DEVELOPMENT OF AN OBJECT-ORIENTED SOFTWARE APPLICATION TO PROVIDE A TRANSPARENT INTERFACE BETWEEN SPACE NETWORK OBJECTS AND A TELEMETRY SYSTEM FOR TRAINING

Mitchell Kleen and Joey White and Joseph Policella
CAE-Link Corporation
2224 Bay Area Boulevard
Houston, Texas 77058-2099

ABSTRACT

The Space Station Verification and Training Facility is using an object-oriented design methodology for software design, a rate monotonic scheduling and message passing system to support the highly distributed environment, and the Ada language to implement most of the software. One of the subsystems within the Space Station and Training Facility is the Space Network Simulator. Space Network simulators are used to provide training of ground controllers and flight crews, providing a model of real-world formats and protocols. This gives the controller the appearance of a real-world network, providing valuable training. To develop a simulation of the space network within this distributed environment, software objects are under development to dynamically simulate the existence of the space vehicle(s) and their communication components. Communication components include the on-board antennas, transponders, communication systems, and corresponding communication ground control facilities. Telemetry systems are used in the simulation to provide the control of actual data manipulation, as a function of the state of the simulated Space Network. The telemetry system automatically formats appropriate telemetry characteristics through mode and control commands. A software model is under development to provide a transparent interface between the software objects and the telemetry system, allowing the objects to execute without knowledge of the particular telemetry system in use. A transparent interface between the software and hardware, within this object-oriented methodology, reduces the propagation of change to software models as the interface requirements change.

KEY WORDS

Object-oriented, Composition, Inheritance

INTRODUCTION

The purpose of this paper is to describe a sample of a current object-oriented development within the Space Network Simulator (SNS). The SNS is one of the subsystems within the Space Station Verification and Training Facility (SSVTF). The SNS requirements are to train Ground Support Personnel (GSP) in the Space Station Control Center (SSCC). In the SNS, the Instructors who operate the simulator replace the people in the real world who operate the Network Control Center (NCC) at the Goddard Space Flight Center (GSFC) and at the White Sands Complex (WSC). The SNS simulates the interface between the SSCC and the simulated Space Station, and it is the hardware used in the simulation that is subject to the most change. If the SNS software to hardware interface remains transparent to the SNS software, the SNS software models will not have to be modified to add additional capability when hardware requirements are modified. The SNS has experienced numerous hardware design changes throughout the life of the SSVTF, but regardless of the changed requirements, the software design of the simulated communication systems has remained. This paper describes this design methodology.

THE SSVTF ARCHITECTURE

The SSVTF Ada software architecture must support a general distributed hardware environment. Figure 1 shows the general SSVTF hardware architecture with two session computers and the various non-session assets connected via the Real-Time LAN (RT LAN) and the nodes on the General Purpose LAN (GP LAN). Also shown are the multiple cpus per node and multiple nodes per session computer system. Cpus communicate in local memory, nodes communicate via reflective memory, and other assets communicate on the RT LAN. The SNS is one of the asset computers that exists in this SSVTF Ada architecture.

The real-time system interfaces include a generic thread executive that provides a periodic Rate Monotonic Scheduling (RMS) task to cycle a model at a given rate, a messaging system that allows models to communicate in a distributed environment, and the Distributed Identifier Specification (DIS) which provides the association of logical names to physical data variables for instructor Station (IS) display and manipulation. The real-time system services provide a virtual machine on which the SSVTF models execute. These services support a distributed Ada environment. Each model exists in a self-contained structure denoted as the partition.

