Proceedings of the 8th IEEE International Symposium on Intelligent Control, Chicago, IL
September 25-27, 1993.

A SOFTWARE TEMPLATE APPROACH TO BUILDING COMPLEX LARGE-SCALE INTELLIGENT CONTROL SYSTEMS

Richard Quintero
Robot Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899
and
A. J. Barbera
Advanced Technology and Research Corporation
15210 Dino Drive, Burtonsville, MD 20866-1172

Abstract

This paper presents a task oriented systems engineering approach for developing complex, large-scale, integrated, intelligent machine systems using software templates. This methodology can be applied to a wide variety of real-time machine applications. The method presented here is built upon the *Real-Time Control System (RCS)* Reference Model Architecture being defined by researchers in the Robot Systems Division at the National Institute of Standards and Technology (NIST). We believe that the reference architecture and methodology described here could form the basis for defining an open-systems architecture for intelligent control systems applications.

Introduction

This paper is a condensed version of [1]. The methodology presented here is based upon a task oriented systems engineering approach originally conceived by Barbera [2]. The methodology also complies with the *Real-Time Control System (RCS)* Reference Model Architecture published by Albus [3, 4]. Researchers at the National Institute of Standards and Technology (NIST) are exploring several RCS implementation approaches in addition to this one. Each of these approaches are generally optimized for a particular class of RCS applications.

The method described here is particularly well suited to applications which are rule driven (i.e., they employ strategies, tactics, and process knowledge). Such applications are characterized by a need to monitor sensory input to detect events and objects in the intelligent system's environment. These systems use sensed events to trigger desired system activities and to react to exception conditions. The methodology also accommodates control systems problems dealing with path planning, trajectory generation, and control law algorithms.

Product Endorsement Disclaimer

References to specific brands, equipment, or trade names in this document are made to facilitate understanding and do not imply endorsement by the National Institute of Standards and Technology.

Background

Early work by Albus [5] and Barbera [2], in the Automated Manufacturing research Facility (AMRF) [6], gave rise to the first definition of a Real-Time Control System (RCS) systems engineering approach focusing primarily on software design. This approach was derived from a control systems engineering perspective rather than a data processing perspective. The Robot Systems Division has refined and evolved these techniques by applying the RCS approach to a number of robotic problems in manufacturing as well as robotic applications intended for unstructured environments (see [3],[4],[7],[8],[9],[10], and [11]).

RCS versions have been implemented using the FORTH, C, and ADA languages and running on 680x0 series processors as well as on 286/386 machines and on Multibus and VME backplanes. Applications have been built using real-time operating systems such as: GRAMPS, pSOS, and VxWorks. RCS applications have also been hosted within the DOS operating system on personal computers (PCs).

The approach presented in this paper is based on our most recent work in applying RCS techniques to

the automation of submarine maneuvering control, under Advanced Research Projects Agency (ARPA) sponsorship, and in demonstrating the automation of a continuous coal mining machine, under the sponsorship of the U. S. Bureau of Mines. This work is being carried out by a team of researchers from NIST and the Advanced Technology and Research (ATR) Corporation.

What is a Control System Architecture?

The Random House College Dictionary, [12], defines ***architecture*** as "the character or style of building; the structure of anything". The RCS Architecture is a style of building real-time intelligent control systems. These systems generally include software, hardware, machines, people, communications, information repositories, information/knowledge models and real-time software execution models. The RCS Architecture defines a highly structured, modular organization of these control system components.

The basic building block of our implementation approach is a ***controller module***. A controller module does not contain any submodules but it may encapsulate any number of functions, subfunctions, or processes. A controller module can be viewed as a systems integration "wrapper" which is implemented as a template. We encapsulate software within this wrapper to ensure compliance with our integration rules. Every module built using the controller module template inherits a software execution model, a communications mechanism, performance measurement capabilities, and debug mechanisms. All of these properties are discussed in detail in [1].

The RCS Problem Domain

RCS specifically addresses intelligent machine control systems problems. We define ***intelligent machines*** to be machines designed to perform useful physical work while employing in situ knowledge (sensory input data), and a priori knowledge, tactics, and strategy. Intelligent machines use feedback from the physical environment to manifest "intelligent behavior" in real-time via computerized real-time control of the machine's electro-mechanical actuators and sensors. In addition, we believe that practical intelligent machines almost always require some level of human interaction. The definition given above is intended to include: ***automation systems, embedded systems,*** and ***robotic systems*** ranging from factory floor robots to space vehicles and planetary exploration robots.

