DEALING WITH A VARIETY OF RESOURCES IN DIS IMPLEMENTATIONS

John O'Reilly
Reflectone, Inc.
Tampa, Florida

ABSTRACT

Today's simulators involve many varied computational systems. Interoperating these devices is a key strategy for extending the life of current simulators. The Distributed Interactive Simulation (DIS) protocols solve the maze of interoperability.

These statements are well-known to the simulation industry, but the stark reality of the situation is that varied systems produce varied implementations of DIS. The full gamut of DIS involves the varied processors, varied operating systems, varied programming languages and structures, varied interface hardware, varied coordinate system implementations, and varied data base formats. Realizing these six areas simultaneously is a particularly demanding chore. This paper attempts to show how one implementor went about producing code for DIS, seeking to provide reusable code in the process. The lessons learned from this venture are discussed.

Using a PC-based radar simulation system as the baseline, the paper discusses the research and development of DIS in this varied environment. Although a radar appears to be a "receive-only" entity on a DIS network, in order to locally test such a system, test vectors (or PDUs in the DIS parlance) must be generated. Thus, the baseline requires some way to construct test vectors, such as through semi-automated forces (SAFOR) or Computer Generated Forces (CGF) generators. A limited CGF for the required purposes is described in the paper.

Other steps in the implementation include actually producing, transmitting, receiving, and displaying the CGF state vectors. Production involves coordinate conversion schemes, PDU receive and transmit functions become as dissimilar as their associated processors, and display techniques require limitations in the scope of what can be displayed. So, the paper surveys network I/O techniques and selects the correct one for the radar simulation. The last stage (displaying) requires a filter of the vectors since all processors (and especially the PC in question) have limits in terms of computing power.

Intermediate steps in the full implementation of a DIS system involve determination of correct protocols sent from the CGF and the use of terrain and feature data bases. Both of these areas are also discussed including the fields of network analyzers, DMA maps, and Project 2851 SIF.

The paper points out that, although realizing a DIS interoperation can be straightforwardly done, care must be taken to understand that there is more to the "varied" problem than just the obvious processor incompatibilities.

BIOGRAPHICAL SKETCH

John has worked in the simulation industry for 25 years beginning in the CAE world of circuit simulation using SCEPTRE and SPICE. He has been in the employ of Reflectone for ten years working variously as a controls engineer, computer systems manager, acoustic simulation engineer, radar system engineer, and R&D investigator. Research interests and experience include artificial intelligence, radar simulations, data base design, and DIS.

John holds a BSEE and MSEE from the University of South Florida in Tampa and has done post-graduate work in multi-disciplinary simulation.

DEALING WITH A VARIETY OF RESOURCES IN DIS IMPLEMENTATIONS

John O'Reilly
Reflectone, Inc.
Tampa, Florida

INTRODUCTION

Interoperability is both the blessing and the curse of the Distributed Interactive Simulation (DIS) implementor. Allowing us, by definition, to communicate seamlessly between networkable computers using pseudo-real-time protocols provides many coding advantages:

- Drop-in code – We can develop code once for many platforms (although this is not necessarily accurate, as we will see),
- A functional separation of decomposition is readily available,
- Information hiding design methodologies fit in well from processor-to-processor, and
- Standard protocols are easier to debug, assuming a network analyzer is available.
- A system with the DIS interoperability capability is an "easy sell" in the competitive simulation industry.

But DIS has its disadvantages, too. We will discuss these problems areas with an example.

The 1992 I/ITSEC conference's state-of-the-art theme involved a demonstration of DIS. Over twenty corporations were involved in defining, discussing, implementing, and demonstrating the use of DIS protocols during the three-day affair. This was the first major exhibition of DIS's capabilities in a public forum – a vast undertaking in its own right – and the first undertaking by this researcher of any coordinated network simulation. Reflectone chose to put a "listen-only" Digital Radar Landmass Simulation (DRLMS) in the demonstration network. Although this listener was only expected to receive network packets, the basic process of appearing on the network was found to be equivalent to performing a full-scale interoperability simulation. This concept will become more apparent as we discuss the implementation and its attendant problem resolutions.