Each SSVTF software model within a partition is decomposed in an object-oriented fashion based on real-world structures and assemblies. Object-oriented means that data and the data associated operations are grouped into class structures. A class

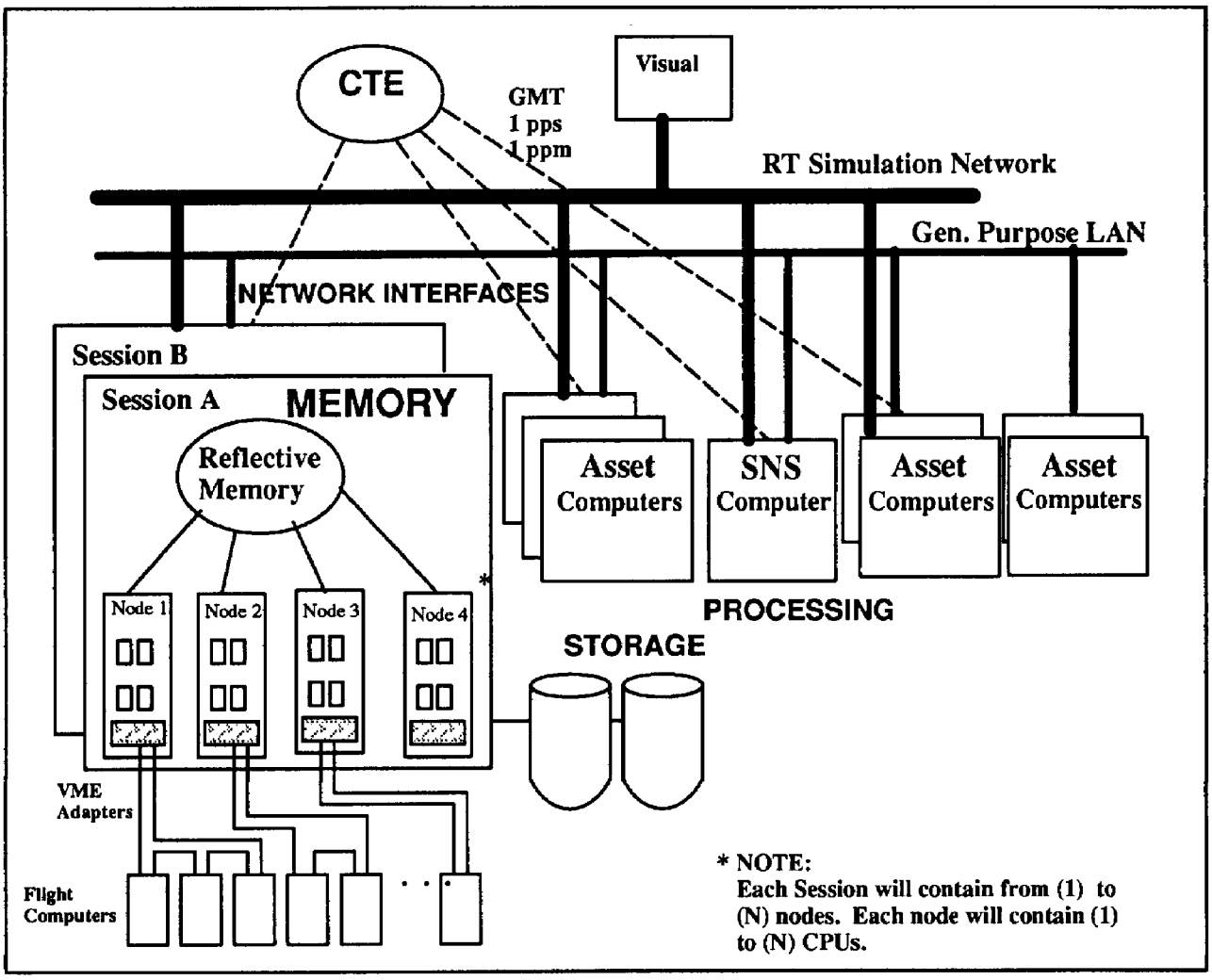

Figure 1

structure encapsulates the hidden portion of the object's attributes and operations and exports the data type that abstractly represents the object and the valid operations. Class structures may be made up of other classes by declaring instances of lower-level classes in the object-attribute record of the higher-level class. If the higher level class represents a less abstract form of the lower-level class, then this structure is defined as inheritance. If the higher-level class represents an assembly where the lower-level classes are sub-parts of the higher level class, then the structure is called a composition.