In priority order, our objectives in developing RCS are to:

1) Improve human understanding of the design.
2) Manage software complexity.
3) Provide for robust, verifiable, efficient, coordinated, real-time performance.
4) Provide for extensibility, portability, and software reuse.

Two Robot Systems Division papers [13] and [1] elaborate on these objectives.

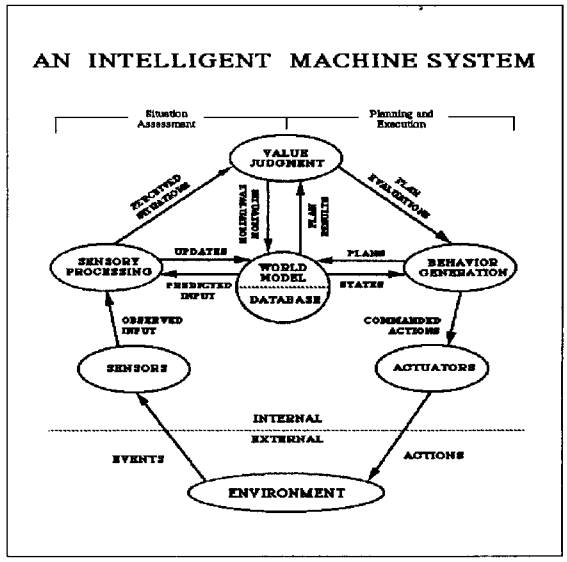

Figure 1.

The RCS Architecture Reference Model

A comprehensive treatment of the RCS Architecture is contained in the following publications: [2],[3],[5],[14],[15], and [16].

Albus models the most primitive form of an intelligent machine as a closed-loop control system (see Figure 1.). A closed-loop system is formed in the machine by inputting sensory data to ***Sensory Processing (SP)***, passing the processed information off to the ***World Modeling (WM)*** function, which maintains the machine's best estimate of the state of its world, and finally closing the loop through ***Behavior Generation (BG)*** which plans and executes actions to be performed through the machine's actuators. The ***Behavior Generation (BG)*** function is further decomposed into ***Job Assignment (JA), Planner (PL), and Executor (EX)*** functions. The value system, or the ***Value Judgment (VJ)*** function, is used in goal selection to

direct Behavior Generation in selecting alternative plans and actions.

The RCS Reference Architecture, defined by Albus, extends the notion of an intelligent machine design containing the basic SP, WM and BG functions by creating a hierarchy. Each controller is assigned a set of tasks at an appropriate level of abstraction and each has a limited range of authority and responsibility within the chain-of-command formed by the RCS hierarchy (much like a human military command structure would be organized).

The terms, ***World Model or World View***, are used to describe the intelligent machine's collective capability to perceive the world in which it functions (both external and internal).

Global Memory (GM) is the complete collection of globally defined variables in an RCS application. In many applications GM is implemented in a distributed manner.

In RCS, there is a notion of decomposing the control system design into an indefinite number of layers or levels of abstraction. See [3] and [1].

RCS Method Tenets

We use the word tenet, here, to mean guidelines and engineering rules of thumb which characterize this RCS Methodology approach. Together the RCS Architecture definition and these tenets form a basic set of rules or systems integration standards for building real-time control systems. Tenets 1) through 5) are generally applicable to all RCS Methodology approaches while tenets 6) through 10) are expressed in terms of the Barbera approach. An in-depth discussion of these tenets is presented in [1].

1) Use task oriented decomposition (driven by scenarios)
2) Use hierarchical organization and assign responsibility and authority
3) Organize the control hierarchy around tasks top-down and equipment bottom-up
4) Partition by an order of magnitude between levels (spatial and temporal resolution) and roughly ten decisions or less per plan
5) Use seven + or - two subordinates per supervisor and only one supervisor at a time
6) SP/WM/BG functions are distributed throughout RCS and assumed to exist in each node
7) Allow human I/F at any node
8) Controller modules are finite state machines communicating through Global Memory
   * Use a controller template as the basic building block
   * Use cyclic sampling rather than interrupts for context switching
   * Surround everything with data buffers
   * Use non-blocking input/output (I/O)
   * Implement Global Memory using a One Writer, Many Readers Paradigm
   * Match the control cycle time to the demands of the control application
9) Design for concurrent processing
   * Measure execution time performance
   * Allocate sufficient computing resources
10) Use synchronous control at the lowest levels, transitioning to asynchronous control at the highest levels

RCS Plans

An RCS control system can be viewed as an integrated collection of finite state machines which are capable of selecting or generating and executing RCS plans in real-time. The RCS Methodology described here uses both rule plans and path plans.

