Coal extraction using RCS

John Albert Horst
Robot Systems Division
The National Institute of Standards and Technology (NIST)
U.S. Department of Commerce

Anthony J. Barbera
Advanced Technology and Research Corp. (ATR)
Laurel, MD

ABSTRACT

Application of the Real-time Control System (RCS) reference model to a simulated underground coal mining machine is described. RCS is characterized by explicit software modules that perform task decomposition, sensory processing, and world modelling functions at different hierarchical levels. We use a detailed and sharply defined approach to RCS design characterized by task-oriented problem analysis, generic software "objects," rule-based control (using finite state machines), cyclic execution of manually scheduled processes, and generic communications interfaces.

1. BACKGROUND

The theory, design, testing, and implementation of large-scale intelligent control systems has been the focus of much effort at the Advanced Research and Technology Corporation (ATR) and the National Institute of Standards and Technology (NIST). NIST is particularly concerned with the development and dissemination of reference models that will improve all aspects of real-time control system development. An example of such a model developed at NIST is the Real-time Control System (RCS) model [Albus 89, Albus 92, Quintero 92, Huang 92].

The RCS reference model specifies hierarchical levels of control where each level has a characteristic spatial and temporal purview. The critical components of the system are control, sensing, and world modelling. In RCS, each hierarchical level can have multiple modules each of which contains sensing, world modelling, and control functions. The approach to RCS design described in this paper is a more sharply defined version of RCS than that described in Albus 89. For the sake of clarity we will refer to this approach as RCS throughout the paper.

Our focus is the current status of an implementation of RCS for the control of a continuous mining machine used in underground coal mining. This work is being performed by NIST and ATR in support of the US Bureau of Mines (BOM). BOM is involved in a long term effort to bring aspects of underground coal mining under computer control in order to enhance the health and safety of coal miners.

2. CONTINUOUS MINING MACHINE CONTROL SYSTEM DESIGN USING RCS

The US Bureau of Mines (BOM) is developing prototype systems that perform all the typical tasks of a continuous mining machine (CM) under computer control [Schnakenberg 92]. These tasks can be grouped into two general areas, namely, free space motion and cutting.

Continuous mining machines have appendage controls that are of the on-off or "bang-bang" type. This type of actuation in concert with the complicated coordination and sequencing required by the coal cutting control task is well served by a rule-based design approach like RCS.

2.1 SCENARIO

This section gives a realistic scenario of the operation of a CM performing a box cut as might be performed in an underground coal mine. The highest level task this control system has implemented is a box cut of operator specified length. A drawing of a CM executing a box cut is in figure 1.

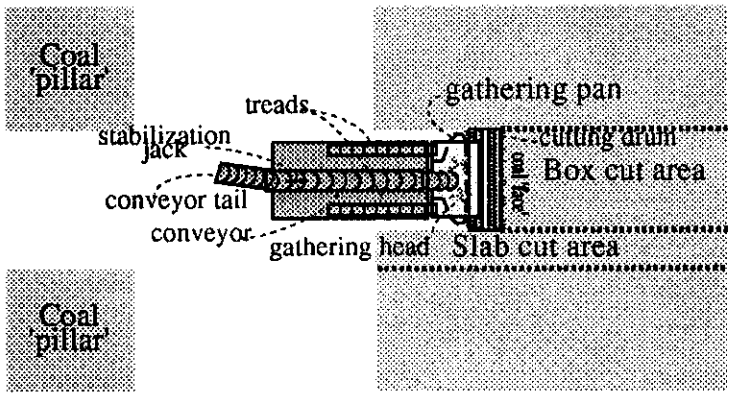


figure 1: A continuous mining machine executing a box cut

A box cut is ready to be performed when the CM is within a few feet of the coal face. When in this position (assuming no other errors), the cutting drum is raised to the proper height and turned on. The CM trams forward toward the coal face. During this initial tram forward, the current load on the cutting drum motor is monitored. If this load exceeds threshold, the CM has contacted coal and is at the face (the place where the cutting operations occur). In addition, the orientation of the CM is closely monitored for deviation from the proper orientation. This is called the initial_approach_face task.

Within a box cut operation, the initial_approach_face task precedes the first execution of a sump_shear_cusp task. After completion of the initial_approach_face task, the sump begins with the drum at the ceiling and rotating. During a sump the following occurs: 1) the gathering pan is put in the float position, 2) a command is sent to an operator to put the tail of the conveyor belt into the correct position for deposit of coal into a shuttle car or continuous haulage conveyance, and 3) the CM trams forward about half the diameter of the cutting drum. This event triggers the start of the shear operation.