In order to place a radar simulation system on the I/ITSEC network, a full set of DIS Protocol Data Unit (PDU) software was required. Entity State, Fire, Detonation, and Collision PDUs would have to be sensed from the network, decoded accurately, and converted to the pre-existing radar simulation's coordinate system. The radar simulation was hosted on a PC-compatible machine running the standard one megabyte operating system. No prior experience on this use of networking protocols, especially UDP/IP (User Datagram Protocol/Internet Protocol, as defined by the DIS Standards), was available for the host computer. As an additional sidelight, the I/ITSEC demonstration made the Project 2851 Simulation System Database Interchange Format (SIF) format database the standard. This aspect added more complexity to the DIS Interoperability Demonstration problem, although SIF is completely distinct from DIS in principle.

The scope of the DIS standards were somewhat reduced by popular opinion (e.g., limited set of PDUs allowed on-line, simplified coordinate system, and broadcast-only packets employed), but the mainstays of DIS were left in place (i.e., full entity state PDUs, UDP/IP protocols, dead-reckoning algorithms, geocentric coordinate system).

Having inroduced the scope of the problem (DIS Interoperability Demostration using a DRLMS), the remainder of this report will discuss the implementation of the DIS and DRLMS requirements.

VARIED RESOURCES ARE NEEDED TO MEET REQUIREMENTS

The Institute for Simulation and Training (IST, the organization tasked with both putting together the DIS Standards and the 1992 I/ITSEC Interoperability Demonstration) realized that some means of testing DIS systems was necessary. Thus, IST generated a test procedures document which all participants in the demonstration were required to pass before being certified to interoperate on the I/ITSEC network. The document also provided processes by which IST would verify passage of the test procedures. However, this document and its processes were not available early enough in the development cycle to provide any testing methodology by the participants. Thus was born the "FLY" program.

FLY is a semi-automated forces generator (SAFOR) specifically designed to produce accurate and (for testing purposes) repeatable DIS PDUs. It was decided that it would be nice to host the FLY routine on a processor unlike the target radar processor, so FLY was generically designed and implemented in C to operate on both a VAX system and a Motorola UNIX system. The choice of the processors was foremostly made because of availability, but secondarily made to locate and understand how DIS *should and could be implemented for true interoperability.*

The widely varying aspects of the VAX with the VMS operating system and the Motorola Delta box with its UNIX operating system injected some unexpected difficulties. The first concern was raised by the speed of all of the processors. Relative CPU horsepower is certainly a concern for any networked design due to throughput considerations – Is the processor capable of transmitting and receiving and decoding DIS PDUs in a timely fashion? It turned out that the PC was probably the most efficient in this aspect, because of its single user operating system as opposed to the multi-user VMS and UNIX environments. In order for VMS and UNIX to be as sophisticated as they are, they must provide packet trapping mechanisms well above those of the DOS-based PC operating system. The major concern here was thus to reduce the effect of the operating system on the application's design limitations. The FLY and Radar simulation applications should implement the dead reckoning and coordinate transformation algorithms as efficiently as possible, cutting corners as much as possible. Assumptions of limitations will be discussed more in the next section.

The varying host processors necessary to implement any DIS design also have internal hardware variances. The hardware implementation of floating point numbers is a distinct problem area. The DIS standard quotes the IEEE *Standard for Binary Floating Point Arithmetic* (IEEE 754-1985) as the required standard. This standard is implemented in many processors directly (in this case, the Motorola 680X0 and Intel 80X86 processors) but is not implemented in others (specifically, the VAX architecture). The VAX floating point format matches for single-precision floating point numbers, but has two double-precision formats, neither of which follow the IEEE 754 standard. Reformatting of floating point values from and to the DIS standard is not necessarily an easy transformation. For the VAX, a conversion from one of the formats to the other and a simple multiplication of the result by four is needed. Other processors may require some difficult bit manipulation routines to perform the conversion.

