USING DATAFLOW ARCHITECTURE TO SOLVE THE TRANSPORT LAG PROBLEM WHEN INTERFACING WITH AN ENGINEERING MODEL FLIGHT COMPUTER IN A TELEMETRY SIMULATION

Joey White
CAE-Link Corporation
2224 Bay Area Boulevard
Houston, Texas 77058-2099

ABSTRACT

One of the most challenging technical problems in the development of a spacecraft telemetry simulation is the interface with a flight computer running real-world flight software. The ability of the simulation to satisfy flight software requests for telemetry data, and to load, mode, and control the flight software along with the simulation, can be constrained or degraded using conventional interface solutions. Telemetry dataflow architecture systems can be utilized to solve the interface problems with less constraints. This is an especially attractive solution in a telemetry simulation where the telemetry system can also be used to format and serialize spacecraft telemetry, and receive and preprocess commands. This paper discusses the concepts developed for such a system for a training simulation of the Orbital Maneuvering Vehicle for NASA at Johnson Space Center.

INTRODUCTION

The purpose of this paper is to illustrate the applicability of the dataflow architecture to a telemetry simulation, and the problem of interfacing the simulation with an engineering model flight computer. The concepts discussed in this paper were developed for application on a telemetry simulation for real-time man in the loop training of control center personnel for the NASA Orbital Maneuvering Vehicle (OMV).

The OMV was designed to be a telerobotic space vehicle which was to be piloted from the ground at Johnson Space Center (JSC). The vehicle was designed to downlink real time status telemetry, and digital video from on-board cameras. The vehicle was to be piloted during critical docking maneuvers by a pilot manning a control console in the control center. The pilot viewed real time video on a graphics

terminal which also displayed numerical and graphical depictions of status telemetry. The pilot maneuvered the vehicle in space using hand controllers. Movement of the controllers resulted in formatting and uplink of appropriate thruster firing commands from the ground. The pilot, and supporting control center personnel had an ability to send commands to commandable spacecraft systems. Communications with the vehicle for commands and telemetry during nominal mission operations was via the NASA Tracking Data Relay Satellite System (TDRSS). Figure 1 illustrates a simplified OMV data flow.

The vehicle design incorporated dual redundant on board computers for sequenced mission control and on board systems management. The computers had a local ability to command the same on-board systems as the ground via uplink. The computer also had an ability to request telemetry parameters from instrumented on-board systems. In addition, the ground had an ability to uplink commands to the computers, as well as software and mission sequence data table loads.

The OMV simulation requirement was to simulate the spacecraft in such a manner that there was no discernable difference to the pilot and flight controllers between the simulation, and the spacecraft. This included production and digitization of video scenes as viewed from the spacecraft, production of all telemetry parameters visible to the control center, and an ability to receive and respond in a realistic manner to all possible uplink commands. The simulations physical interfaces to the control center were required to be in real world telemetry and video formats, so that the actual systems to be used during an OMV mission could be exercised during training scenarios. Likewise, the simulator was required to be able to receive uplink commands in the real world command format. One of the more interesting requirements was a simulation of the real world delay introduced by the TDRSS network, and the OMV on-board systems. Since the spacecraft was flown from the ground during critical docking maneuvers, it was deemed critical that the pilots have an ability to practice docking with a delay of several seconds between the movement of hand controllers, and visual feedback on the video screen showing results of hand controller manipulation.

While interesting in many respects, the area of interest in this paper is the interface with the on-board computers, and how the dataflow architecture lent itself to solving many of the most challenging problems of the simulation. Studies done at JSC indicated that the least risk design for simulation of the on-board computers was to use real world equivalent engineering models of these machines, running actual flight software. The spacecraft had multiple design reference missions. A functional flight software simulation (coded in software and resident on the simulation host computer) would require software rewrites for each different design reference mission.