During a shear, the following occurs: 1) the stabilization jack is lowered, 2) the cutting drum remains on and is lowered, and 3) the gathering head and conveyor are turned on. The latter commences only if enough coal has piled up to warrant turning on the conveyance systems and a status message is received from the operator interface module stating that the positioning of the conveyor tail is done. A shear may be paused if loose coal needs to be removed by the conveyance system and a signal has not been received from the operator that the conveyor tail (see figure 1) is in position over the haulage system.

The shear task is complete when the cutting drum boom reaches the appropriate angle down near the floor of the mine. The choice of an appropriate angle is dependent on whether the coal seam is level or sloped fore-aft at that point. Due to the cylindrical shape of the cutting drum, the sump and shear operations leave a residue of coal, called the cusp, on the mine floor between the cutting drum and the gathering pan. The cusp is removed by raising the stabilization jack and tramming in reverse with the cutting drum on. This completes the first sump_shear_cusp cycle.

The next task is the approach_face command which in most respects is the same as the initial_approach_face command described above. During approach_face, the exact location of the face relative to the CM is now known which was not true during initial_approach_face. Therefore, after raising the cutting drum boom as before, the CM need only tram forward a specified distance after which contact with the face is guaranteed; no cutter current load monitoring is required. However, in a final system it would be useful to monitor cutter current load as well as CM position during the approach_face command (this would assure robust performance of each approach_face command). Another sump_shear_cusp cycle is executed exactly as before. At this point in the box cut task, a sequence of approach_face and sump_shear_cusp commands are executed until the total cut distance is reached. The box cut task is now complete.

2.2 TASK TREE

After developing the scenario we generate tasks whose relationship can be described in the form of a tree as in figure 2. In figure 2, each task name is prefixed by a two letter mnemonic where each mnemonic corresponds to the controller module within which it is grouped.

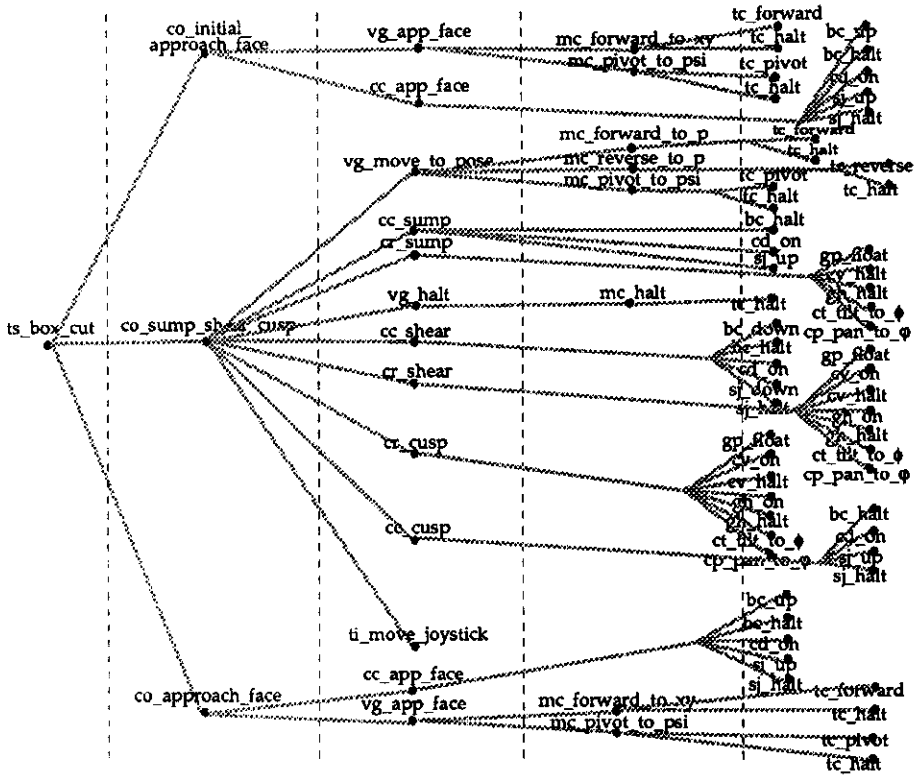
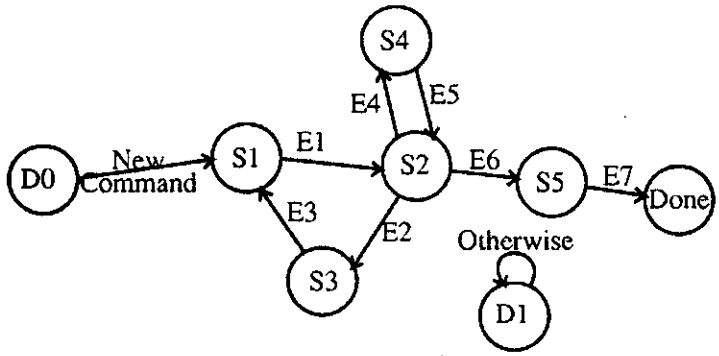