RCS Rule Plans are uninstantiated plans (or plan schemas) which can be represented using some form of *If-Then-Else* construct. Rule plans specify branching conditions and they can be represented by state transition diagrams. ***RCS Path Plans*** are ordered sets of instantiated poses, knot points, commands, or other variables specifying a sequential order of execution. Path plans do not specify branching conditions. A rule plan is required to specify conditions to be monitored in order to interrupt the execution of a path plan for branching (i.e., out-of-tolerance condition branching). In addition to rule and path plans, RCS also accommodates goal-point generators or control-law algorithms. Such algorithms (typically mathematical) are used to generate the next goal-point for an actuator movement at the Servo-Level or the next goal-command for a subordinate module at other levels of an RCS hierarchy.

We use state graphs and state tables to represent RCS rule plans as illustrated in Figure 2. These plans embody strategies, tactics, and process knowledge. A rule plan is a set of uninstantiated rules for accomplishing some task. Rule plans are used throughout the RCS Architecture. Rule plans

are often very simple at the low levels and more complex at the higher levels.

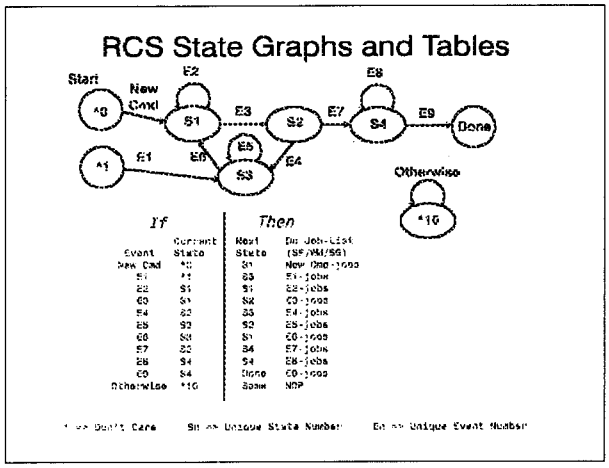

Figure 2.

Implementing a Controller Template, the Basic RCS Building Block

Building understandable large systems requires defining systems integration standards. The RCS implementation approach described here requires only two basic building blocks: An RCS Controller Template and a Main Program Template.

Our generic RCS controller module template provides a software execution structure within which the basic functions of Sensory Processing, World Modeling, and Behavior Generation may be implemented. In addition this model addresses the interface definitions required to integrate a set of controllers to form an RCS hierarchy. A controller module must be able to accept task commands from its superior and send commands to its subordinates or to actuators if the controller is at the lowest level of its branch in the RCS hierarchy. It must be capable of accepting status from its subordinates and sending status to its superior. A controller module must be capable of accommodating a human interface and it must have the capacity for communicating with other controllers and the knowledge base through some set of Global Memory communications mechanisms. A controller must also be capable of directly accepting sensor data for processing.

Since a controller module is a finite state machine, its response to stimulus is deterministic for any given execution cycle. Its output is only a function of its current state and its input event space. Furthermore, its execution time can be measured or calculated for every event-state pair in a given plan.

A controller module built from an RCS Controller Template performs Preprocessing, then Behavior Generation (also referred to as Decision Processing), followed by Post-Processing on each control cycle.