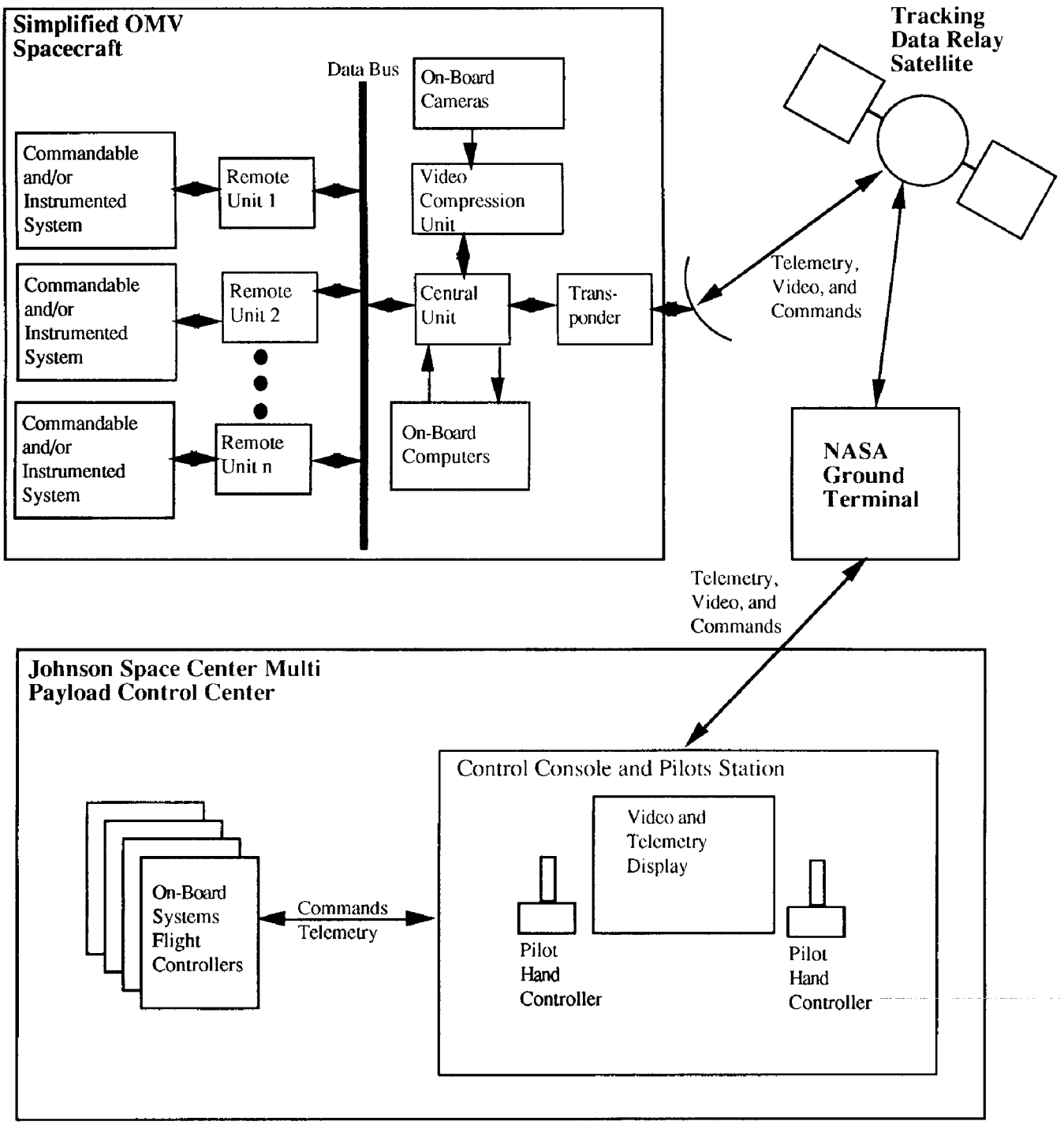

Figure 1 - Simplified OMV Data Flow and Interfaces

Engineering model hardware with real-world flight software avoided this problem by using the real world software.

The most challenging aspect of the simulation design was to determine the most effective means of interfacing the simulation with the on-board computers in such a manner that the flight software running in them did not require modification to run in the simulation. It was important that the interface to the on-board computers and their flight software be satisfied to the extent that there was no difference in the execution profile of the flight software in the simulation from the execution profile aboard the actual spacecraft.

THE TRANSPORT LOOP LAG PROBLEM

Aboard the vehicle, commands and requests for telemetry data from the on-board computer to on-board systems were output via a high speed serial interface (512 Kbps) to a central unit of the on board Command and Data Handling system (C&DH). From the central unit, the commands and requests for telemetry data were relayed via a data bus to a remote unit which acted both as a multiplexor and demultiplexor for telemetry and commands. The remote unit provided interface signal conditioning. This design allowed transport of commands from the on-board computer to on-board systems in a sub-millisecond time frame. This design allowed requests for telemetry data by the on-board computer from on-board systems to be satisfied in approximately . 1 milliseconds.