THE SNS ARCHITECTURE

The SNS is one of the subsystems within the SSVTF architecture. In the SNS, the instructors who operate the simulator replace the people in the real world who operate the NCC and the WSC. The SNS requirements are to train Ground Support Personnel (GSP) in the Space Station Control Center (SSCC) . The SSCC console positions

which receive the majority of the training provided by the SNS, are the Ground System Manager (GSM) and his support personnel. When running in integrated mode, the SNS is receiving and transmitting data at real world rates and in real world formats, to both the SSCC and the vehicle simulation. An interfacing telemetry system provides this interface (see Figure 2 below). The SSCC is not required to support a training configuration which differs from that needed to support the Space Station Facility (SSF). All the SSCC console positions receive the same data as they would during operational support. A primary training goal, of crew/GSP interaction is satisfied through sessions conducted in the integrated mode. The third type of interface supported is the network management data interface. The SNS Tracking Date Relay Satellite System (TDRSS) simulation generates the network management data, as well as the SSCC interface. This data consists of TDRSS Status and Ground Control, and Tracking. Since these data interfaces already exist, in support of the Shuttle Training System and other NASA spacecraft, the format is the 4800 bit NASCOM block protocol.

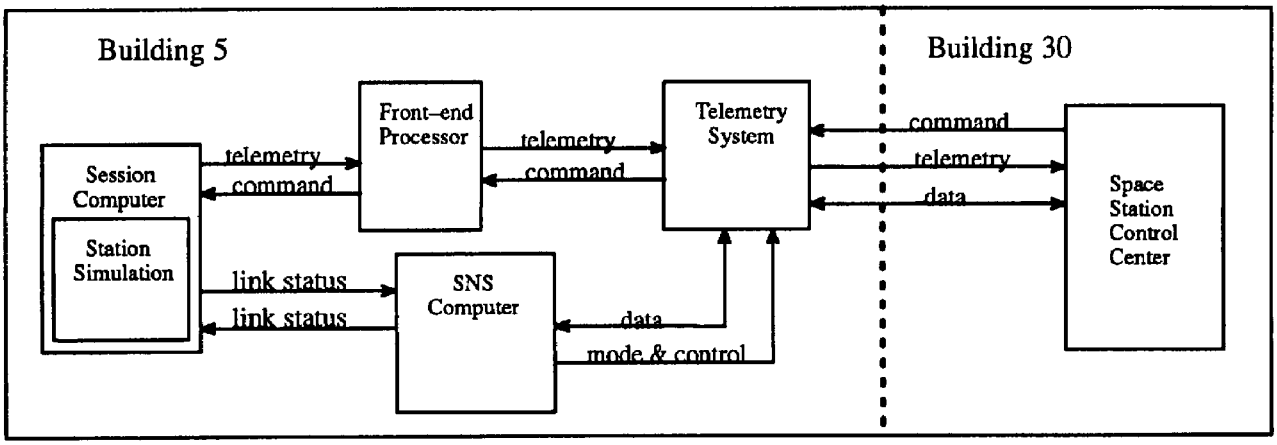

Figure 2

IMPLEMENTATION

To develop a simulation of the space network with the described requirements and within this distributed environment, software objects are under development to dynamically simulate the existence of the space vehicle(s) and their communication components. Communication components include the on-board antennas, transponders, communication systems, and corresponding communication ground control facilities. The telemetry system is used in the simulation to provide the control of actual data manipulation, as a function of the state of the simulated Space Network. The telemetry system automatically formats appropriate telemetry characteristics through mode and control commands, and provides an interface to receive SNS network management data.