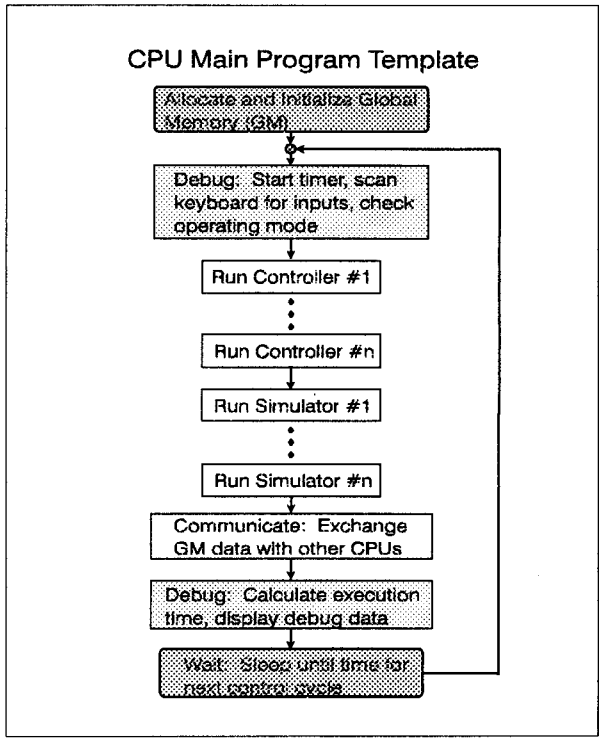

Figure 3.

Multi-Tasking on a Shared CPU, using a Main Program Template

Multitasking within a shared CPU is implemented in RCS with a CPU Main Program Template (Figure 3.). The Main Program allows initialization of CPU parameters, such as declaring global variables (externals), and the loading of starting values in Global Memory.

After initialization the Main Program begins running the heartbeat control cycle for the CPU. First, a Debug function runs to check for operator inputs indicating a change in operating mode (e.g., debug single step, normal run mode, etc.) and to start a timer to measure control cycle execution time. Once that completes each RCS controller module (including simulation modules) runs in sequence according to the precedence order of the execution schedule established by the programmer. Of course, in compliance with the RCS tenets, all of the controllers must be able to complete their

execution within the established heartbeat control cycle time. If the controllers overrun the cycle time then the RCS designer must reassign one or more to different CPUs or increase the cycle time.

At the end of each control cycle the communications controller modules are executed to exchange Global Memory data with other CPUs within the backplane and over any networks being used. The total execution time is calculated as a last step to be sure that the heartbeat control cycle time has not been exceeded (a debug error is posted if it ever does). At that point the Main Program enters a wait loop (or goes to sleep) until time for the next control cycle.

RCS Target Hardware and Operating Systems

Any multiprocessor backplane hardware suite may be used (e.g., VME, Multibus, Nubus, etc.) for implementing an RCS control system. Using this type of hardware suite in an RCS implementation makes it very easy to tailor the hardware selection and the communications network to meet the real-time requirements of the application. It is also very easy to extend such systems as the system evolves.

"Hard" real-time applications must respond to the physical environment within some set of time limits. Therefore, only operating systems with real-time (RT) extensions such as memory-locking (e.g., VxWorks, Lynx OS, RT-Posix, etc.) or single user operating systems, like DOS, should be considered as host environments.

RCS Methodology Development Steps

The methodology described here should be interpreted as an iterative, "rapid prototyping", real-time software development method. The steps listed in Table 1 are roughly in the sequential order of a first pass through the method to achieve a skeleton of the overall RCS architecture to be implemented. Once a skeleton is developed, the developer(s) should iterate within the steps to develop executable controller modules in a bottom-up process.

Table 1. Summary of the RCS Methodology Steps

1) Concept Development
   A) Gather domain knowledge.
   B) Develop the problem description / scenario
   C) Conceptualize the Controller Hierarchy, the Operator I/F System, the Data Management System, and the Communications Management System.
2) Design the Hierarchy using Task Decomposition
   A) Develop a task tree.
   B) Choose a "thread" of tasks spanning the tree.
   C) Design by iteratively adding task threads.
   D) Design software by adding detail using generic RCS templates in a bottom-up process.
3) Coding and Testing RCS Software
   A) Incrementally develop code for each controller in a bottom-up fashion.
   B) Incrementally develop simulators to drive each controller in a closed-loop fashion.
   C) Develop simulators for the human interfaces.
   D) Measure the execution time of each controller.
   E) Map the controller modules (software processes) onto the computer hardware.
4) Port the software to the target hardware system.
5) Incrementally integrate and test the system.
   A) Perform lab tests.
   B) Perform field tests.
6) Develop a simulator to animate the robotic system in the envisioned physical environment (workspace).
7) Design, code and test the Operator I/F System, Data Management System and the Communications Management System
8) Integrate the RCS Controller Hierarchy with the Operator I/F System, Data Management System, and the Communications Management System.
9) Produce final documentation for the system.
10) Iterate all of the steps above extending the RCS system, in a "rapid prototyping" fashion, by adding new controllers and/or processing modules to execute additional task threads.

