The TAVERNS Emulator: An Ada simulation of the Space Station Data Communications Network and software development environment

by Dr. Norman R. Howes

Introduction.

The Space Station DMS (Data Management System) is the onboard component of the Space Station Information System (SSIS) that includes the computers, networks and software that support the the various core and payload subsystems of the Space Station. Although some of the DMS software runs in the subsystem computers, the subsystem computers themselves are not considered to be part of the DMS. Also, the applications software that is specific to a subsystem (e.g., the Communications and Tracking Subsystem) is not considered part of the DMS.

The various core subsystems (there are 22 of them) are to be implemented on Standard Data Processors (SDPs). This does not imply a standard computer has already been selected for this role but that all subsystem computers are to have the same instruction set architecture (ISA). It is also possible that a single SDP may host more than one subsystem. A diagram of the DMS together with the various subsystems is shown in Figure 1.

Figure 1 shows each SDP connected to a Core or Payload network via a Network Interface Unit (NIU). The NIU is itself a computer, probably with the same ISA as the SDP. The NIU hosts the Network Operating System (NOS) component of the DMS. On the other hand, the SDP hosts the application software for one or more subsystems. The SDP has an operating system (OS) of its own that is some times referred to as the local operating system (LOS).

A great deal of the DMS software resides in the SDPs. Those parts of the DMS software that provide the file management capability and the data base management capability are examples of DMS software that resides in the SDPs. Most of the DMS software that supports the actual transmission of data (both datagrams and virtual circuit transmissions) resides in the NIU and is referred to in general as the NOS.

The Ada packages of services available to the core or payload application programmer for (1) network communication, (2) file management, (3) database management, (4) data acquisition and distribution and (5) crew workstation services are documented in the DMS Test Bed Users' Manual (NASA/JSC No. 22161).

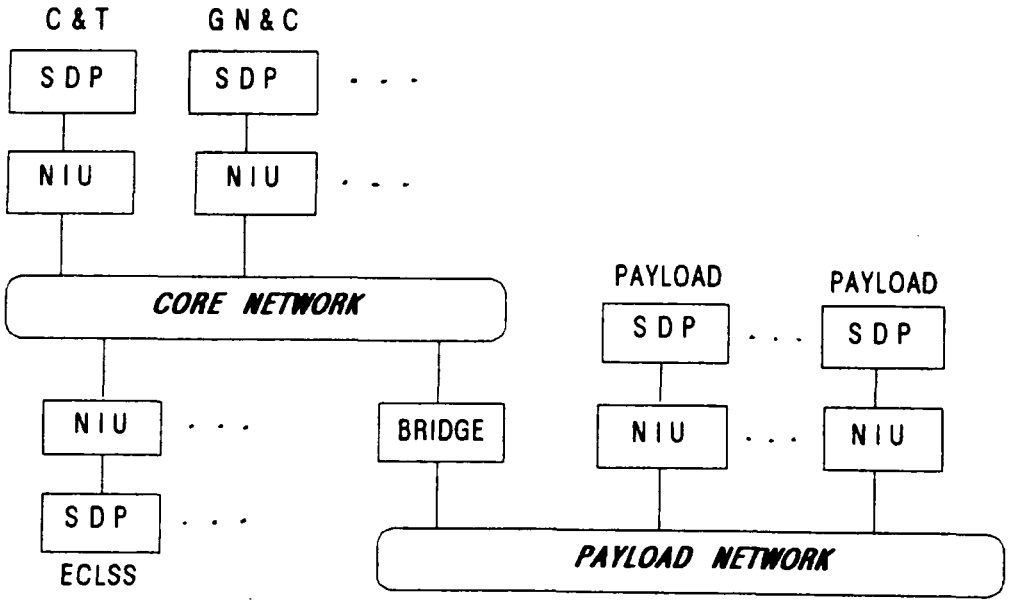

Figure 1

The TAVERNS Concept.

TAVERNS is a distributed approach for development and validation of application software for Space Station. The acronymn TAVERNS stands for Test And Validation Environment for Remote Networked Systems. The TAVERNS concept assumes that the different subsystems will be developed by different contractors who may be geographically separated.

In this approach, each software development contractor for the station will be provided with a miniature version of the Space Station DMS complete with three SDPs. One of the SDPs is for developing the subsystem software, one hosts the Displays and Controls software and the third hosts a simulation of the network core subsystems (e.g., ECLSS, C&T, GN&C, etc.) and the network loads. A diagram of such a TAVERNS DMS Emulator is illustrated in Figure 2.

