N90-27316

AN APPLICATION GENERATOR FOR RAPID PROTOTYPING OF ADA REAL-TIME CONTROL SOFTWARE

Jim Johnson
Haik Biglari
Boeing Aerospace & Electronics, 499 Boeing Blvd., Huntsville, AL
Larry Lehman
Integrated Systems Inc, 2500 Mission College Blvd., Santa Clara, CA

Abstract - The need to increase engineering productivity and decrease software life cycle costs in real-time system development establishes a motivation for a method of rapid prototyping. The design by iterative rapid prototyping technique is described. A tool which facilitates such a design methodology for the generation of embedded control software is described.

Introduction

The software crisis

Due to the increasing complexity and size of software projects there is currently a software crisis. Software is on the critical path of project development time lines. It is usually late and over budget and often doesn't perform the necessary functions. One of the primary reasons for the crisis is that software development is a labor intensive and error prone task. In an effort to overcome these problems many automation tools have come into being to assist the programmer. In the area of real-time control software, these tools must assist hardware and software engineers as both are involved in the crisis.

The need for prototyping

Designers are faced with a dilemma. On one hand they need a plan prior to beginning design work and on the other hand they need some initial design analysis for the establishment of a reasonable and adequate plan. The two design philosophies in conflict are the "plan it" approach and the "do it" approach. Both approaches have considerable merit.

In the plan it approach the need for adequate preparation and specification is emphasized. Great pains are taken to ensure a fully specified and documented plan prior to the onset of preliminary design. The plan it approach attempts to avoid inadequate planning and the resulting failure which can be very expensive. The do it approach is to do a prototype design and learn from the mistakes of that initial design prior to the actual system design. The do it approach provides insight into design feasibility. It eliminates the costly and wasteful time spent designing systems on paper without the foggiest notion of whether or not a design will meet requirements. It is desired to combine the strong points of both approaches while avoiding the pitfalls of each.

On a large long term software project, changes in objectives (requirements) are inevitable, so as Brooks states, prepare for them.[1] During preliminary design, changes in development strategy and technique are typically frequent. According to Brooks, one should always have a prototype or pilot model from which to learn. "The throw-one-away concept is itself just an acceptance of the fact that as one learns, he changes the design." Plan to throw the first one away, or at least to modify it significantly.

Rapid prototyping in iterative design

On a large project it is not possible to prototype the entire project due to time and economic constraints. However it is still desirable to be able to use the prototype approach. The prototype method best for large projects is the iterative prototype method. Iterative prototyping implies many prototypes. This requires a tool capable of generating rapid prototypes for evaluation of incremental changes. The prototyping tool should be based on natural language of the intended user. In the control engineering domain this language would be based on block diagrams. A tool which provides such capabilities is the Application Generator (AG). The AG solution to the design dilemma fulfills the needs of adequate planning and design prototypes in a design methodology henceforth called "design by iterative rapid prototyping."

Copyright © The Boeing Company, 1989,
All Rights Reserved.

PUBLISHED WITH PERMISSION

The Application Generator

The Application Generator is an alternative to conventional application software development. It reduces the software life cycle cost associated with application software generation activity. The principle behind the AG is that process control engineers must perceive control concepts and strategies that require control of several processes within a particular plant simultaneously. To achieve this (see Figure 1,) the process