figure 2: Task tree for a box cut of a continuous mining machine

2.3 STATE MACHINE EXAMPLE: A SUMP_SHEAR_CUSP COMMAND

RCS adopts a finite state machine model where each command has finitely many internal states and responds to input signals by performing a transition to a new state. While performing a box cut, a fundamental operation of the CM is the sump_shear_cusp command for removing coal (as described in section 2.1 above). This command is decomposed into lower level commands and delivered to its three subordinate control modules within a decision state table. This decomposition can be seen in the task tree of figure 2. The task tree is limited in that it only reveals the set of tasks that are derived from each higher level task. The task tree neither reveals the conditions that trigger the execution of each subordinate task, nor does it specify the precise order in which each subordinate task is executed. Therefore, a decision structure of some sort is required beyond a simple task decomposition. The state graph and state table for this specific command are given in figure 3.



Conditions | Actions

Event	Current State	Next State	Jobs	Commands to subordinates: Coal removal (cr)	Vehicle guidance (vg)	Coal cutting (cc)
New Command	D0	S1	set pose for sump	cr_sump	vg_move to pose	cc_sump
E1:sump distance reached	S1	S2	none	cr_shear	vg_halt	cc_shear
E2:popped out of shear	S2	S3	none	cr_halt	vg_halt	cc_shear correct
E3: shear correct done	S3	S1	set pose for sump	cr_sump	vg_move to pose	cc_sump
E4:too much loose coal	S2	S4	none	cr_halt	vg_halt	cc_halt
E5:tail in position	S4	S2	coal removal	cr_shear	vg_halt	cc_shear
E6:shear complete	S2	S5	set pose for cusp	cr_cusp	vg_move to pose	cc_cusp
E7:cusp complete	S5	Done	none	cr_halt	vg_halt	cc_halt
Otherwise	D1	Same	NOP	NOP	NOP	NOP

Di => don't care condition Sn => n^{th} state En => n^{th} event

figure 3: A state graph and state table for a sump_shear_cusp task

2.4 DESIGN OF THE CONTROLLER HIERARCHY

Throughout the process of scenario development and task decomposition, tasks of the same level of abstraction are grouped into what are called controllers. At the lowest level, the controllers are matched with the appropriate actuators. At all