Likewise, host processors vary widely in hardware network interface capability. The main concern here is in regard to conversion of the network physical medium to thinnet, thicknet, or AUI cable connectors. This conversion is relatively straightforward, although costly under some mechanisms. Another network concern is the capability of a processor to accept true "Blue Book" Ethernet versus IEEE 802.3 protocols. Again, this is generally not a concern since interfaces can be switched between the two (Ethernet vs. 802.3), usually via a software protocol converter. But, PC network interfaces are generally Ethernet only.

At a higher level, implementation of DIS standards can be affected by programming languages and the standards for programming. For example, although the use of CASE tools is highly effective in code generation, real-time and object-oriented CASE systems are not available for PC assembler language (in which the pre-defined radar simulation was written). Likewise, for the case of non-standard floating point formats, the FORTRAN language may not be appropriate to implement the necessary bit manipulation for IEEE 754 conversions. This is usually not a major stumbling block, however.

Various coordinate system implementations were expected to be involved in the demonstration, because simulations would be running on varying hardware platforms simultaneously during during the networked demonstration sessions. For example, a simulation running on one host may use a world coordinate (Lat/Lon) positional system and another processor may be implementing a simulation using topocentric coordinates (X, Y, Z). These various implementations exist quite often in the flight simulation arena due to varying requirements for each simulation: one system may require a very large gaming area in which a geodetic coordinate system is necessary, whereas another system running in a smaller gaming area would only need a "flat earth" topocentric system. The DIS interface for these two examples would *obviously be quite different in design, although hooks could* be installed in the code to provide access to the correct coordinate conversion routines dependant upon the coordinate system locally used. So, regardless of which coordinate system is employed in a simulation, the DIS PDU packing/unpacking routines would convert to/from geocentric according to the requirements of the local simulator.

Databases are utilized heavily in DIS implementations. Terrain, features, and moving models for visual and sensor systems all use these databases and they will use them differently, on differing processors, at differing levels of detail, and for differing reasons. Visual systems need highly detailed database inputs; low-level "fishing" radars need much less accurate database information. The correlation of these databases is required so that, say, a bridge appears at the same relative location on a visual IG as it does on a Digital Radar Landmass Simulation. Also, the correlation between the databases and entity positions and orientations must be accurate in order to keep ground vehicles from flying above the terrain or flying entities from appearing to burrow through the terrain. This correlation is mainly a function of accuracy and consistency in coordinate transformations. If one high-detail system performs double-precision mathematics, its results will appear more realistic than one using lower precision algorithms. In fact, costly (in terms of CPU horsepower) floating point algorithms provide better correlation to databases. The choice of conversion algorithm may also affect correlation accuracy. If two hosts compute the same value differently, their position updates to the other hosts may not be well-correlated.

IMPLEMENTATION DETAILS AND DIFFICULTIES

In order to provide some more detail to the above-mentioned concerns, we address some realizations.

The FLY and radar simulations were designed to be efficient regarding network throughput. Assumptions have to be made in any simulation design, so that difficult requirements will fit within processing constraints. This means that corners must be cut (or at least shaved) in the design of DIS processes. Corner cutting must be selective, of course, in order to not reduce the realism of any simulation. Filtering of the PDUs is the most practical way to reduce both host network traffic and host resources. The filtering must be a bottoms up one; that is, the quickest and earliest filtering (or ignorance) of PDUs is essential. Since all PDUs are broadcast (in the current implementation), none could be ignored on network address alone, although that would have been the earliest filtering possible. Non-Ethernet packets could be ignored by low-level, raw interfaces, but VMS and UNIX do not afford this direct capability. However, filtering at this level was implemented on the PC. Also, non-UDP/IP packets fit the same mold as those of the non-Ethernet form. The demonstration used a single exercise ID, so filtering could not be performed at that level, either. Filtering finally can be implemented in any of the varying resources at the PDU type level. For example, the radar simulation was designed to ignore any PDU which was not an entity state, and the FLY routine ignored any PDUs not specifically designed for the demonstration (i.e., entity state, fire, detonation, and collision). The next stages of filtering can be performed on the types of entities provided within the PDUs. Thus, the radar simulation would ignore "small" entities such a dismounted infantry and light vehicles.