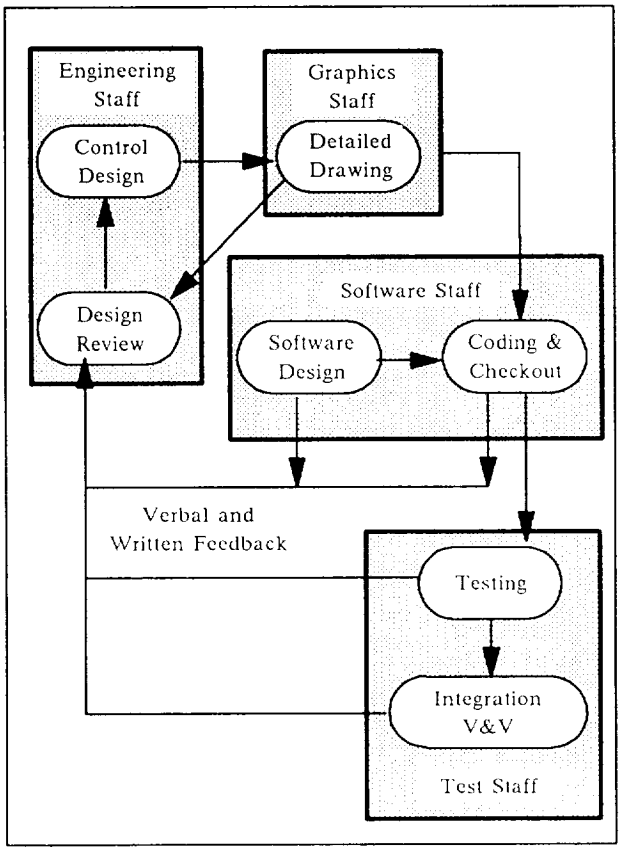

Figure 1 - Conventional Control Algorithm Development And Implementation

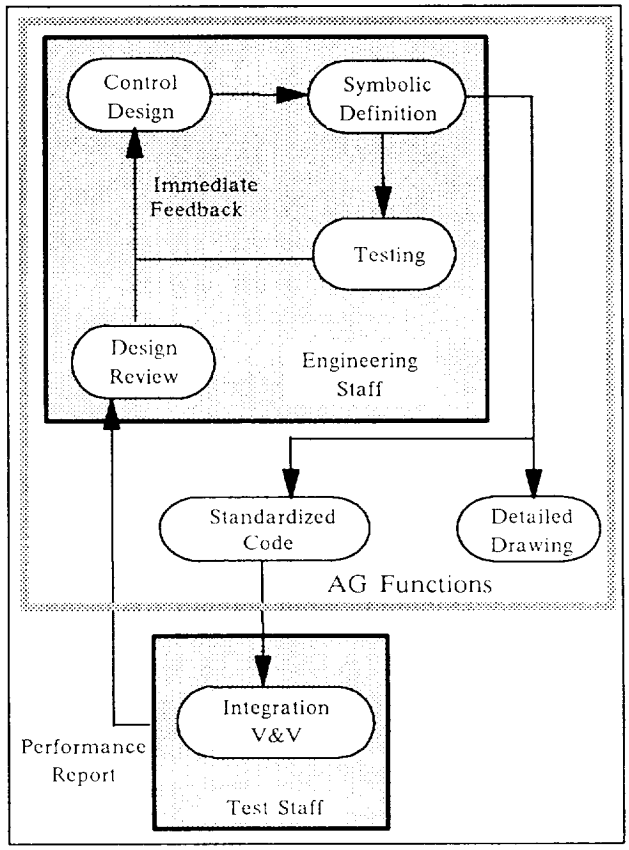

Figure 2 - Control Algorithm Development And Implementation Using the AG

control engineer draws spatial and temporal diagrams representing the processes to be controlled and derives the necessary control algorithm for each particular mode of operation. Conventionally, software engineers take this information and convert it to sequential codes suitable for a digital computer. Process control engineers evaluate the performance, take corrective action, and improve the control strategies. The result is submitted to software engineers and the cycle continues until a desired system response is obtained. The AG simplifies this procedure significantly by allowing the process control engineers to enter their control specifications directly into a computer and obtain analysis and simulation responses that they may use to modify their algorithms (see Figure 2).[2]

Ada Software Prototyping

Motivation

Development of prototype real-time software can be nearly as difficult and expensive as development of production software. Costs, debug time and tedious algorithm coding of embedded control systems are software development problems which can be addressed by an automated prototyping tool. Some of the features which should be included in the tool to make it an effective prototyping tools are a graphical interface, a means for providing interactive animation and automated generation of real time control software. The graphical interface should be a block diagram editor with icons familiar to control engineers. The modification, interconnection and manipulation of these blocks should be straightforward in a user friendly environment.