The SDPs and NIUs on this mini Space Station DMS will host the same Services as the real DMS, so to the applications programmer, it will appear that the entire Space Station DMS environment is present. In turn, these TAVERNS systems will be interfaced with the Space Station SSE (Software Support Environment) and there will also be a TAVERNS on the station. In this way, software can be developed and checked out by different contractors at different locations. Completed and tested applications can then be transferred to the SSE for validation.

After an Initial Operational Capability (IOC) has been achieved for the Station, new validated software modules can be transmitted to the station where they will be revalidated on the onboard TAVERNS before being placed in service.

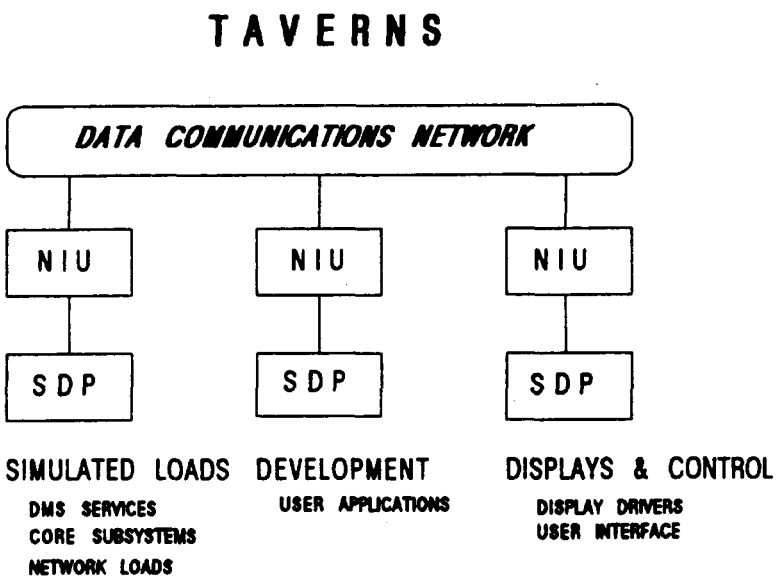

Figure 2

The TAVERNS Emulator is an Ada simulation of a TAVERNS on the ASD VAX in Building 16A. The software services described in the DMS Test Bed Users' Manual are being emulated on the VAX together with simulations of some of the core subsystems and a simulation of the DCN. The TAVERNS Emulator will be accessible remotely from any VAX that can communicate with the ASD VAX.

The purpose of this simulation is to (1) test the functionality of the DMS Services as documented in the DMS Test Bed Users' Manual, (2) provide a DMS software environment that is consistent with the one described in the Users' Manual where subsystem test bed developers can attempt to interface their subsystems with one another and (3) provide an environment where the TAVERNS concept itself can be evaluated and improved.

Ada features of the TAVERNS Emulator

Purposes (1) and (2) above are of special interest to the software engineer or programmer who will be designing or coding programs in Ada. In a way, the DMS Services as described in the Users' Manual can be thought of as an extension of the Ada language for distributed applications. These services are actually packages of utilities (subprograms or tasks) for performing certain operations such

as transmitting a message from one subsystem to another or opening a file at a remote node (subsystem) and reading records from it.

These utilities are not only written in Ada but their intent is to operate on Ada data structures in a transparent manner. For instance most data communications networks with which anyone has any experience only allow a user to transmit data in a single predefined format such as in ASCII or binary packets. These packets may be as small as a single character as with asynchronous communications or a large binary block as with synchronous communications. On the other hand, when using an object oriented language like Ada what one would like to do is transmit an entire Ada object without first having to convert it into an ASCII or binary string.

The DMS Test Bed Users' Manual describes Ada oriented utilities such as this for transmitting Ada objects, writing Ada objects to remote files, etc. In fact, the Users' Manual describes even higher level services that use various Ada objects. For instance, one of the most frequently needed communications capabilities for the Space Station core subsystems is a request to read a set of measurements. By a measurement is meant the reading of a certain sensor (such as temperature or pressure) or the determining of the state of something (such as a valve being open or closed).

For most applications more than one measurement needs to be read at a time. The DMS provides a service for assigning logical "set names" to a set of measurements and a service for requesting the reading of a whole set of measurements by issuing a single command. When such a request is made, the DMS returns all of the readings in an Ada structure that depends on variant records that is optimized for this application and is independent of the Ada types that correspond to the various measurements in the measurement set. Furthermore, not all the measurements in the set have to be located at the same node on the network.