The radar simulation system was written using a flat earth topology. That is, the spherical earth was locally flattened (via a Sanson-Flamsteed transformation) into an X-Y coordinate system. This was dictated by the host flight simulation. Conversely, it was found to be most effective to design the FLY routine to operate in the geocentric coordinate system. Computing the DIS required geocentric positions was effortless and accurate, because flight models do not generate new positions, but rather generate coordinate system independent velocities.

This varying choice of coordinate system obviously injects some correlation errors. Reflectone's implementation was not too concerned with these errors, because the radar simulation was low in detail; so the issue remains open. Solutions remain to increase computational accuracy (double precision floating point evaluations) and to assure consistency in the choice of conversion algorithms.

- Network I/O is much like radio transmissions: A radio transmitter is much easier to implement than a receiver, because there is no need to consider noise. Likewise, a network write command is much easier to design than a network read command because of all the extra (noisy?) packets and their asynchronicity in the receive mode. Implementation of the send/receive mechanisms depends upon the particular processor in use.

In practice, the implementation of network I/O on a PC can be handled in several ways. The most direct method involves proprietary software which is compatible with a single type of network interface card. This method was rejected as too costly in the long run because of being locked into a particular interface. As an alternative, there are public domain PC network drivers available from Clarkson University, the clearinghouse for a large set of consistent drivers. The consistency comes from a public domain specification for the drivers from a company named ftp Software. Each interface manufacturer writes the driver for his card and places the driver in the public domain. The interface to the drivers is assembly language – a perfect match for the radar simulation. The packet drivers are raw drivers – the user must perform bottom-level interfaces to the network's physical layer. For a PDU transmission, the user must place the appropriate Ethernet, Internet Protocol (IP), and User Datagram Protocol (UDP) headers and trailers around the DIS PDU and send this whole packet to the Clarkson driver interrupt with a "send message" command. For packet receipt, the following extended operations must be performed:

1. Locate the Clarkson driver and ensure it is installed,
2. Request device information from the driver,
3. "Access" the driver by supplying a receiver routine's address,
4. Initialize a packet-ready counter to zero,
5 On each pass through the simulation, check the packet-ready counter,

6. If non–zero, read the raw packet, and decrement the packet–ready counter. If zero, there are no packets ready and continue the simulation.
7. Check the packet for correct Ethernet headers,
8. Check whether the packet is an ARP request. If so, request the Ethernet address of the local interface card from the Clarkson driver, format an ARP reply packet, send the reply, and continue the simulation.
9. Check the packet for IP and UDP flags, and check the UDP port number for correctness.
10. Check the packet for smallest DIS PDU length,
11. Check the exercise ID and DIS Standard version for correct values,
12. Accept the packet as a DIS PDU.

ARP (Address Resolution Protocol) was required by the DIS Interoperability Demonstration. ARP involves a separate host requesting the Ethernet address of a known Internet node. The ARP request is transmitted in broadcast mode, is received and decoded by the appropriate node, and replied to by that node only. Thus, even though it was initially thought that the radar simulation running in the PC was to only receive packets, it was required to transmit an ARP reply also.

Most other implementations (including the costly PC version) involve direct "socket" library calls to send and receive DIS PDUs. Socket libraries handle the majority of the interface protocols from Ethernet, to IP, to UDP, including broadcast addressing capabilities. Socket send routines packetize the PDUs and receive functions remove the headers and trailers from packets returning only the PDU information. But, even these socket libraries require some extensive coding in practice. Once again, the send mode involves fewer steps than PDU receipt, but both involve the concept of binding a socket to a particular protocol (UDP/IP in this case) under a particular port number (DIS Interoperability Demonstration chose port number 3000.). Within the simulation loop, a "select" function determines if and when a probable PDU exists in the read buffer or when a socket is available for sending a PDU. We say "probable" PDU, because even with this higher level of coding at the socket library level the minimum PDU length, exercise ID, and DIS Standard version must still be checked in the returned packet to ensure that the packet is a PDU. Most implementations, including the VAX and UNIX implementations mentioned herein for the FLY routine, automatically check incoming packets for ARPs and reply as needed. With the use of socket libraries, however, comes a new set of obstacles. Broadcasting is a relatively high overhead network concept. Every broadcast packet must be received by every other node on the local area network. Thus each node's throughput is decreased because of the added packet count. Consequently, some systems (notably UNIX System V) disallow broadcast transmissions except by the "super–user." Only the root user, therefore, can operate on the DIS network. Broadcasting can clog a LAN very quickly with what is known as a "broadcast storm." These storms occur when routers amplify the number of broadcast packets to their local systems due to address servers re–sending ARP requests back to the broadcasters. This situation becomes unmanageable in wide–area networks.