Reducing Software Life Cycle Costs

Conventional software development is modeled using the waterfall type phased development. The conventional software life cycle is shown in Figure 3. With the Application Generator, the software life cycle is

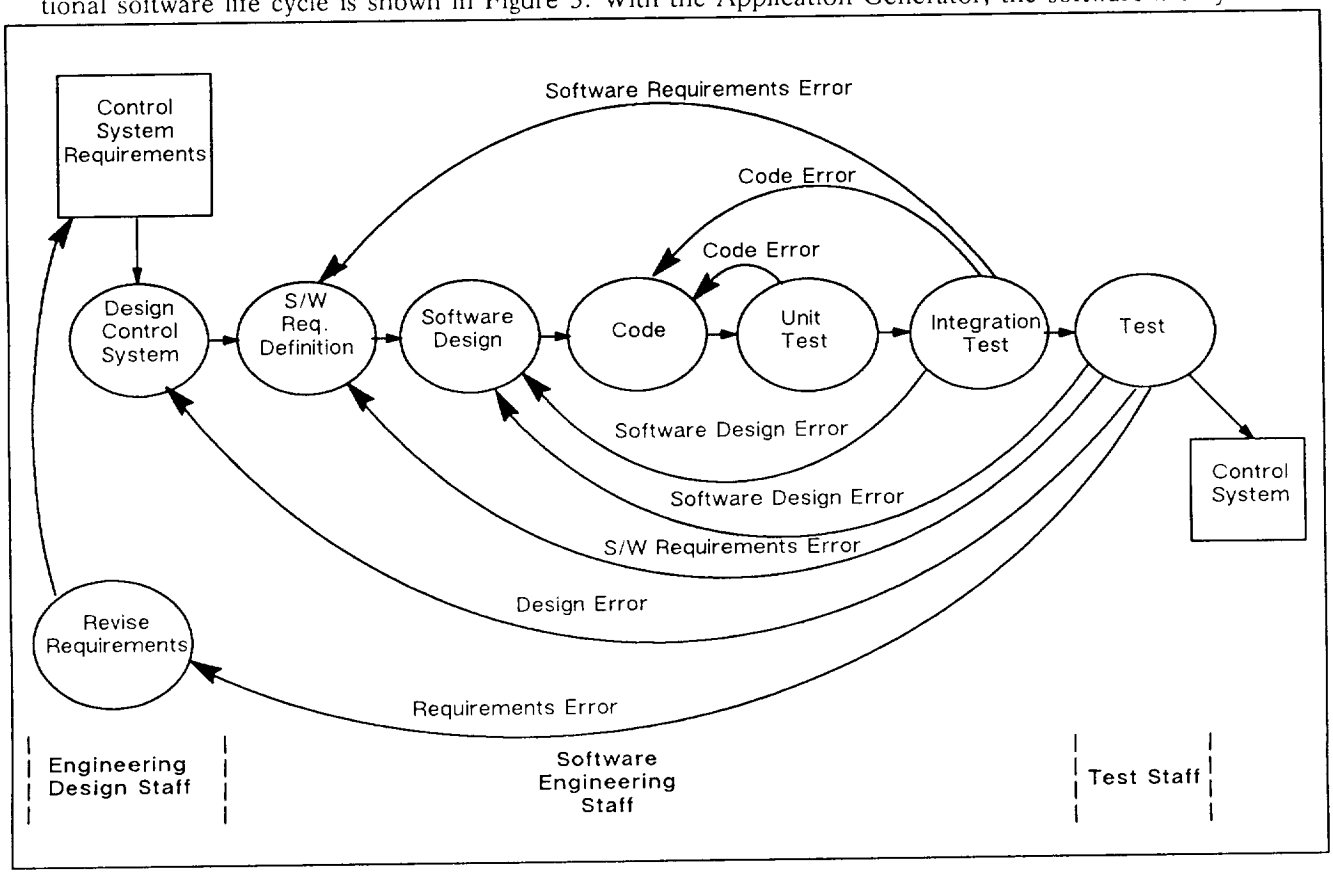

Figure 3 – Conventional Control System Software Development

modified as shown in Figure 4. The AG life cycle is modified by introduction of iterative rapid prototyping and an integrated toolset. Figure 4 displays how requirement and design errors can be discovered earlier in the life cycle and handled inside the AG process.