The Ada Simulation

The Ada simulation is being designed to run in a single VAX with access from another VAX. The intent here is for the VAX in which the simulation runs to represent all of the TAVERNS system except the SDP node and the other VAX represent the SDP. The user of the TAVERNS Emulator can develop Ada code on any VAX and then link to the TAVERNS Emulator VAX via the simulated DMS Services thereby simulating the way a contractor would develop Space Station applications software on the SDP node of a TAVERNS system. A diagram of the Ada simulation is shown in Figure 3.

The Ada simulation is being developed in two phases. The first phase configuration is shown in Figure 3. The first phase consists of a demonstration in

which the user's only participation is that of responding to prompts on the display. In this configuration, the display handler software consists of the Ada procedures that control the menus from which the user chooses options during the demonstration and the various screens the user sees as part of the demonstration.

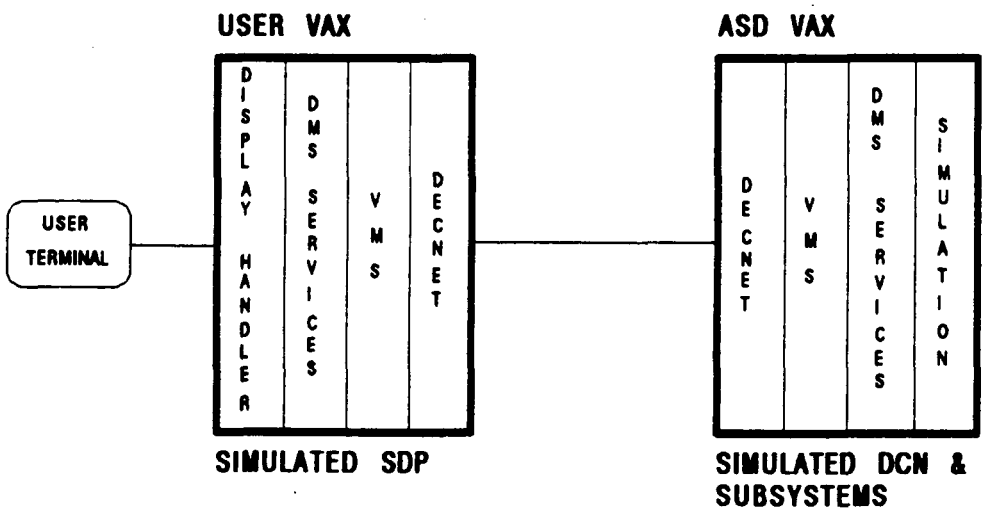

Figure 3

For the most part the DMS Services software maps the various DMS service commands onto the appropriate VAX VMS or DECNET service or combination of services to accomplish the specified DMS service. Where no existing combination of VAX services will accomplish a DMS service the necessary Ada subprograms are being developed.

The simulation software that runs in the ASD VAX is based on deterministic models of three subsystems for the first phase demonstration. These are the Communications and Tracking (C&T) Subsystem, the Environmental Control and Life Support (ECLS) Subsystem and the Mass Memory Management (MMM) Subsystem. The C&T and ECLS subsystems are modeled as a set of measurements. During the demonstration the values of the measurements change in accordance with a predetermined algorithm. The MMM subsystem is modeled as a set of data structures that relate which files belong to which subsystems and how files are related to each other through directories.

The demonstration consists of prompting the users for which sets of measurements the user wants to see displayed; using the DMS Services to request a reading of these measurements across the network (across the physical network between VAXs), to reply to the request at the other end of the network and to build the

display in response to the user's request; and using the DMS Services to handle the supporting file management functions (the globally known measurement names are stored on a remote file). This last feature may seem a bit contrived as it was incorporated into the demonstration in order to insure that the functionality of the distributed file handling services of the DMS were tested. In the real DMS the globally known names may well be stored at every node.

The second phase of the simulation will consist of a set of transportable Ada packages for the user's host VAX that will enable the user to call the DMS Service utilities from user written application programs. The user's requests for remote services will be transmitted to the Ada simulation running in the ASD VAX for servicing. In the second phase simulation, the user's host VAX will appear as a node on the DCN and requests for local services will be considered to be remote since the simulation will only reside in the ASD VAX and not a remote user VAX.

Datagram Service Simulation

The Datagram Service is simulated on the VAX using six Ada tasks as shown in Figure 4. Three of these tasks run in each of the two VAXs involved in the simulation. Task SEM is a semaphore that controls access to the underlying DECNET network "file" (DECNET looks like a file to an Ada subprogram or task). Tasks INQUEUE and OUTQUEUE continually pass a token back and forth across DECNET until one of the OUTQUEUE tasks has a datagram to transmit. When this task gains possession of the token it transmits its datagram(s) and then goes back to circulating the token.

