RAPID PROTOTYPING AND AI PROGRAMMING ENVIRONMENTS APPLIED TO PAYLOAD MODELING

Richard S. Carnahan, Jr.
Andrew P. Mendler

Martin Marietta Astronautics Group
Box 179
Denver, CO 80201

ABSTRACT

This effort has focused on using Artificial Intelligence (AI) programming environments and rapid prototyping to aid in both space flight manned and unmanned payload simulation and training. Significant problems which have been addressed by this work are (1) the large amount of development time required to design and implement just one of these payload simulations and (2) the relative inflexibility of the resulting model to accept future modification. To date, results of this effort have suggested that both rapid prototyping and AI programming environments can significantly reduce development time and cost when applied to the domain of payload modeling for crew training and, while the focus of this work has been modeling/simulation development for training, the techniques employed are applicable to a variety of domains where models or simulations are required.

INTRODUCTION

This work was initiated as a response NASA Marshall Space Flight Center (MSFC) concerns regarding current modeling techniques applied to model/simulation development for Spacelab payload crew training and operations. Early discussions suggested that problems identified with current modeling processes would provide ideal test cases for the application of advanced software development techniques, focusing on the use of AI programming environments and rapid prototyping as capable of reducing the amount of time and cost required for model/simulation development, resulting in software systems which are easier to modify and extend. To date, results have suggested that rapid prototyping and AI programming environments show great potential to reduce development time and cost when applied to the domain of payload modeling for crew training. In addition, movement toward a greater level of system autonomy for both payload crew training and on-board payload operations will likely result in more efficient and cost effective Space Station payload operations.

PROBLEM AND OBJECTIVES

The current function of the Payload Crew Training Complex (PCTC) is to provide an integrated simulation of on-board Spacelab experiment operations for purposes of instruction in (1) procedures designed to bring the system to an operational or shut-down state as well as monitor its health and status while running, and (2) predefined fault handling command sequences. During an integrated simulation, the crew is required to respond to various scenarios for each of the experiments included in a particular mission; scenarios may include normal as well as faulted conditions. As the simulation proceeds, the simulation director, whose job it is to continually monitor the simulation while running, may dynamically modify a limited number of experiment parameters. It is important to realize that all crew procedures are rigidly defined and few, if any, deviations from those procedures are allowed. When following the defined command sequence does not bring the system to a nominal configuration, the crew requests assistance from ground support. Presently, there is little facility for allowing the crew to perform fault detection/isolation activities or reconfigure the system in the case of an anomoly; all these functions are the responsibility of the PI and other ground support personnel.

There are several problems with techniques traditionally used to develop models/simulations required for payload crew training: first, the process of modeling payload experiments and simulating associated procedures is slow and cumbersome. Second, due to several interacting factors, current models are extremely difficult to either modify or extend, and third, several different (and often incompatible) computers may be used to develop simulation/training models for different phases (e.g., design, flight operations), resulting in inconsistant software systems across the development life-cycle. In summary, the current simulation/model development process is inflexible and resistant to change. The notion of a simulation building/modeling environment which should be used across all life-cycle phases

is appealing, and presents an opportunity for examining the effectiveness of alternative/advanced software development techniques.

This effort examined the feasibility of applying rapid prototyping and AI programming environments to some of the problems outlined above. Particularly, it was important to assess how effective, in terms of reducing both cost and time incurred for model development, such techniques would be when used to model Spacelab payload experiments.

The objective of this activity has been to develop and evaluate a software/hardware environment for rapidly prototyping payload experiment simulations for training and operations. More specific technical objectives included using rapid prototyping techniques and AI programming environments to (1) model three Spacelab experiments (Wide Field Camera [WFC], Geophysical Fluid Flow Cell [GFFC] and Biorack) on the basis of documented experiment simulation modeling requirements, (2) model the functionality of a Dedicated Experiment Processor (DEP - a microprocessor provided by the experiment PI to increase the capability of the system), (3) simulate control/display panels for the GFFC and Biorack experiments, (4) simulate current payload crew-machine interfaces (both display and keyboard input) for the WFC and Biorack experiments, (5) design and implement prototype interactive crew-machine interfaces, (6) design and implement a prototype simulation director console for use in future payload simulation design and autonomous payload crew training and (7) examine the potential for integrating advanced software/hardware with existing PCTC software and hardware. It is important to understand that the techniques employed are applicable to a variety of domains where models or simulations are required. The remainder of this paper describes in more detail the approach taken and results obtained.