The relative cost to make a change increases throughout the lifecycle.[3] The overall life cycle is unchanged, but now there is tighter feedback, faster iteration. The design engineering includes concept, design, off-line simulation and prototyping. The software engineering includes prototyping, implementation and maintenance. The prototyping phase is included in both the engineering design and software engineering activities.

How the AG is used in the proof of concept, design, off-line simulation, prototyping, implementation and maintenance of control law development is detailed below:

a. Concept
   Proof of concept can be demonstrated in the modeling phase of design using the AG. Requirements specifications can be evaluated for feasibility as the models are simulated. The model serves as a baseline for development of control strategies.

b. Design
   Interactive graphical design using standard control engineering block diagrams facilitates an efficient and effective means to expedite design specifications and implementations.

c. Simulation
   Simulation of the modeled plant and control provides immediate feedback to the designer. It provides an indication of whether design goals can be achieved.

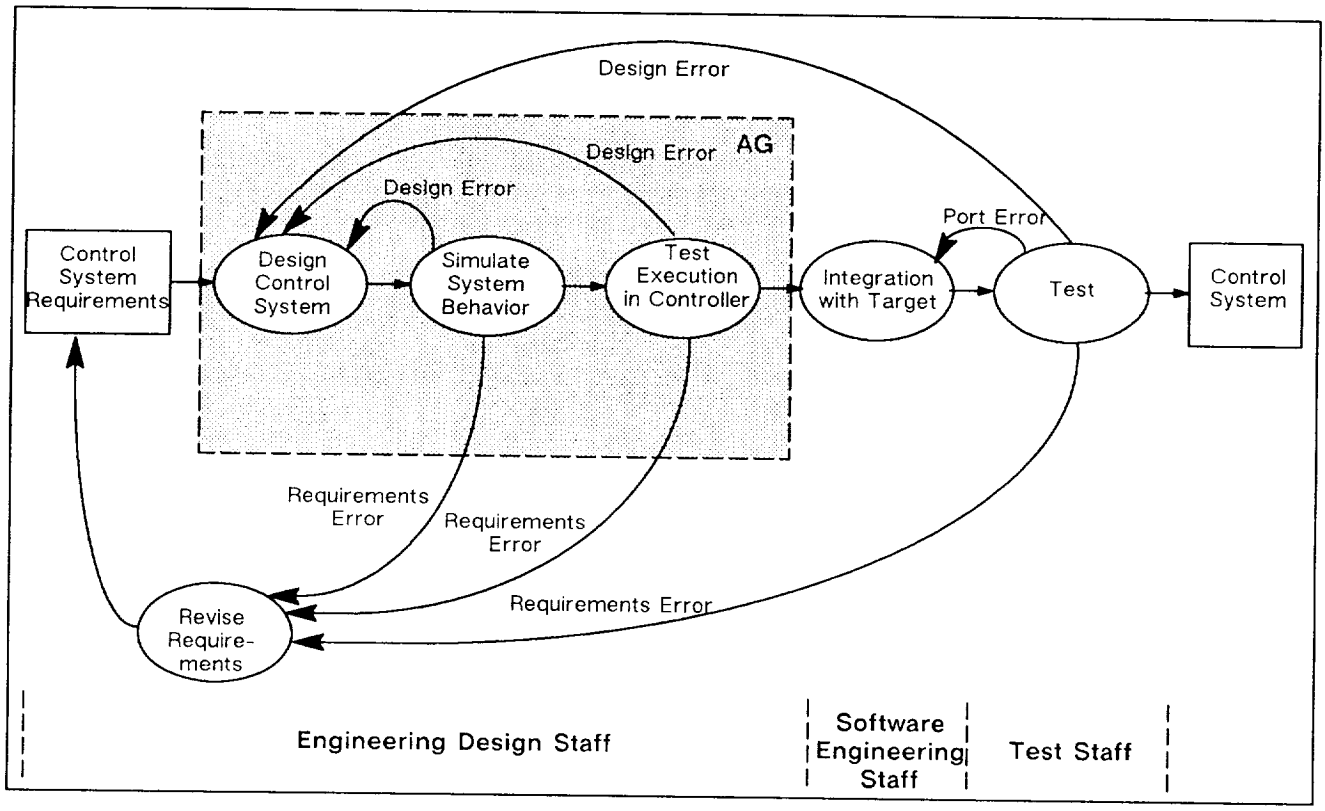

Figure 4 AG-Based Control System Software Development

d. Prototype

Rapid generation of prototype software enables the control engineer to verify design goals. It supplies the ability to test various control strategies, and it allows evaluation of alternatives and options to determine the best design.