The SNS software consists of 9 partitions. All of the partitions, with the exception of the Hardware Interface Partition, represent an SNS abstraction of the simulated real-world objects. The communication models within the White Sands partition, the Space Station Partition, and the TDRS partition communicate to each other through the RfLinks partition. This is essentially a real-time handshake of information, to determine whether a transmitting and receiving entity have the correct signal characteristics for an acquisition of signal. Sun and earth position is also included in this calculation. The RfLinks partition sends this data to the Hardware Interface partition. Based on this information, the Hardware Interface partition modes the telemetry system to manipulate the simulated data stream. At the same time, the White Sands partition also receives performance data from the Hardware Interface partition. The Hardware Interface partition receives this information through the telemetry system interface. White Sands uses this information to build performance data blocks, based on the network configuration commanded by the simulated GSFC. The simulated GSFC, like White Sands, also has an interface with the Hardware Interface partition. It receives Ground Control Message requests from the SSCC through this hardware interface. The output of the Goddard partition, after receiving the performance data, is to pass the information on to the Hardware Interface partition, who uses the telemetry system to pass the data forward to the SSCC. As noted, the Hardware Interface partition provides a transparent interface to the software models described above. This transparent interface for software development of the SNS allows models to continue executing without having knowledge of the hardware interface in place.

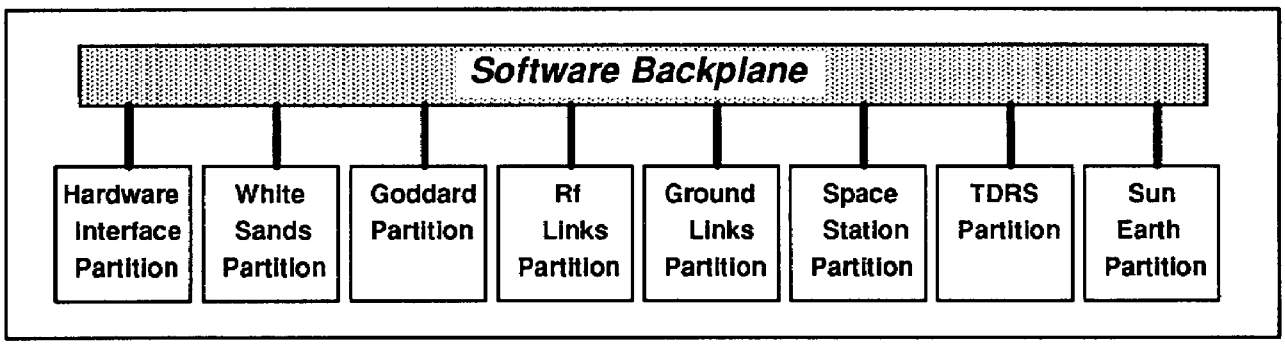

Figure 3

At the object level, the Hardware Interface Partition instantiates the Hardware Interface Control Class and the Hardware Interface Data Class creating a Hardware Interface object for controlling the telemetry system, and a Hardware Interface object for transmitting and receiving data, respectively. It instantiates The Thread Exec in the Simulated Virtual Machine distributed executive that is an instantiation of the periodic package, defined by the Generic_Model package. The partition supplies the mode routines during the instantiation. The thread exec executes the mode routines at the

appropriate times. Internal partition mode routines iterate the objects at a rate of 25 hertz. Messages to other partitions are sent to through the message system.