Conclusions

In this paper we have attempted to begin to define a consistent set of systems engineering rules for building, evolving, and maintaining large, complex, intelligent control systems. In our approach:

1) We build on the work of Albus, Barbera, and others over the last two decades.
2) We use task scenarios in the knowledge engineering process.
3) We emphasize hierarchical organization as a powerful method of complexity management.
4) We have selected cyclic sampling and the finite state machine as our execution model in order to ensure our designs are deterministic and verifiable.

5) We have emphasized rule plan knowledge models (state graphs and state tables) which are compatible with and can be directly executed by finite state machines.
6) We use a primitive communications mechanism (GM) which is compatible with cyclic sampling and provides for non-blocking I/O.
7) We have defined generic RCS Controller Module Templates and the RCS Main Program as our basic systems integration wrapper mechanism to simplify the development and integration process.
8) We have presented an outline of a set of rapid prototyping steps which can be used as a systems development life cycle approach.

The NIST Robot Systems Division is currently conducting a long term research program, called the Intelligent Machines Initiative, which is focusing on SP and WM for machine vision as well as many of the other issues not addressed here in detail.

References

[1] R. Quintero and A.J. Barbera, "An RCS Methodology for Developing Intelligent Control Systems," NISTIR 4936, October 1992.
[2] A.J. Barbera, J.S. Albus, M.L. Fitzgerald, and L.S. Haynes, "RCS: The NBS Real-Time Control System," Robots 8 Conference and Exposition, Detroit, MI, June 1984.
[3] J.S. Albus, H.G. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," NIST (formerly NBS) Technical Note 1235, April 1989 Edition.
[4] J.S. Albus, R. Quintero, H. Huang, M. Roche, "Mining Automation Real-Time Control System Architecture Reference Model (MASREM)," NIST Technical Note 1261 Volume 1, May 1989.
[5] J.S. Albus, Brains, Behavior and Robotics, BYTE/McGraw- Hill, Petersborough, NH, 1981.
[6] J.A. Simpson, R.J. Hocken, and J.S. Albus, "The Automated Manufacturing Research Facility of the National Bureau of Standards," Journal of Manufacturing Systems, Vol.1, No. 1, pg. 17, 1983.
[7] H. G. McCain, et al., "A Hierarchically Controlled Autonomous Robot for Heavy Payload Military Field Applications," Proceedings of the International Conference on Intelligent Autonomous Systems, Amsterdam, The Netherlands, December 8-11, 1986.
[8] J.S. Albus, "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles Project," NIST Technical Note 1251, September 1988, p. 126.
[9] S. Szabo, H. A. Scott, R. D. Kilmer, "Control System Architecture for the TEAM Program," Proceedings of the Second International Symposium on Robotics and Manufacturing Research, Education and Applications, Albuquerque, NM, November 16-18, 1988.
[10] H. Huang, J. Horst, R. Quintero, "A Motion Control Algorithm for a Continuous Mining Machine Based on a Hierarchical Real-Time Control System Design Methodology," Journal of Intelligent and Robotic Systems 5: 79-99, 1992, Kluwer Academic Publishers, Dordrecht, The Netherlands.
[11] H. Huang, R. Quintero, J.S. Albus, "A Reference Model, Design Approach, and Development Illustration toward Hierarchical Real-Time Control System for Coal Mining Operations," Control and Dynamic Systems: Advances in Theory and Applications Volume 46: Manufacturing and Automation Systems: Techniques and Technologies, Part 2 of 5, Edited by C. T. Leondes, Academic Press,1991.
[12] Random House College Dictionary, 1982, Revised Edition
[13] J.S. Albus, R. Quintero, R. Lumia, M. Herman, R.D. Kilmer, K.R. Goodwin, "A Reference Model Architecture for ARTICS," ASME and IIE, Manufacturing Review Volume 4, Number 3, September 1991.
[14] J.S. Albus, "Outline for a Theory of Intelligence," IEEE Journal, Transactions on Systems, Man and Cybernetics, Volume 21, Number 3, May/June 1991.
[15] J.S. Albus, "The Role of World Modeling and Value Judgment in Perception," Proceedings of the Fifth IEEE International Symposium on Intelligent Control, Philadelphia, PA., September 5-7, 1990.
[16] J.S. Albus, "Hierarchical Interaction Between Sensory Processing and World Modeling in Intelligent Systems," Proceedings of the Fifth IEEE International Symposium on Intelligent Control, Philadelphia, PA., September 5-7, 1990.