e. Implementation

Because the AG controller emulates the target controller, implementation is simply a matter of determining porting and final integration issues. This can be performed once and used by all subsystems developers.

f. Maintenance

Maintenance is accomplished at the control block diagram level, which means that control diagram documents are automatically available and that they are maintained consistent with the control software. Also the control engineer has high visibility into the design maintenance.

Eliminating Software Coding Errors

The later in the life cycle that an error is detected, the more expensive it is to find and repair. It must be determined if the bug is in the software or in the control design. "Due to the introduction of bugs, program maintenance requires far more system test per statement written than any other programming." [1] Some methods to mitigate bug introduction in the implementation of designs are to find ways to use fewer people, fewer interfaces, and greater uniformity.

Automating Algorithm Coding

It is well understood that high level languages and self documenting techniques can be used to reduce errors induced by changes.[1] The control diagram level input is a self documenting high level language. There is a risk of error introduction in the translation of control diagrams to software code. This risk is minimized by automating the coding of algorithms directly from the block diagrams.

Variation among programmers in productivity [3] is a big problem in trying to estimate costs, but variation among subcontractors in coding style and software design philosophy can be a nightmare for software integration. On large programming projects, consistency of code from subcontractors is a major concern for a prime contractor responsible for integration. Code generated by the same tool, an Application Generator, is uniform across all subcontractors in terms of style, format and software design methodology.

System bugs arise from mismatched assumptions made by various people. [1] The assumptions and requirements are more highly visible at the block diagram level than when embedded in the syntax of a programming language.

All too often a software engineer is frustrated in attempts to debug an apparent software error when the actual problem is a design error. But in the numerous steps of translation from design drawings, to final coding, the error source is obscured.

Many design tools which speed up design work already exist. New features which are needed include a prototyping capability, a friendly user interface, high performance, deterministic behavior, and an environment to support increased levels of fidelity between the simulation model and the target code.

Project Description - Space Station Freedom

Background

Boeing's Space Station Freedom role is to develop the pressurized living, working, and storage areas under contract to NASA's Marshall Space Flight Center (MSFC). Onboard systems contained in these elements include thermal control, environmental control and life support, internal audio and video, and experiment facilities. The onboard Data Management System (DMS), which includes processors, networks, workstations, operating systems, network operating systems, data base management system, and a user interface language will be supplied to Boeing by NASA. However, Boeing must develop and test application software for the onboard systems before the DMS components are available. An interim development system is required to rapidly design and control the onboard systems. The Application Generator (AG) concept was selected for the development system because of its productivity potential and usability by control systems engineers. [2]

Requirements For Application Generator Tools

Large systems such as Space Station Freedom have numerous processes that require a large number of controllers to control and monitor the entire system. Such control is best achieved if a local area network is utilized that is capable of having many controller nodes. It is also necessary to have programmability for the controllers so that future needs can be accommodated with minimum cost and effort.

To further reduce systems operation costs, commonality must be maximized. One method of achieving commonality is to establish a set of standard control system icons. A data table corresponding to the selection and organization of these icons in a control design can be used by a common code generator. Commonality can also be maximized by having a common run time kernel. At the workstation level the process and the control algorithms are defined as a data structure which is used to generate high level source code. This source code is cross-compiled, linked with the run time kernel, and down-loaded to the controller's RAM. This provides an environment that supports all phases of development and testing (see Figure 5).

Four categories of requirements are detailed in the following paragraphs. They are software requirements, user interface requirements, integrated toolset requirements, and hardware requirements.

Software Requirements

The three main software requirements are design simulation, rapid prototyping capability, and generation of reusable software modules.

a. Design Simulation
   Simulation of the system with Ada executing on the AG controller provides an easy means of testing software design.