The simulated Datagram Services such as SEND or RETRIEVE are procedures that either place a datagram in an outgoing queue or fetch one from an incoming queue. The datagram service supported by the simulated DMS services is a very Ada oriented service in that the datagrams themselves are Ada objects that are prefaced by a header that contains the transmission parameters.

Package DATAGRAM is a generic package that a user of the Datagram Services instantiates for each different Ada object to be transmitted. At the receiving end, the type of object being transmitted can be determined by first examining the header.

The format of the SEND command for datagrams is:

SEND(MESSAGE,ADDRESS); or SEND(MESSAGE,NAME);

where MESSAGE is the datagram to be transmitted and ADDRESS is the logical

network address of its destination. When the alternate form of the command is used, the parameter NAME is the name of a list of addresses to which the datagram is to be sent. The simulated Datagram Service supports a multicast capability for selectively sending datagrams to a list of predefined addresses. The command for assigning a logical name to a list of addresses is the MULTICAST command and its format is:

MULTICAST(NAME,AddressList);

where NAME is the name to assign to the list of addresses and AddressList is a linked list of addresses. A broadcast capability to all nodes on a given LAN or to all nodes on the network is provided by supplying a "broadcast address" in the first form of the SEND command shown above. The ability to scan the incoming datagram queue for messages with a specific combination of transmission parameters (e.g., priority, time-tag, etc.) is provided by the SCAN command which returns the message count (number of messages) with this combination of transmission parameters.

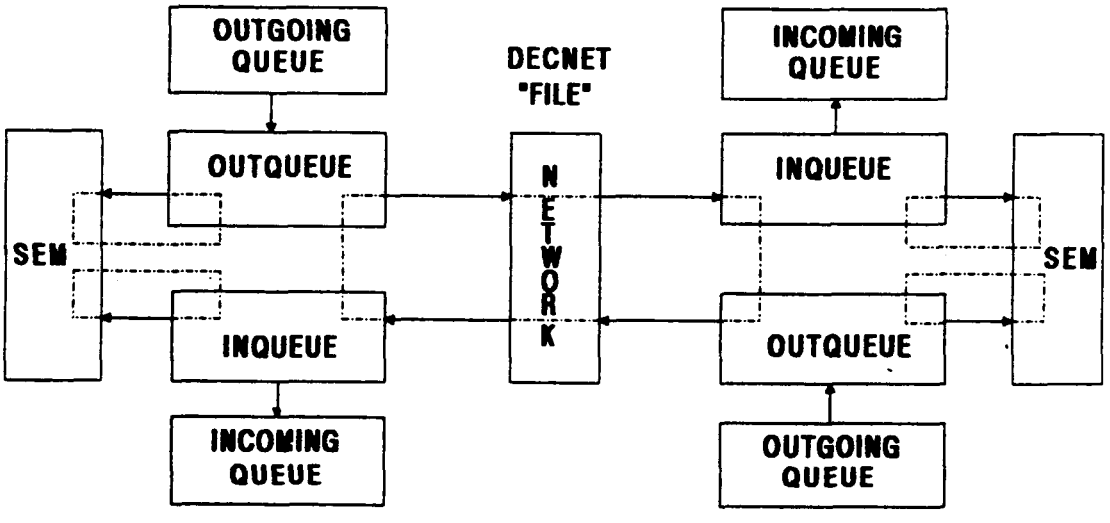

Figure 4

A message can be physically retrieved from the incoming queue using the RETRIEVE command. The RETRIEVE utility provides selective retrieval for a specific combination of transmission parameters or it can be used without parameters to retrieve the highest priority message in the incoming queue.

Virtual Circuit Service Simulation

When the information exchange between subsystems of the DMS must simulate a continuous dialogue over a physical circuit or when near real time transmissions are necessary it is usually more efficient to establish a "virtual circuit" between the subsystems. Some of the advantages of a DMS virtual circuit are:

(1) the routing information (source and destination address) does not have to be provided to the NOS for every transmission,
(2) network bandwidth is reserved for the dialogue, insuring a certain maximum transmission delay and
(3) messages are always received and handled in the same sequence they are transmitted.

The simulation of the virtual circuit capability on the TAVERNS Emulator is very similar to that of the datagram service. The main difference is that dedicated incoming and outgoing queues are established for each circuit in both the subsystem requesting the connection and the subsystem being connected. The other main difference is that virtual circuit traffic is "multiplexed" over the DECNET connection to simulate the reservation of bandwidth but the ability to assure a maximum transmission delay of the order of magnitude anticipated for near real time communications on the station is not possible using DECNET when other users are on the system.