APPROACH

Hierarchical, object-oriented modeling/simulation is seen as an effective means of reducing inadequacies associated with current simulation techniques. Relationships between and among dynamic system components are often difficult to understand when traditional simulation methodology is used. Component behavior is difficult to trace since code which directly impacts behavior is spread throughout the simulation model and is not necessarily referenced to specific system components. For the same reasons, component interactions are even more difficult to comprehend. Assumptions about system performance are not made explicit but are masked by low-level mathematical/engineering information which can, again, be found throughout the model. In such cases, modifications to the model are difficult to make and resulting interactions are often untraceable [1].

The approach we have taken to model/simulation building is hierarchical and top-down; modeling is viewed as an iterative process in which the first model version represents a high level description of system functionality in terms of qualitative relationships between and among system components. The idea here is that the system may be abstractly described, and yet a 'working' model results. Indeed, general domain information related to the system under consideration may be used to set the framework for dynamic system component interactions and may also make reasoning about system performance more efficient (e.g., causal modeling, model-based reasoning). When tied to a graphic representation of the system, the model, at any level, can be tested to examine the accuracy of current system information. As knowledge about the system becomes more refined, the model can be easily modified or extended, at the appropriate level, to incorporate the new information. The complex and frequently untraceable interactions which often develop when modifications are made to traditionally developed models are minimized.

The combination of rapid prototyping and model hierarchies provides a more efficient process for model development, particularly in relation to training simulations. The design and implementation of models/simulations for training purposes need not, and indeed probably should not, be a separate activity from the design and implementation of models/simulations for operations or verification and validation. The approach employed here for model/simulation development uses rapid prototyping to facilitate both the requirements definition and implementation phases of hardware/software development as well as speed the process of code generation. Hierarchical model development results in using only those levels of model/simulation fidelity required by the training process. Higher fidelity versions of models/simulations can then be used for verification and validation or operations. In summary, only one software development effort occurs (two separate pieces of software are not required) resulting in one, common piece software system with different levels of abstraction. Hardware modifications which may impact simulations required for verification and validation or operations would only be incorporated in the training software if those modifications would affect the training process.

LISP AS A MODELING LANGUAGE - LISP appears to be an ideal choice for a rapid prototyping programming language and was the language chosen for this work. LISP's primary advantage lies in its ability to handle symbolic manipulation as well as numerical computation; an appropriate capability for the hierarchical modeling/rapid prototyping approach outlined above. Abstract system component functional relationships can be easily represented and the inherent flexibility of the language provides for rapid code extensions or modifications when more detailed system information becomes available. Another basic feature of LISP which supports the modeling approach we have taken is its ability to be run as either interpreted or compiled code. System modifications are rapidly accomplished since only the modified piece of code need be reevaluated. In addition, this facility allows for an incremental simulation capability in which simulations may be interrupted while running, parameters or other features modified/examined and the simulation restarted from that point; such a capability is highly useful both for system design as well as operations. Finally, in support of hybrid models, LISP allows function calls which access other programming languages; readily available low-level FORTRAN component models may be incorporated as part of the total system software.

PRODUCTION SYSTEMS AND MODELING - Production systems are useful for representing domain knowledge which can be stated as condition-action pairs and have been traditionally employed to encode an expert's knowledge about a particular domain for use in an expert system. However, their representation structure can also be applied to any modeling application where logic tables are used as part of the modeling scheme. Note that for these simulations, expert systems were not being created using production rules; the production system provided an efficient bookeeping technique, allowing for ease in system modifiability; potentially dangerous system interactions could be handled without great difficulty. Another benefit can be seen if one proposes the eventual incorporation of an expert system for purposes of experiment fault detection/isolation or intelligent training. Much of the system knowledge has already been encoded and extensions to the knowledge base are easily made.

Experiment modeling was completed using a three-level model hierarchy. Control passes from Level 1 (written in LISP), which has responsibility for overall program flow, to Level 2, the experiment logic (written in the production system language HAPS - Hierarchical Augmentable Production System) to Level 3, the math model (written in LISP). If a rule fires which requires the calculation of a parameter, the appropriate equation is called and the resulting value is passed to Level 1 or 2 depending upon the event. Modifications to the system can be made at any level without adversely affecting any other. It should be clear that neither production rules (representing logic tables) nor equations (representing low-level engineering information) were initially required to design and implement an abstract, high-level model of the experiment. All that was needed to begin was a general narrative statement outlining the overall system operational sequence; as necessary, logic tables and equations could be added later.