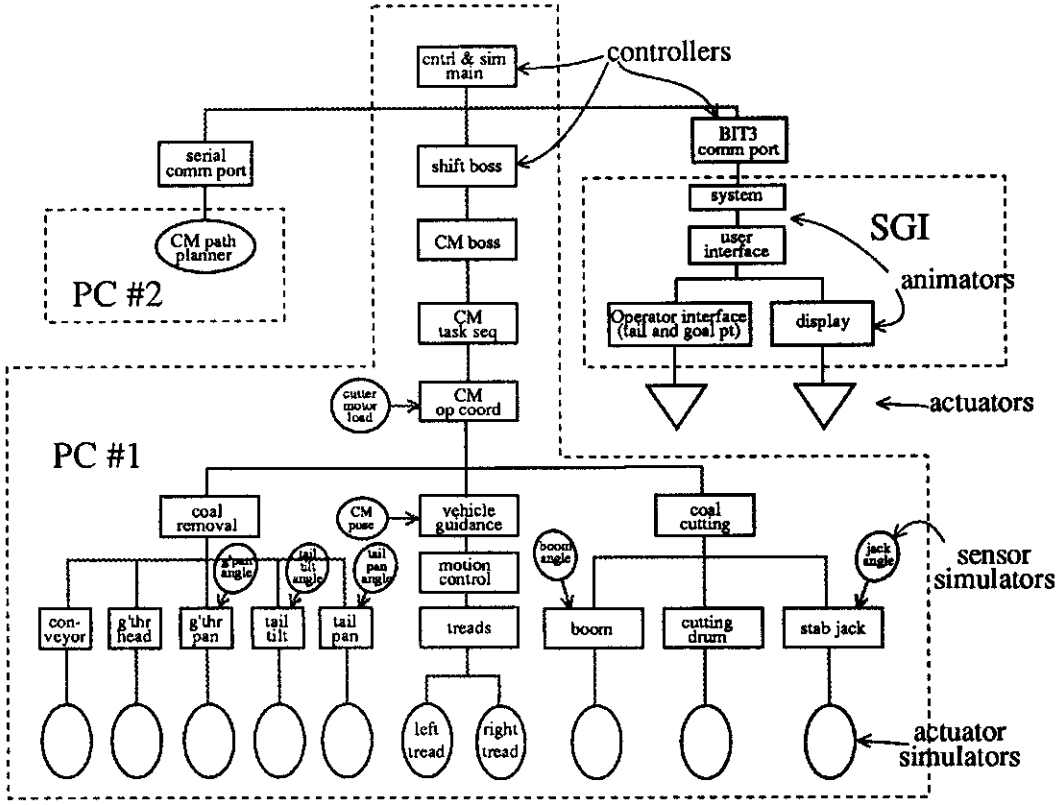


figure 4: Continuous mining machine control, simulation, and animation hierarchy

levels, sensors along with world model values and functions are matched to the appropriate controllers. The relationship between the task tree and controller hierarchy can be seen by comparing the task tree of figure 2 and the controller hierarchy of figure 4. The two letter mnemonic prefix on each task in figure 2 describes the controller in figure 4 to which that task belongs.

Controllers are the key 'objects' in the RCS methodology. Tasks are encapsulated within each controller. The communications interface to its supervisor and subordinate(s) is of the same form for all controller 'objects'. Each controller is responsible to synchronize and coordinate the tasks of all its subordinates. Tasks at similar levels of abstraction are grouped into controllers. For example, we grouped all cutting related tasks at a certain level into a single controller (coal cutting) and, similarly, grouped tasks relating to coal removal into the coal removal controller. This design effort produces the controller hierarchy shown in figure 4. As the number of tasks grows in a particular controller, it can be split up into two or more separate controllers.

In this application, we have developed and used software templates containing that which is common to all controllers. Using a controller template the system designer can simply enter the details unique to that controller while not having to repeat the standard parts of the code. The existence of these templates is a modest step towards the automation of RCS design.

One characteristic of all controllers is the processing pattern [Quintero 92]. All controller code is divided into pre-processing, decision-processing, and post-processing which are executed in order. Pre-processing includes reading commands and status, debug-related processing, executing sensory and world model functions, and on-line planning. Decision processing includes plan specific sensory processing, world model processing, planning, and decision making (as in figure 3). Finally, post-processing includes writing

commands and status, more debug-related processing, and sensory and world model processing.

2.5 PROCESSING AND SYSTEM ASPECTS

2.5.1 CYCLIC PROCESSING WITHOUT INTERRUPTS

Real-time control is often accomplished through the use of automatic scheduling of prioritized processes using interrupts (pre-emption) and time slicing. In contrast, RCS, as herein described, uses manual scheduling of cyclically executing processes. RCS processes can be executed serially or in parallel. Parallel execution removes the requirement to maintain synchrony and so there are concomitant latencies. Latencies are a necessary sacrifice, since we want the freedom to move processes to other CPUs with a minimum of system software changes. Time slicing is also done manually in RCS by requiring that each real-time process (e.g., controllers and simulators) be designed to execute to completion within a period of time such that real-time system response is assured.

Both interrupt driven methods and RCS can achieve real-time execution. However, the use of interrupts tends to complicate the code more than necessary, particularly as the number of real-time processes increases. Additionally, if one allows a real-time task to be interrupted, the state of the system or world may have changed in an unknown way after servicing the interrupt. One needs to handle a prohibitive amount of exceptions in order to determine what an interrupt would mean at any point in the control code. Since processor costs are comparatively low, we see little reason to employ pre-emptive scheduling which was created to optimize processor usage.

Manually scheduled, cyclically executed controllers help achieve determinism, allowing for an assured real-time response as long as the maximum execution times of controllers are known and there is never any looping within processes causing the rest of the system to wait. We might also describe this method of scheduling as static process scheduling, since the processes are executed according to a fixed schedule. Each process is designed to execute repeatedly, and on each invocation, it will execute to its conclusion. Each process must be executed at a specified frequency. If the resources of the system are insufficient to meet the required schedule, it can be discovered by the scheduler using the debugging tools provided.