The transport loop lag problem is brought about because the software models of the various on-board systems in the simulation were hosted on a host computer separate from the on-board computers. These models substitute for the on-board systems with which the on-board computer communicated via the C&DH aboard the actual spacecraft. An example is a software model of the propulsion system of the spacecraft.

Since the software model host computer and the engineering model flight computer were separate physical devices, it was necessary to interface them using an intermediate hardware device. There were no commercially available interface devices which could transport the on-board computer requests for telemetry data to the simulation host computer, and output a response containing a telemetry value within a sub-millisecond time frame. Most available commercial devices exceeded this turn around time requirement by an order of magnitude.

Budgetary constraints on the program dictated use of an existing system design for on-board system model execution. This design used the classical cyclic executive approach, which is prevalent in real-time training simulation. On-board system

software models cycle at pre-determined fixed iteration rates. In this case the model iteration rate was a maximum of 31.25 Hz, meaning each model would get a chance to execute once per 32 milliseconds.

Additional constraints were that the on-board computer's Input/Output (I/O) profile was variable based on loading. The number of commands and requests for telemetry data depended on current mission circumstances, and the time at which any particular command or request needed to be responded to was dependent on the flight software. This meant that responses to requests for telemetry data and response to commands could not be immediate from the on-board systems models, and that the start of execution times for on-board systems models could not be "tuned" in any methodical manner so that the transport loop lag problem was avoided.

SIMULATION MODING AND CONTROL

Several additional problems unique to the simulation environment were caused by the need to interface with engineering model flight computers. The most significant was the requirement for the simulation to be moded from an Instructor Operator Station (IOS). The major modes required for the OMV simulation were:

1. Run - an ability to start execution of the flight software running in the engineering model computers at a point in time synchronized with the start of execution of the on-board systems models on the host computer

2. Freeze - an ability to suspend execution of the flight software at a point in time co-incident with the end of a major on-board systems model iteration, in such a manner that the simulation can return to "Run" without loss of data

3. Datastore - an ability to store the current contents of the engineering model flight computer (software, data, internal registers, program execution counter) in such a manner that the stored contents can be reloaded in the computer and execution can resume at exactly the point execution was stopped

4. Return to Datastore - an ability to reload a "Datastore" into the flight computer, and resume execution at the point in time the Datastore captures, by moding to "Run".

A PAST APPROACH TO SOLVING THE PROBLEM

The Space Shuttle Mission Simulator (SMS) at Johnson Space Center has similar requirements for interface to on-board computers. This simulation incorporates

hardware engineering models of all 5 Space Shuttle General Purpose Computers. The solution deemed most effective at the time of the SMS implementation (mid 1970's) was to dedicate a large mini computer to servicing the simulation interface. In addition a complex custom hardware simulator interface device was built to physically interface the minicomputer to the on-board computers.

This design has proven effective, however it is still necessary to patch the Shuttle flight software and tune the interface to the simulation with every new on-board computer software load. This effort requires dedicated and specialized sustaining engineering resources. Since the interface equipment is custom, it is also relatively expensive to maintain. As with most custom solutions, this type of interface would also be prone to early obsolescence because no similar systems with later generation technology would be in development.

THE DATAFLOW ARCHITECTURE SOLUTION

A dataflow system of the type build by most of the telemetry equipment manufacturers is an effective solution for a telemetry simulation with these requirements. A system diagram is shown in Figure 2. The reasons this architecture is effective are described below in the order of most obvious to least obvious.

INTERFACE WITH THE SIMULATION HOST COMPUTER

Most telemetry vendors offer a two way interface from host computers to their dataflow architecture systems. This interface provides direct access to the system data bus both to and from the host computer. This ability was used in the design to pass model telemetry data to the telemetry data bus, and to pass processed commands from uplink and from the on-board computer, to simulation models resident on the host computer.