OBJECT-ORIENTED MODELING - The decision to use object-oriented programming as a primary technique for experiment modeling was grounded in (1) the type of application and (2) that part of the approach which required the design and implementation of dynamic, interactive graphics interfaces. Experiment system monitoring which requires procedural input from the crew appears to be ideally suited to the message-passing capabilities which object-oriented programming provides (flavors were used for this work). In general, the notion of representing a dynamic system as a group of objects (components) and their associated behaviors is intuitive and provides a structural framework from which inferences concerning system behavior can be made. Message-passing among objects emulates potential system interactions since a single message can initiate any number of complex behaviors or other messages. It is important to understand that objects need not be limited to hardware component representation, but can be used to mirror other features of total simulation/model implementation such as process control functions and their interactions. In addition, objects may be combined to form other objects, a facility which is particularly useful when representing complex, dynamic systems [3].

The crew interface for experiment system monitoring was designed using the Phoenix Graphics Editor (also flavors-based) which supports rapid prototyping of color, interactive graphics interfaces for a variety of applications. Using Phoenix, objects can be graphically displayed and their behaviors either externally or internally defined. Graphic objects can be made interactive so that the user can directly manipulate the color screen; when the user interacts with an object on the screen, interactive behaviors associated with that object are triggered. Objects provide a flexible and efficient means for prototyping user interfaces in dynamic systems. Interfaces for both the prototype simulation director console and experiments were designed and implemented in significantly less time than it would have taken without the benefit of object-oriented programming.

One other significant benefit of object-oriented programming, crucial to the ease and rapidity with which systems are developed and modified, needs to be mentioned: the notion of inheritance. Object hierarchies provide a natural analogy to the modeling approach used for the present work. Higher-level objects correspond to higher-level qualitative system information, whereas lower-level objects are more closely tied to lower-level quantitative information. Object hierarchies can be defined such that modifications made to behaviors or attributes of an object will not affect any superordinate object. On the other hand, object modifications will be inherited by any subordinate object (specific exceptions to this rule are possible). In addition, multiple inheritance (inherited information from multiple objects) is useful when it is important to view an object from different perspectives [2].

INTELLIGENT MAN-MACHINE INTERFACES FOR TRAINING - Although not the primary focus of the initial effort, part of the work accomplished to date has been the design and implementation of a prototype intelligent simulation director console. As outlined above, the present situation in the PCTC requires that the simulation director console be continually manned while a simulation is progressing. Such a mode of operation is becoming prohibitively expensive. In addition, Space Station requirements that training software be transportable will compel the development of software which is significantly more robust and capable of running more autonomously. The prototype Space Station simulation director console was implemented to allow some idea of the potential intelligent support might provide. The focus of the present prototype was in the area of simulation design, and rudimentary interface functionality includes the capability to load various experiment software systems, choose simulation scenarios based on the experiments loaded, change parameters for those experiments loaded, create experiment timelines for the simulation and dynamically add events to appropriate timelines by directly interacting with the screen.

The Phoenix Graphics Editor (developed by MMAG) was used to prototype initial versions of advanced crew interfaces for purposes of training and operations. Initial interface designs may be rapidly implemented (1-2 days) and are easily associated with an underlying system or procedural model. System operations or perturbations are reflected by visual changes in graphic component representations. The Phoenix system allows windows created using the editor to be combined with any kind of window available on the Symbolics machine, providing flexibility in overall interface design and functionality.

It would appear that the potential for applying intelligent crew interface support features is significant. In an operational state advice concerning procedures to be taken in a given system state might be made available to the crew. In a training scenario, advice would be more tutorial in nature. Other intelligent support features (e.g., automated procedural sequences, automatic report generation) could also be added. Future work will examine these important issues.

RESULTS AND CONCLUSIONS

The two major problems, the large amount of time required for model development and the resistance of finished models to modification, have both been successfully addressed. To date, comparisons of advanced and traditional modeling approaches suggest software development time and resulting cost are significantly less when the advanced approach is used. The WFC experiment model/simulation was completed in 3.0 man-months (NASA's estimate was 28.0 man-months), while undergoing several major modifications. Modeling the GFFC experiment, including the DEP and simulating the control/display panel, was completed in 1.5 man-months; it is estimated that the GFFC experiment represented a more difficult case than the WFC due to the inclusion of the DEP and the control/display panel. DEP modeling was not a significant issue since it could be subsumed in the overall control structure. Indeed, GFFC DEP functionality was limited by NASA to handling inputs from the control/display panel and processing a master fault condition. Modeling the Biorack experiment, including the simulation of all current DDUs (Data Display Unit) and control/display panels, was accomplished in 2.0 man-months. In terms of overall system functionality, models developed using the advanced approach have been externally validated and appear to evidence the fidelity expected from experiment models developed using present techniques.