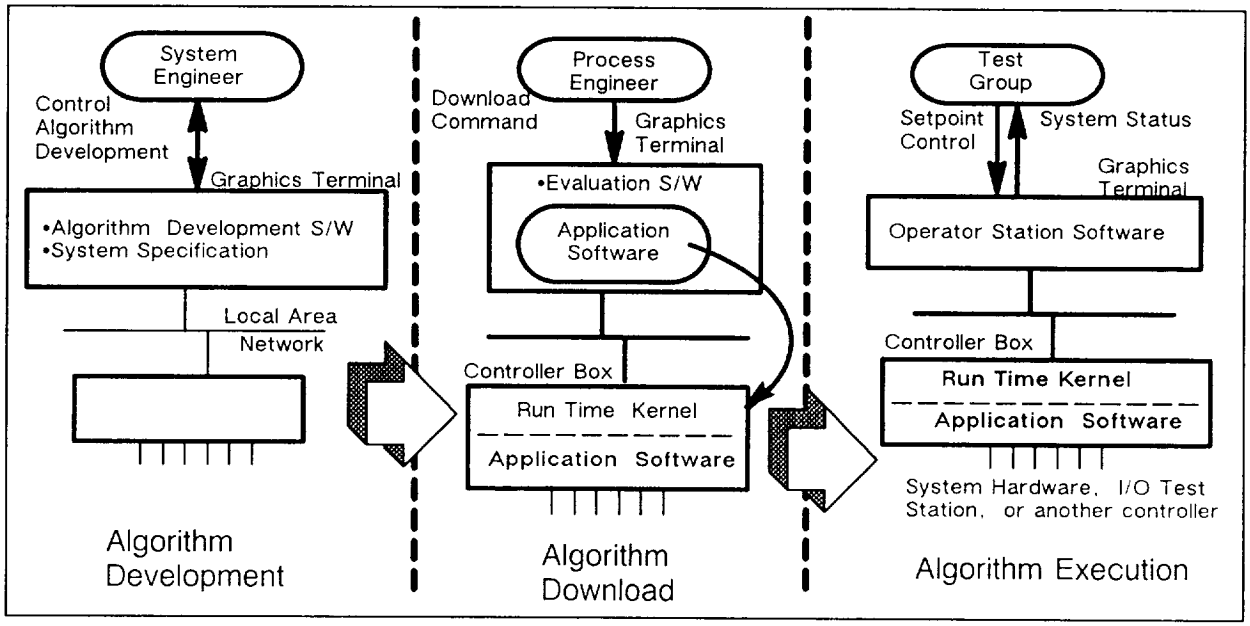

Figure 5 - AG Software Generation Phases

b. Rapid prototyping

The process of rapid prototyping makes requirement specification, design, simulation, validation, and hardware/software trade-off analysis an integral part of the design and development process. [4] The standard waterfall type development alternative treats these tasks as afterthoughts, as if they were simply by-products of the coding phase.

c. Reusable Software

AG supplied control diagram blocks are a form of reusable software. Blocks can be grouped together into user designed super blocks. These super blocks can also be cataloged making them available for reuse.

User Interface Requirements

A rapid prototyping environment requires a user friendly graphical interface and animation capabilities as described below:

a. Graphical Interface

The AG workstation software uses multiple windows, pull-down menus, and mouse input for operator interaction. The major components of the workstation software are the interactive animation and graphical programming environment utilities.

b. Graphical Programming Environment

Using standard control engineering diagram icons available in the software, control engineers are able to input control law algorithms efficiently.

c. Interactive Animation

To test the functionality of the real-time control software in a typical set of environmental and user inputs, it is desirable to have an interactive animation capability. (see Figure 6). Note that the interactive animation ties in directly to the control system on the AG-100 controller. Interactive animation is used in simulation, software prototyping and hardware prototyping. The interactive animation schematic contains icons which represent actual hardware and its associated sensors and actuators.

Integrated Toolset

It is advantageous to have an integrated toolset which resides on one workstation. Some of the benefits are that there is only one set of workstation environment software to learn, data transfers between various tools are local to a workstation and all applicable data sets are resident on the local workstation. The types of tools which comprise the toolset are listed below.