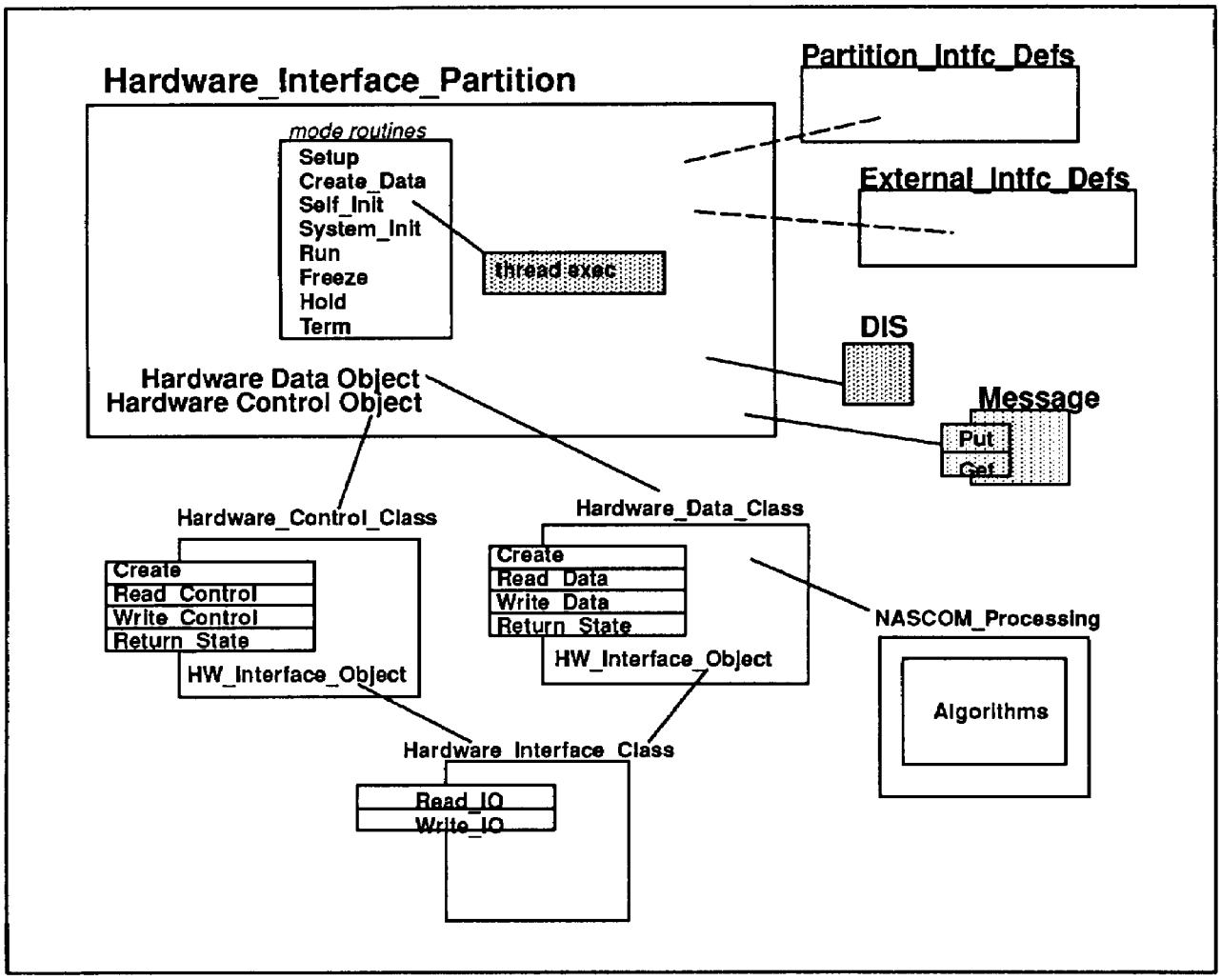

Figure 4

The Hardware Interface Class is the parent class that both the Telemetry Hardware Interface Control Class and the Telemetry System Hardware Interface Data Class inherit operations from for basic low-level read and write processing. Commercial-Off-The-Shelf Software (COTS) device drivers are provided with the procurement of the interfacing workstation. The procedures from the COTS are accessed by a child class through the generic instantiation shown below. A key data structure, the Data Block, is also a generic instantiation. The child classes provide the attributes that defined this basic structure for read and write operation. Note that all processing necessary to manipulate the object are performed in the package body of the Hardware Interface Class and are hidden from the child. Functions return the data block and state of the data block itself, providing an instance of the child class with all the necessary information to continue processing

```
with System; with Low_Level_Types;
generic
    type The_Data_Block is private;
    with procedure Read (X : The_Data_Block; Y : in System.Address);
    with procedure Write (A : The_Data_Block; B : out System.Address);
package Hardware_Interface_Class is
    type Object is private;
    procedure Io_Read (Instance : Object; Block_Address : in System.Address);
    procedure Io_Write (Instance : Object; Block_Address : out System.Address);
    function Get_Io_Status (Instance : Object) return Low_Level_Types.Io_Status_Kind;
    function Get_Data_Block (Instance : Object) return Low_Level_Types.A_1500k_Byte_Block;
private
    type Object is
        record
            The_Data_Block : Data_Block;
            The_Block_Address : System.Address;
            The_Status : Low_Level_Types.Io_Status_Kind;
        end record;
end Hardware_Interface_Class;
```