Using Phoenix as a rapid prototyping tool for the design and implementation of interactive crew interfaces proved to be extremely beneficial. Current payload crew interfaces for the WFC and Biorack experiments, as well as a prototype interactive interface for the WFC experiment, were rapidly implemented (2 man-weeks). In addition, control/display panels for the GFFC and Biorack experiments were easily simulated (3 man-days). It is important to realize that rapid implementation of interfaces facilitates immediate testing of the experiment model under

development. Additions/modifications to the WFC, GFFC and Biorack models could be dynamically tested and feedback provided concerning model accuracy by visually inspecting system performance. The usefulness of Phoenix for person-machine interface design is apparent since modifications to any interface were easily made and their impact immediately assessed. Using HAPS to represent experiment logic tables provided an analogous rapid prototyping environment for the implementation and modification of certain model information. Also, the data represented by HAPS rules are the beginning of a knowledge base which most certainly would be required in the event an expert system for experiment fault detection/isolation and system reconfiguration or intelligent training was needed. As indicated above, using HAPS allows for rapid future model/system modification and enhancement.

Future work will focus on two major areas: (1) continued development of the simulation director console which will automate a majority of tasks manually performed by the current simulation director, and (2) integration of advanced software/hardware with the system configuration already in place at the MSFC PCTC. Both of the these areas are considered crucial to future Spacelab and Space Station payload crew training. For Spacelab, work has already begun which examines the feasibility of integrating advanced software/hardware with existing PCTC software/hardware. What is proposed is control, through our software/hardware, the training simulation in the PCTC while at the same time continuing to use present displays and hardware for purposes of crew training continuity. For Space Station, the development of autonomous training simulation control, the capability for intelligent support in simulation design and the design of more effective crew interfaces are all areas which need to be examined.

The work described in this paper suggests several conclusions: first, the use of rapid prototyping and AI programming environments is an efficient and cost effective means of accomplishing payload modeling for purposes of crew training. Hierarchical, top-down modeling in combination with object-oriented programming appears to provide a flexible structure for rapid model design, implementation, modification and enhancement. Using this approach implies that (1) high-level, abstract models could be used for training prior to the availability of higher-fidelity models, (2) lower-fidelity models may be all that are required for some training applications and (3) if needed, low-level engineering math models may be added later; additions which could provide in-flight operational software. Second, better crew interfaces would reduce crew training time by easing memory load and providing more efficient system monitoring capabilities. Intelligent crew interface support could provide needed advice in the event of an anomolous condition, in either training or operational settings, and could carry out more routine procedural sequences now manually accomplished. Third, there appears to be significant potential for intelligent support in training simulation design and operation, for both Spacelab and Space Station.

ACKNOWLEDGMENTS

The authors would like to thank Bert Dolerhie at MSFC for serving as the NASA technical liaison for our work. Bert provided everything we required in a timely fashion and gave us some excellent insight into the workings of the PCTC as well as very useful comments on an early draft of this paper. We would also like to express our appreciation to Harvey Golden at MSFC who frequently took time out of his busy schedule to meet with us to discuss programmatic features of the work. Finally, Salem Suleiman was instrumental in a large portion of the actual coding and used HAPS to great effect; his efforts are greatly appreciated.

REFERENCES

1. McArthur, D. J., Klahr, P., & Narain, S. (1986). ***ROSS: An Object-oriented Language for Constructing Simulations.*** In P. Klahr, & D. Waterman (Eds.), Expert Systems: Techniques, Tools, and Applications (pp. 70-91). Reading, MA:Addison-Wesley.

2. Stefik, M., & Bobrow, D. G. (1986). ***Object-oriented Programming: Themes and Variations.*** AI Magazine, 6(4), 40-62.

3. Weinreb, D., & Moon, D. (1980). ***Flavors: Message Passing in the Lisp Machine.*** AI Memo No. 602, MIT Artificial Intelligence Laboratory.

C-4