space, they are valuable tools. The VCR gives you the capability to set up a complex demo, record it, then replay it over and over easily. It is used in engagement analysis with RDT or any simulation exercise.

*Bibliography*


1. Gardner, Michael T. *A Distributed Interactive Simulation Based Remote Debriefing Tool for Red Flag Missions*. MS thesis, AFIT/GCS/ENG/93-09. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1993.

2. "Standard for Distributed Interactive Simulation -- Application Protocols." Version 2.0 fourth draft, Institute for Simulation and Training, University of Florida, 4 February 94.

3. McCarty, Dean W. and others. "A Virtual Cockpit for a Distributed Interactive Simulation. (linked flight simulators)," *IEEE Computer Graphics and Applications, 14*: 49-55 (March 1994).

4. Adams, C. "DOD Embraces Warfighting Simulation," *Federal Computer Week, 7*: 8-10 (November 1993).

5. Cubic Defense Systems. "Computer Program Performance Specification for the Display and Debriefing Subsystem (DDS) of the REDFLAG Measurement and Debriefing System (RFMDS)," 20 June 92. CDRL B001.

6. Sheasby, Steven M. *Management of SIMNET and DIS Entities in Synthetic Environments*. MS thesis, AFIT/GCS/ENG/92D-16. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.

7. Guckenberger, Dutch. "DIS Research Tool: Enhanced Logger Playback Functions," *11th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Position Papers,* 1: 263-267. September 1994. Contract N61339-91-C-0091. STRICOM Summary Report.

8. Burke, Allen and Wentao Lyou. "DIS Test Tools BDS-D Architecture and Standards," *10th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Conference,* I: D57-D68. March 1994. Contract N61339-91-C-0091. STRICOM Summary Report.

9. Green, Keith L. "A Position Paper for Standard Datalogger Formats," *10th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Position Papers,* II: 546-555. March 1994. Contract N61339-91-C-0091. STRICOM Summary Report.

10. Juliano, Mark. "Standard DIS Data Logger Format," *11th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Position Papers,* 1: 127-131. September 1994. Contract N61339-91-C-0091. STRICOM Summary Report.

11. Miller, Jon T. "Time Scaling of Simulations in a DIS Environment," *10th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Position Papers,* II: 223-232. March 1994. Contract N61339-91-C-0091. STRICOM Summary Report.

12. "Enumerations and Bit-encoded Values for use with IEEE 1278.1-1994, Distributed Interactive Simulation--Application Protocols," March 1994.

13. Shneiderman, Ben. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley Publishing Company, 1987.

14. *Turbo Vision Programming Guide.* Borland International, 1992.

15. *IEEE Standard 754 for Binary Floating-Point Arithmetic.* ANSI/IEEE Std 754-1985, 1985.

16. Overmars, Mark H. *Forms Library v2.1 - A Graphical User Interface Toolkit for Silicon Graphics Workstations.* Utrecht University, Netherlands, Department of Computer Science, 1992. email markov@cs.ruu.nl.

17. Foley, James D. and others. *Computer Graphics: Principals and Practice* (Second Edition). Addison-Wesley Publishing Company, 1990.

18. "Working Group Minutes," *10th DIS Workshop on Standards for the Interoperability of Defense Simulations, Working Group Minutes,* III: 391-460. March 1994. Contract N61339-91-C-0091. STRICOM Summary Report.

*Vita*

Captain Fortner was born on 28 December 1959 in Oakland, Indiana. He graduated from John Marshall High School in Indianapolis, Indiana in 1978 and enlisted in the U.S. Air Force on 12 January 1979. Following his selection for the Airman Education and Commissioning Program in 1987, he attended the University of Missouri - Rolla. Afterwards, he attended Air Force Officers Training School and was commissioned on 1 October 1990. Acceling in his first assignment at Headquarters Ballistic Missile Organization as an Interface Control Project Officer, he was awarded the Air Force Commendation Medal. In 1993 Captain Fortner was selected to attend the Air Force Institute of Technology to complete a Master of Science degree in Electrical Engineering. He is married to Christine Lynn (Beach) Fortner of Indianapolis, Indiana. They have three children: Jonathan, Geneva, and Jared.

Permanent address:

6929 Hubbard Drive
Dayton, Ohio 45424-3533

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | December 1994 | Master's Thesis |

**4. TITLE AND SUBTITLE**
DISTRIBUTED INTERACTIVE SIMULATION VIRTUAL CASSETTE RECORDER (DIS VCR)
A DATALOGGER WITH VARIABLE-SPEED REPLAY

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**
Jonathan L. Fortner

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Air Force Institute of Technology
WPAFB, OH 45433-6583

**8. PERFORMING ORGANIZATION REPORT NUMBER**
AFIT/GE/ENG/94D-10

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
ARPA/PM
Warbreaker Simulation Engineering and Modeling
4301 N. Fairfax Ave., Suite 200
Arlington, VA 22203

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
Distribution Unlimited

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

The overall objective of the Distributed Interactive Simulation Virtual Cassette Recorder (DIS VCR) is to add a flexible replay capability to any DIS environment and specifically to the Remote Debriefing Tool (RDT). The DIS VCR's abilities include selective filtering of incoming DIS Protocol Data Units (PDUs), variable-speed replays, ability to pause, fast-forward, rewind, efficient data storage and retrieval, and an interface that simplifies the execution of those functions. The thesis includes a DIS VCR-compatible design for concurrent replay of audio extracted from signal PDUs and an extension to the replay design that supports unmodifiable rendering or receiving applications. For variable-speed replays, we created a scalable simulation clock and a new PDU (the Replay PDU). Applications modified for replays use the simulation clock to govern their dead reckoning algorithms while the DIS VCR uses it to control the timed release of stored PDUs. The Replay PDU communicates mode and speed changes between the DIS VCR and replay-modified applications. The DIS VCR's full functionality was successfully demonstrated at the 1994 AFA convention.

**14. SUBJECT TERMS**
DIS, Distributed Interactive Simulation, RFMDS, Mission Analysis, Datalogger, RDT, Remote Debriefing Tool, Replay

**15. NUMBER OF PAGES**
103

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

## GENERAL INSTRUCTIONS FOR COMPLETING SF 298

**The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to *stay within the lines* to meet *optical scanning requirements*.**

**Block 1.** Agency Use Only *(Leave blank)*.

**Block 2.** Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

**Block 3.** Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

**Block 4.** Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

**Block 5.** Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

| | | | |
|---|---|---|---|
| C | - Contract | PR | - Project |
| G | - Grant | TA | - Task |
| PE | - Program Element | WU | - Work Unit Accession No. |

**Block 6.** Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

**Block 7.** Performing Organization Name(s) and Address(es). Self-explanatory.

**Block 8.** Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

**Block 9.** Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

**Block 10.** Sponsoring/Monitoring Agency Report Number. *(If known)*

**Block 11.** Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

**Block 12a.** Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- DOE - See authorities.
- NASA - See Handbook NHB 2200.2.
- NTIS - Leave blank.

**Block 12b.** Distribution Code.

- DOD - Leave blank.
- DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- NASA - Leave blank.
- NTIS - Leave blank.

**Block 13.** Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

**Block 14.** Subject Terms. Keywords or phrases identifying major subjects in the report.

**Block 15.** Number of Pages. Enter the total number of pages.

**Block 16.** Price Code. Enter appropriate price code *(NTIS only)*.

**Blocks 17. - 19.** Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

**Block 20.** Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.