The Telemetry System Hardware interface Data Class class provides the transport interface between the telemetry system device driver and the SNS application models necessary to transmit and receive NASCOM blocks. A NASCOM block is a data structure the real-world space network uses to transmit and receive information. Once the data is read from the telemetry system, an instance of the Telemetry System Hardware Interface Data Class uses an instance of the Nascom Structures to format the block into a Nascom block, and extract display information (details of this display structure not shown) out of the block for the Instructor Station. Then, the formatted NASCOM block is propagated to the other SNS partitions through the Hardware Interface Partition (via the message system). To write to the telemetry system, the instance of the Telemetry System Hardware Interface Data Class receives a Nascom block (passed down at the partition level from the Ground Link partition) and is returned to the form necessary to pass through the device driver software.

```
with Hardware_Interface_Class; with Low_Level_Types; with Generic_Device_Driver; with Nascom_Structures;
package Telemetry_System_Hardware Interface_Data_Class is
    type Object is private;
    procedure Receive_Telemetry_System_Data (Instance : Object; Data_Block : in
        Low_Level_Types.A_1500k_Block;
    procedure Collect_Ground_Link_Data (Instance : Object; Nascom_Block : in
        Nascom_Structures.Nascom_Block);
    function Nascom_Block (Instance : Object) return Nascom_Structures.Nascom_Block;
private
    procedure Read_From_Telemetry_System is new Generic_Device_Driver.Read
        (Low_Level_Types.A_1500k_Block);
    procedure Write_To_Telemetry_System is new Generic_Device_Driver.Write
        (Low_Level_Types.A_1500k_Block);
    package Hardware_Interface is new Hardware_Interface_Class
        (Low_Level_Types.A_1500k_Block, Read_From_Telemetry_System, Write_To_Telemetry_System);
```

```
    type Object is
        record
            The_Nascom_Block : Nascom_Structures.Nascom_Block;
            The_Interface : Hardware_Interface.Object;
            The_Display_Parameters : Parameters_For_Display;
        end record;
end Telemetry_System_Hardware_Interface_Data_Class;
```