INVERSE ENGINEERING UNIT CONVERSION

General purpose programmable processor cards are available for most dataflow architecture systems. These cards are able to accept data from the telemetry data bus, process it, and return it to the bus with a new tag identifying it as processed data. One of the common uses of these devices is to preprocess downlinked PCM values into engineering units for a host computer. An application of this device in the design was to accept engineering units from the host computer and perform the opposite function, that is, convert them into PCM. PCM was the format for downlink and the format expected in reply to telemetry data requests from the on-board computer. It is generally most convient in a software simulation to keep all parameters in Engineering

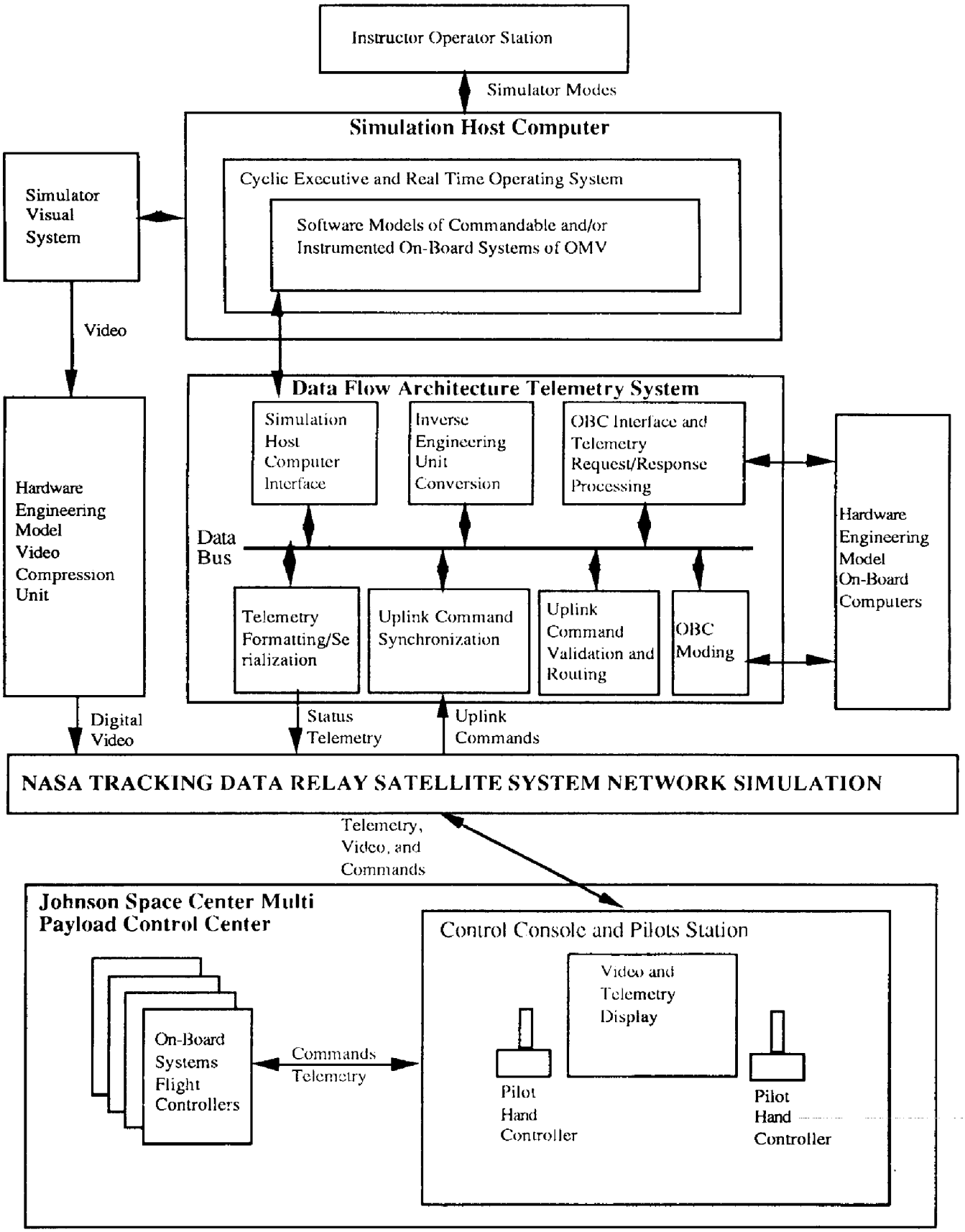

Figure 2 - Telemetry Simulation System Diagram

Units. In the design this capability off loaded conversion of in excess of 2000 telemetry parameters per second from the simulation host computer. The output of this function was processed telemetry, which was to be input from the bus by both a telemetry formatting and serialization function, and an on-board computer interface function.