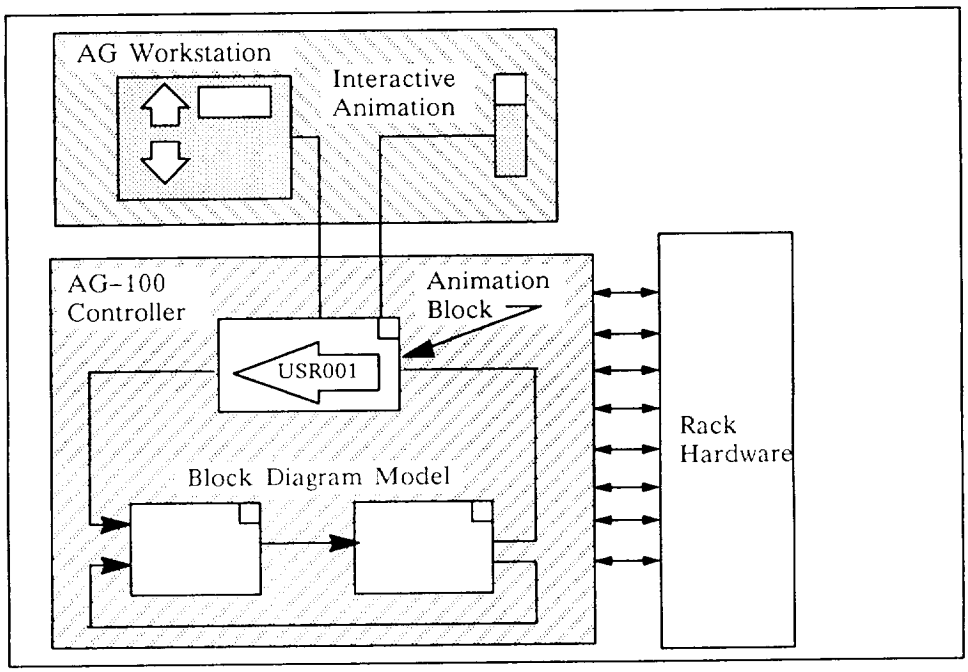

Figure 6 - Rack Hardware Test

a. Design and Simulation Tools
b. Software Support Tools
c. Documentation Tools
d. Project Management Tools
e. Data Dictionary
f. Interface To Other Engineering Databases

Hardware Requirements

The AG hardware components are the workstation, the controller, the optically isolated I/O modules, and the local area networks.

a. Workstation
The AG workstation is a VAXstation 3100 graphics workstation with 19" color monitor using a DEC-window user interface. It has 8 Mbytes of memory, 208 Mbytes of disk storage and thinwire/thickwire Ethernet network capabilities.

b. Programmable Controller
An embedded data processor (EDP) emulation using 80386 and 80186 message processor is provided by the AG controller. The programmable controllers operate in real-time and have multi-tasking capability, allowing them to run several different rate control algorithms simultaneously.

c. Optically Isolated I/O Module Rack
A low bandwidth noise suppression, noise tolerant I/O capability is necessary for the Space Station Freedom environment.

d. Local Area Networks
There are two communications networks, the Ethernet for Host/Workstation/Controller communications and the Token Bus for Controller/Supervisor communications. The Ethernet has a 10 megabit/second data rate to facilitate a common communication medium used to connect workstations, controllers, and user-provided equipment.

Using The AG On Space Station Freedom

The AG is used to aid in the integration of low level code modules into rack level hardware and system level software. AG generated code has not yet been approved as actual flight code, so current use of AG code is restricted to prototyping of hardware and software.

Real Time Embedded Control Processor

The Application Generator includes hardware and software to support the embedded controller. The tools which support the embedded controller are integrated with the tools used to develop control algorithms allowing for rapid iterative prototyping.

For real time systems there is the special need for Ada tasking, and schedulers. The task of generating real time embedded control software is complicated by the extra implementation steps of cross compilation and downloading as well as the more primitive debugging tools common to embedded controllers.