The Telemetry Hardware Interface Control Class provides the interface between the hardware device driver and the application software models necessary to mode and control the SNS telemetry system. Based on the state of the SNS RfLink models (via the partition to partition message system), the bit synchronizers within the telemetry system are moded to receive the simulated real-world data stream. The data stream can consist of command data from the SSCC or telemetry data from the simulated station (refer to figure 2). Additional mode and control occurs if the state of the SNS models require noise to be added to the serial data, or if the SNS has been commanded to a trainer specific standalone mode. In the standalone mode, the telemetry system is commanded to generate its own data streams internal to the system. This commanded mode is required for maintenance and telemetry system check-out. However, the SNS models continue to execute their operations without any knowledge of a new trainer mode.

```
with Hardware_Interface_Class; with Low_Level_Types; with Generic_Device_Driver;
package Telemetry_Hardware_ Interface_Control_Class is
    type Object is private;
    procedure Interrogate_Models
        (Instance : in out Object; Rf_State : in Rf_Link_State; Ground_State : in Ground_Link_State);
    procedure Mode_Bit_Synchronizer (Instance : in out Object; Link : in Link_Characteristics; Synchronizer : in
        Integer);
    procedure Mode_Standalone_Drivers (Instance : in out Object; Link : in Link_Characteristics);
    function Get_Bit_Sync_State (Instance : Object; Bit_Sync : in Integer) return Bit_Sync_State;
    function Get_Data_Block (Instance : Object) return Low_Level_Types.A_l500k_Block;
private
    procedure Read_From_Telemetry_System is new Generic_Device_Driver.Read
        (Low_Level_Types.A_l500k_Block);
    procedure Write_To_Telemetry_System is new Generic_Device_Driver.Write
        (Low_Level_Types.A_1500k_Byte_Block);
    package Hardware_Interface is new Hardware_Interface_Class
        (Low_Level_Types.A_1500k_Block, Read_From_Telemetry_System, Write_To_Telemetry_System);
    type Object is
        record
            Bit_Syncs_State : Bit_Syncs_State;
            The_Status_Parameters : Parameters_For_Status;
            The_Interface : Hardware_Interface.Object;
        end record;
end Telemetry_Hardware_Interface_Control_Class;
```

CONCLUSIONS

High fidelity simulations of telemetry and data communications networks have been utilized by the US space program for training of flight crews and mission control center personnel for many years. For the space program, changing requirements are common, and it is imperative in this budget constrained environment that simulation systems be designed in a way that is resilient to change. This is a fundamental design concept for the SNS. The SNS simulates the interface between the Space Station Control Center and the Space Station, and it is the hardware used in the simulation that is subject to the most change. For the SNS, software models maintain the simulated state of the real-world communication models, and based on their state a hardware interface agent modes a telemetry system. At the same time, this hardware interface agent must also receive data from the telemetry system, and populate the software models with simulated performance data. If this software to hardware interface remains transparent to the software, the software models will not have to be modified to add additional capability when hardware requirements are modified. A transparent interface between the this software and hardware, within this object-oriented methodology, reduces the propagation of change to software models as the interface requirements change. For SSVTF, this approach has withstood several design changes. The hardware requirements have been modified or considered for modification, and the analysis has always concluded that there was minimal impact to the simulated communications systems.

ACKNOWLEDGEMENTS

The authors wish to thank Robert H. Sturtevant and members of his Real-Time Sessions group who provided much of the basis for this architectural platform. Significant contributions for object-oriented techniques in Ada was provided by Stanley R. Allen and David G. Weller.

REFERENCES

Policella, Joseph, "A GENERIC OBJECT ORIENTED DESIGN FOR A RADIO FREQUENCY SIMULATION IN A SPACE TELEMETRY AND COMMAND ENVIRONMENT", Proceedings of the 1991 International Telemetry Conference.

White, Joey, "A TELEMETRY AND SPACE COMMUNICATION NETWORK SIMULATION FOR TRAINING", Proceedings of the 1991 International Telemetry Conference.

Policella, Joseph and White, Joey "AN OBJECT ORIENTED COMMUNICATIONS NETWORK SIMULATION FOR SPACE STATION FREEDOM CONTROL CENTER TRAINING", Proceedings of the 1992 Intra Service Industry Training and Simulation Conference.

Policella, Joseph and White, Joey and Shillington, Keith "AN OBJECT ORIENTED COMMAND AND TELEMETRY "BLACK BOX" SIMULATION USING ADA", Proceedings of the 1993 International Telemetry Conference.

Policella, Joseph and White, Joey "A SYSTEMATIC METHOD FOR SYNTHESIS OF OBJECT ORIENTED SOFTWARE DESIGNS FOR TELEMETRY SIMULATION", Proceedings of the 1993 International Telemetry Conference.