Specific to the radar simulation, other design criteria concerned the landmass database and the placing of target returns on the database. PDU filtering was performed as described earlier, but with a twist. All current entities were displayed in a menu prior to operation of the simulation so that an ownship entity could be selected. This allowed the radar simulation to logically attach to any entity on the network and then display other entities within radar range and elevation. Concerns arise from even this basic selection. Filtering had to be done to another level, because the simulation allowed for only twelve targets (but there were many more displayable entities than twelve on–line at any given time). Adding to this complexity was the fact that entities tend to come and go during any demonstration. That is, some nodes on the network would bring in new entities asynchronously and others would drop off the network, especially during testing prior to the show. This was not taxing on the radar design, since lost entities simply would not be displayed any more, and new entities would be ignored. I counted a maximum of 212 entities at one point during testing which were actively filtered with little problem by the supposedly underpowered PC.

A larger problem existed when the ownship dropped off–line. The radar had to be redesigned drastically to alert to this condition. When the condition occurred, the redesigned system halted the radar display and went back to the "ownship selection mode" where the user needed to select a new ownship. Also, the algorithm for an entity dropping off the network remains rudimentary, because the DIS Standard lacks a protocol to indicate loss of an entity. This algorithm assumed that if any entity did not update itself for five seconds (the maximum time during which an active entity must send an entity state PDU as defined in the standard), then it is gone. This adds quite a bit of logic to any DIS implementation.

Another well–documented concern regarded dead reckoning (DR). In order to provide the most accurate DR, it was found that if everyone reckoned in the same coordinate system, less positional/attitudinal error was seen on displays. Not everyone on the demonstration network was dead reckoning in the same coordinate system, although it was suggested that geocentric DR be used.

The radar's landmass database had to be pre-conditioned from SIF to one acceptable to the radar simulation. This is basically a local requirement for any simulation (e.g., visual vendors must pre-format the database to their specifications, etc.). Database pre-conditioning is a major DIS concern. The final SIF database was not available until late in the test sequence (August 15 for a November 1 use date), but preliminary databases were available. So a relatively minor reconditioning effort was all that was needed to install the new database. This brought about the design need for a general-purpose SIF conversion routine, so that when an updated SIF tape was delivered the routine could simply massage the data at the user's leisure.

SUMMARY/CONCLUSIONS

The use of something as conceptually simple as a "listen-only" radar simulation on a DIS network is much more involved than one could expect. Testing of a conceptual design is the most vital concern, one which requires at least one other host node to generate or receive DIS PDUs. Other concerns involve host capabilities (e.g., horsepower, math formats, networking capability), programming languages and design methodologies, and the use of Project 2851 SIF or other formatted terrain/feature/model databases. All of these criteria (or features) must be designed into any new DIS system, and should remain available for later use since the DIS *Standard is an evolving document.*

REFERENCES

Military Standard, "Internet Protocol Specification", MIL-STD-1777.

Military Standard (Final Draft), "Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation", Institute for Simulation and Training, 1991.

Military Standard, "Standard Simulation Data Base Interchange Format for High Detail Input/Output and Distributed Processing", PRC, Inc.

Institute of Electrical And Electronic Engineers (IEEE) Standard, "Standard for Floating Point Numbers", IEEE 754-1985.

DISCLAIMERS