TELEMETRY FORMATTING AND SERIALIZATION

Most of the telemetry vendors offer a telemetry simulator card or serialization card with an on-board programmable ability to input data from the system data bus, automatically format appropriate "telemetry" characteristics such as synchronization codes, frame and sub-frame counter, and even commutate the telemetry data using real-time data from a computer interfaced to the data bus. This was the means to meet the simulation requirement to interface with the control center using the real-world telemetry formats.

UPLINK COMMAND SYNCHRONIZATION AND RECEPTION

Decommutators, which generally have programmable characteristics similar to those of spacecraft command receivers, are naturally able to lock up on uplink commands which have embedded synchronization codes. They are also generally able to discriminate between different "classes" of uplink commands and tag them for different destinations on the telemetry data bus. A DECOM was able to act as the simulation command receiver in the design.

COMMAND VALIDATION AND ROUTING

Another application of a general purpose processor card in the design was to validate the uplink commands once received by the DECOM, strip off artifacts of uplink transmission, and route the commands to the appropriate destination, which could be on-board systems models on the host computer via the bus resident computer interface card, or the on-board computer interface function.

ON-BOARD COMPUTER INTERFACE AND TELEMETRY DATA REQUEST/RESPONSE PROCESSING

Most important in the design was the ability to dedicate a programmable processor card to service of the engineering model on-board computer interface. This card interfaced to the computer using serial I/O ports, and to the telemetry system data bus using the standard data bus interface. From the data bus this card accepted fresh data from the host resident on-board systems models each simulation iteration, which had

been converted to PCM by the “inverse engineering unit conversion” function. In this manner, telemetry data replies were pre-positioned “near” the on-board computers and flight software. It became possible to turn around a request for telemetry from the flight software just as quickly as the real world spacecraft system. In addition, this card could accept on-board systems commands from the flight software and pass them to the command routing function, where they would be processed just like uplink commands destined for on-board systems models.

ON-BOARD COMPUTER MODING

The on-board computer was designed to be moded via a number of discrete I/O lines mapped directly into computer control registers. The simulation design incorporated discrete I/O cards interfaced to the telemetry system data bus which could affect these control lines, and thereby cause the computer to halt, resume execution, read out the contents of the computers memory and internal registers, and load the computer with data via the serial I/O interfaces nominally used to transfer commands and telemetry to and from the computer. This capability allowed satisfaction of the simulation requirements for “Run”, “Freeze”, “Datastore”, “Return to Datastore”, and other minor simulation modes.

CONCLUSIONS

It is difficult to justify the cost effectiveness of use of a dataflow architecture system in a simulation based upon it’s ability to perform any one of the individual functions in the simulation design, however the flexibility of the architecture allows it perform many of the major functions of a telemetry simulation using one system. For this reason, use of such a system is very cost effective. The cost to engineer individual solutions to each of the myriad of problems solved by this architecture would have easily exceed the cost of the designed system, and the combined complexity of individual solutions would have been extremely high.

The ability of a dataflow architecture system to dedicate processor resources to tasks like the engineering model computer interface adds a new dimension to simulation capabilities. It is now possible to do limited but time critical on-board systems models on a general purpose processor card resident on the system bus, making the implementation of certain classes of extremely time critical models possible. In the past impmentation of these types of models were either not technically feasable, or were constrained compromise solutions.

The applicability of a data flow telemetry system to this simulation problem may be surprising because this is not what these systems are generally used for. Since a

telemetry simulation with these stated requirements contains a simulation of a space based data handling system, and dataflow telemetry systems are ground based data handling systems, the parallels in functionality should not be difficult to comprehend.

REFERENCES

NASA, Orbital Maneuvering Vehicle (OMV) Level A Training Requirements, 04 April 1988

NASA, JSC-34240, OMV Training Facility Level B Requirements, 01 May 1989

NASA, JSC-22976, System Functional Requirements for the Orbital Maneuvering Vehicle (OMV) Training Facility (OTF), 19 December 1988

NASA, JSC-22909, OMV Baseline Operations Plan, 13 June 1988

NASA, JSC-23431, Study Analysis Report for the Orbital Maneuvering Vehicle (OMV) Training Facility (OTF) Onboard Computer Implementation, 17 April 1989