Multi-Agent Diagnosis and Control of an Air Revitalization System for Life Support in Space[1]

Jane T. Malin and Jeffrey Kowing
NASA Johnson Space Center
2101 NASA Road 1, ER2
Houston, TX 77058-3696
281-483-2046 and 281-483-2059
malin@jsc.nasa.gov and kowing@jsc.nasa.gov

Debra Schreckenghost and Pete Bonasso
Metrica, Texas Robotics and Automation Center Laboratories
1012 Hercules
Houston, TX 77058
281-244-6134 and 281-483-2738
ghost@hypercon.com and r.p.bonasso@jsc.nasa.gov

Joseph Nieten and Jeffrey S. Graham
LinCom Corporation
1020 Bay Area Boulevard #200
Houston, TX 77058
281-461-2119 and 281-488-5700
jln@lincom-asg.com and jgraham@lincom-asg.com

Land D. Fleming
Hernandez Engineering, Inc.
17625 El Camino Real, Suite 200
Houston, TX 77058
281-483-2058
land.d.fleming1@jsc.nasa.gov

Matt MacMahon and Carroll Thronesbery
S&K Electronics
1016 Hercules
Houston, TX 77058
281-483-8793 and 281-244-5602
matt.macmahon@jsc.nasa.gov and carroll.thronesbery@jsc.nasa.gov

Abstract—An architecture of inter-operating agents has been developed to provide control and fault management for advanced life support systems in space. In this multi-agent architecture, cooperating autonomous software agents coordinate with human agents, to provide support in novel fault management situations. This architecture combines the Livingstone model-based mode identification and reconfiguration (MIR) system with elements of the 3T architecture for autonomous flexible command and control. The MIR software agent performs model-based state identification and fault diagnosis. MIR also identifies novel recovery configurations and the set of commands required to accomplish the recovery. The 3T procedural executive and the human operator use the diagnoses and recovery recommendations, and provide command sequencing. Human interface extensions have been developed to support human monitoring and control of both 3T and MIR data and activities. This architecture has been exercised for control and fault management of an oxygen production system for air revitalization in space. The software operates in a dynamic simulation testbed.

TABLE OF CONTENTS

1. INTRODUCTION

The recent Deep Space 1 Remote Agent (RA) experiment demonstrated autonomous fault management in a remote spacecraft [1]. Autonomous operations and fault management for remote space production plants and habitats will also be needed to support future human exploration. Autonomous agents will be relied on to handle the details of operations, carry out tasks and achieve mission goals. They will be available in-situ with the needed specialized expertise, and will be able to react quickly and flexibly to the information and data that is close at hand.

However, when degradations and faults occur during extended unattended operations, these systems are likely to need human help. Software agents in remote locations will initially manage problems autonomously to maintain system safety and availability, and then will interact with humans to facilitate development and execution of recoveries.

This paper describes a multi-agent architecture for fault management that includes a human agent, elements of the

three-tiered (3T) autonomous control agent [2] and the Livingstone mode identification and reconfiguration module (MIR) [3] from RA. Both RA and 3T use similar layered control strategies and declarative programming styles for implementing autonomous flexible command and control. Each architecture includes a goal-oriented deliberative planner and a task-oriented procedural executive that manages low-level control. RA has an additional model-based module (MIR) that is a specialist in system level fault detection, identification and recovery (FDIR). In this multi-agent architecture, MIR is a cooperating FDIR specialist, in contrast to RA, where MIR is a tightly integrated executive with planning capabilities.

MIR uses abstract models of system device modes and patterns of symptoms to diagnose failures and differentiate between failed parts and sensors. MIR can also identify new recovery configurations after a failure, with the commands required to achieve the configuration.

The goal of adjustable autonomy is to increase flexibility and effectiveness of operations while decreasing costs, by supporting less frequent and lower volume communication and less frequent and detailed interaction with humans. A major challenge is to go beyond operation by human supervision and override, so that both human and software agents can work together to flexibly manage unexpected problems [4]. Advanced adjustable autonomous agents will support human interventions at the middle (task) or high (goal) levels, and will cooperate with the human just like any other agent in the architecture.