To establish a connection (virtual circuit) an Ada subprogram or task calls the CONNECT procedure and to deallocate a circuit (and its associated queues) the DISCONNECT procedure is used. Once a connection is established the connected Ada subprograms can transmit and receive using the XMIT and RECV commands whose formats are:

XMIT(MESSAGE,CIRCUIT); and RECV(MESSAGE,CIRCUIT);

where MESSAGE is the Ada object to be transmitted and CIRCUIT is the circuit number assigned to the virtual circuit by the NOS at the time of connection. The Ada procedures that provide the virtual circuit capability are contained in the generic package VIRTUAL (this does not include the connection service). An Ada subprogram using the virtual circuit capability must instantiate a version of this package for each different Ada object that will be transmitted.

Data Acquisition and Distribution Services Simulation

The DMS Data Acquisition and Distribution (DAD) Service is layered over the datagram and virtual circuit services provided by the DMS. Which of these underlying services is used depends upon whether the usage of the DAD service is

periodic or not (as will be explained below). As previously mentioned, one of the key features of the DMS Services is the ability to request readings of sets of measurements. The simulation of this service in the TAVERNS Emulator is based on the following (simplified) Ada measurement object defined by:

```
type MEASUREMENT(REP: RepType; d1, d2, d3: positive) is
  record
    NAME: string(1 . . 15);
    case REP is
      when FLOATPNT      => FVAL : float;
      when FIXEDPNT      => IVAL : integer;
      when TEXT          => TXT  : string(1 . . d1);
      when FLOATARRAY1 => FVAL1: FLTARRAY1(1 . . d1);
      when FLOATARRAY2 => FVAL2: FLTARRAY2(1 . . d1, 1 . . d2);
        :
        :
    end case;
  end record;
```

where the discriminant REP is a variable of the enumeration type RepType that includes an entry for each data structure that the Data Acquisition and Distribution (DAD) Services supports and where d1, d2 and d3 are parameters that indicate the size of arrays, strings, etc. to be associated with a measurement as its "value".

Measurements are known globally by their NAME which is recorded in the NAME field of the MEASUREMENT variant record object. Each measurement is owned by some subsystem and this ownership is known to the DMS Services. The value of a measurement is stored in the variant part of the MEASUREMENT record object and can be of any type for which a corresponding entry in RepType exists. The enumeration list for RepType shown above is only representative as many of the types are yet to be determined.

The package DATAREQUEST contains the procedures for preparing a request message for transmission. They are REQUEST, MAKESET, GETSET and READNEXT. The MAKESET procedure associates a name with a list of measurements. The format of the MAKESET command is then defined by

MAKESET(SetName,MeasurementList)

where SetName is the name to be assigned to the measurement set and MeasurementList is a list of measurement names. The REQUEST command requests the reading of a set of measurements. The format of the REQUEST command is:

REQUEST(SetName) or REQUEST(SetName,PERIOD)

The second form of this command utilizes the PERIOD parameter which is a request for a periodic reading of the measurement set every PERIOD seconds where PERIOD is a non negative floating point number. The following steps outline the requesting procedure.

(1) call the MAKESET procedure to create the named measurement set,
(2) call the REQUEST procedure to request the reading of the set,
(3) call the GETSET procedure to obtain the set name of the next measurement set that has been successfully processed as a result of a previous REQUEST and
(4) use the STATUS parameter of the GETSET command to determine if the reading of a set has been completed.

The format of the GETSET command is:

GETSET(SetName,STATUS,TIME)

where SetName is the name of a previously requested measurement set whose processing has been completed at the time returned in the TIME parameter. Once a set name is obtained from the GETSET utility, the measurements in the set can be read using the READNEXT procedure whose command format is:

READNEXT(SetName,MEAS)

where MEAS is of type MEASUREMENT. To use the measurement's name as an operand is straightforward since its type is known to always be a 15 character string. To perform an operation on a measurement's value, however, involves examination of the discriminant of MEAS since MEAS is a variant record. The STOP command is used to stop an active periodic REQUEST. The format of this command is:

STOP(SetName).

The DATAREPLY package provides the necessary procedures to be used in responding to a REQUEST. These utilities are: REPLY, SETNAME, NEXTNAME, and WRITENEXT. They are similar in nature to the utilities provided in the DATAREQUEST package.