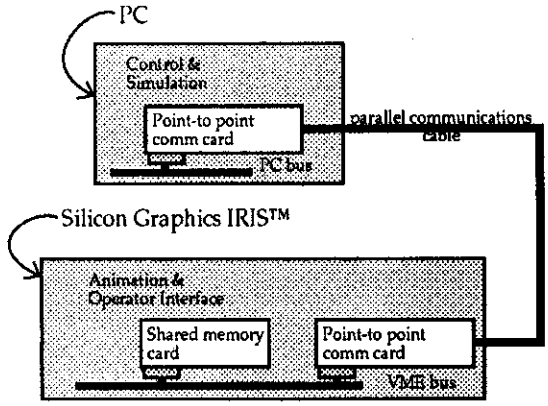


figure 5: Hardware tools

2.5.2 SIMULATION, ANIMATION, AND OPERATOR INTERFACE

Simulation, animation and operator interface are critical components in the development and operation of control systems. Simulation and animation are used in our implementation as a safe, cost-effective way to debug and refine task knowledge. All simulation (control, sensor, actuator, and environment) runs on a single PC (under DOS™) so that it can be made to run deterministically (without interruption). The animation code resides on a Silicon Graphics IRIS™[1] workstation under a non-deterministic operating system (as illustrated in figure 5). This is because the CM and its appendages need only move in a manner realistic to the human observer for animation to be effective.

Much of the animation code used in the CM control implementation is portable from application to application. Similar code has been written for other RCS applications following the RCS design philosophy [Huang 92]. The animation hierarchy is illustrated within figure 4. The existing

[1]Reference to specific products does not imply endorsement by NIST or the US BOM.

animation code is written in 'C' on the Silicon Graphics IRIS™ (SGI™) using 'GL' (an SGI™ specific 'C' library of predefined graphics functions).

A graphics interface is defined which allows the user to change the viewpoint and scene through mouse control. Also available in the animation code are several other operator interface capabilities, for example, 1) position control of the conveyor tail and 2) goal point entry for the movement of the machine in the mine (either for cutting or tramming in free space).

The following design tasks convert the 'generic' animation code into the CM implementation:

1) Draw the continuous mining machine, i.e., specify the coordinates of its body and all its appendages in three dimensional space.

2) Get the SGI™ to read the appendage position values from the PC via the common memory (as in figure 5) and write operator interface values to the same common memory.

3) Adjust the existing animation code to draw the mining machine on the screen cyclically based on both the raw values and the user specified viewpoint.

4) Integrate the operator interface as required. For the CM control implementation, we have control of the conveyor tail and the goal position for CM free space navigation and cutting.

5) Allow the user interface to give the user the ability to change the scene and viewpoint with the mouse or keyboard.

6) Incorporate the display of all common memory values

3. CONCLUSION

Control of the continuous mining machine involves the complex coordination and sequencing of operations. However, at the lowest level, the operations are very simple due to the bang-bang nature of the machine. As a result, the coal mining example further reinforces our belief that this type of problem is very easily handled by a hierarchical, rule-based approach. RCS, as described in this paper, is such an approach. In brief, the methodology consists of 1) a natural, hierarchical decomposition of tasks, 2) the natural grouping of tasks into controllers with generic processing patterns and generic interfaces, 3) the synchronization and coordination of tasks using state graphs, and 4) real-time performance through cyclic execution. The simplicity of the method is one of its major advantages. Mining automation engineers and researchers at the US BOM have warmly received this approach and profited from its use.

4. REFERENCES

[Albus 89] Albus, J. S., McCain, H.G., and Lumia, R., "NASA/NBS Standard Reference Model for Telerobotic Control System Architecture (NASREM)," NBS Technical Note 1235, National Institute of Standards and Technology, U.S. Department of Commerce, April 1989.

[Albus 91] Albus, J. S., "A Theory of Intelligent Machine Systems", Proc. of IEEE Intelligent Robots & Systems 1991 -- Intelligence for Mechanical Systems, November 1991.

[Huang 92] Huang, H. and Hira, R., "Applying the NIST Real-Time Control System Reference Model to Submarine Automation: A Maneuvering System Demonstration," NISTIR, to be published, National Institute of Standards and Technology, U.S. Department of Commerce.

[Quintero 92] Quintero, R. and Barbera, A. J., "A Real-time Control System Methodology for Developing Intelligent Control Systems," NISTIR, October 1992, National Institute of Standards and Technology, U.S. Department of Commerce.

[Schnakenberg 92] Schnakenberg, G.H. and Sammarco, J.J., "Overview of the U.S. Bureau of Mines computer-assisted mining research program," U.S. Bureau of Mines Open Industry Briefing, April, 1992.