Flight Hardware Transition

It is proposed that the AG controller serve as a prototype for the flight controller. As components of the flight hardware such as flight qualified I/O modules become available they will be inserted into the AG controller. This will help to provide a smooth transition from the AG controller to the actual flight controller.

The most challenging tasks expected during this transition are converting from AG runtime kernel to the EDP operating system, the I/O module fidelity, and the speed of the I/O. Errors discovered here are the porting errors (See Figure 4). These are the only class of errors in an AG based control system software development which do not result in changes at the control diagram level. Note that correction of the porting errors does not require a redesign of the control system or the re-coding and unit testing of control algorithms.

System Design

The architecture of software implementation is common across all subsystems. Control engineers use simulation models and software engineers use configuration, implementation and customizing tools. Configuration management of models is implemented in a code management system similar to the DEC code management system, CMS.

The hierarchy of control diagram blocks into super blocks greatly facilitates the design of large systems on the AG. Merging of control algorithms is more readily accomplished at the block diagram level than at the code level.

Integration In a Large Scale Development

AG generated software must communicate with higher level software used to manage an entire Space Station Freedom element, such as the Habitation element or Laboratory element(see Figure 7). The

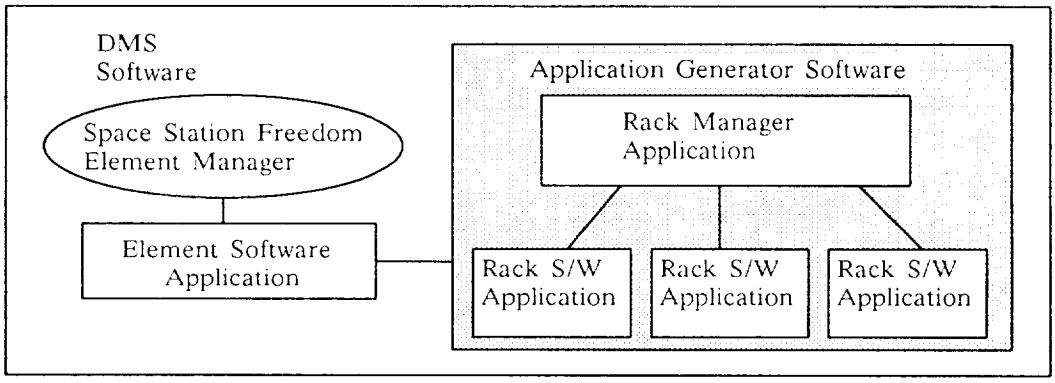

Figure 7 – AG Interface to Space Station Freedom Element

high level interfaces are modeled on the AG in the form of messages to and from the low level AG control algorithms. The AG is limited to development of software in individual racks where each rack typically has one controller. Software at a higher level to control an entire Space Station Freedom element, is not developed on the AG.

Training

A user friendly workstation environment helps to mitigate the complexity of a sophisticated tool such as the AG. Formal training accelerates the learning process. A major objective of training is to encourage the control engineer to become familiar with all phases of design and implementation using the AG. This is consistent with a primary objective of the AG, which is to keep the control engineer in the loop throughout all phases of development.

Summary

Interactive rapid prototyping is a key element in the software life cycle. It allows completely new control software to be developed rapidly so that alternative control strategies can be evaluated in a cost effective manner. It also provides the ability to address system integration and performance issues early in the development cycle.

The Application Generator provides a flexible environment for development of robust process control systems. It is based on automatic code generation directly from control block diagrams. A highly intuitive and interactive environment is used to specify mathematical models of the hardware and implement necessary control strategies.

Acknowledgments

We thank the numerous people who encouraged and supported us and specifically we extend our appreciation to Bill Walker and Roger Williams for their valuable comments and suggestions.

References

[1] Brooks, F.P., Jr: *The Mythical Man-Month*, Addison-Wesley, 1982

[2] Boeing, *Application Generator Statement of Work*, D683-10094-1

[3] Fairley, R.: *Software Engineering Concepts*, McGraw-Hill, 1985

[4] Tanik, M.M. and Yeh, R.T. "Rapid Prototyping in Software Development" *IEEE Trans Comp* vol c. 22 no 5 , May 89