One concept of adjustable autonomy extends manual override with autonomy that can be adjusted by the human agent. Instead of the "on-off switch" of override, there is a "knob to turn" to vary the scope or the level of autonomy. The focus is primarily on control execution responsibility: disabling parts of the autonomous system, or alternating execution between human and autonomous agents. This concept can be broadened to a multi-agent team player concept, where an autonomous system is one of many agents, trading both information and control [5]. This type of architecture will facilitate use of human knowledge, whether it is needed to sequence recovery actions, adjust goals and define partial recoveries or identify a complete reconfiguration sequence for consideration by a software agent.

2. Application Domain: Gas Processing

This architecture of inter-operating and cooperating agents is designed for autonomous control and fault management of advanced life support systems for space exploration. Our demonstration cases are based on autonomous control and fault management of an air revitalization system (ARS) for human life support in space exploration.

A primary goal in the design of Advanced Life Support for space systems is to maximize the recycling of resources from waste products. The intent of this design is to recover virtually all oxygen (O_2) from the carbon dioxide (CO_2) respired by the crew of a spacecraft or space station node. Three major and interrelated subsystems convert CO_2 to O_2:

- Variable Configuration Carbon Dioxide Removal (VCCR) System: CO_2 is removed from the crew cabin atmosphere by adsorption into a sorbant bed, then desorbed from the bed and stored in an accumulator for recovery of the O_2.
- Carbon Dioxide Reduction System (CRS): reacts CO_2 removed from the cabin atmosphere with hydrogen (H_2) to produce methane (CH_4), which is vented, and water, which is transferred to the Water Recovery System (WRS).
- Oxygen Generation System (OGS): Water from the WRS is reduced to O_2 and H_2 by electrolysis. The O_2 is returned to the crew cabin and the H_2 is transferred back to the CRS where it is reused for the CO_2 reduction process.

A schematic of the flows of water and gases among the three ARS subsystems is shown in Figure 1. Carbon and hydrogen are lost from the closed environment in the form of CH_4, which is vented to space.

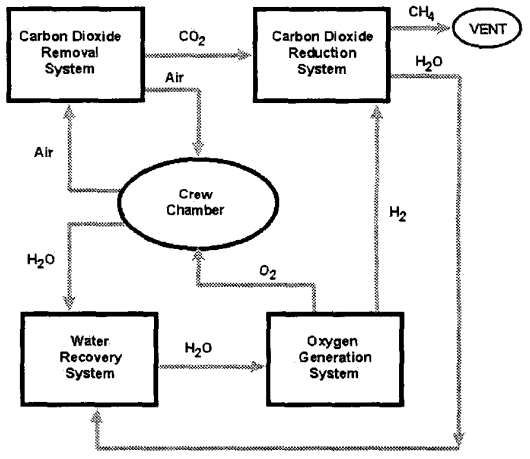


Figure 1 Schematic of gas flows in the Air Revitalization Subsystems

We used this application domain to demonstrate the capability of the multi-agent architecture to support adjustable autonomy for multi-agent fault management. The software operates in the Adjustable Autonomy Testbed, which provides a dynamic simulation of these subsystems as well as the autonomous agents that control them.

3. Cooperating Autonomy Architecture

The overall goal of the cooperating autonomy architecture (CAA) is to provide support for inter-operating autonomous

agents and humans, to control and manage faults in space exploration systems. The goal is to support adjustable autonomy and flexible use of specialist agents, and thereby provide robustness not achievable in other architectures. The architecture is intended to eventually accommodate specialist agents and to handle different varieties of autonomous agents, each with different but not necessarily exclusive roles. Heterogeneous software agents and humans can serve in the various roles in the CAA. Both kinds of agents (human and software) will work together in the implementations of our CAA. In such architectures, roles can reverse such that the software agents can be controllers and the humans can supervise and consult, or vice versa.

The following sections correspond to steps in defining the CAA. The first step is to define scenarios to represent the types of uses for the architecture. Next, agent roles are defined within the architecture. Then agent/tools are selected and adapted for use in the architecture. Finally, we define interoperability processes among the selected agents/tools.

Scenarios

To jointly handle diagnosis and recovery, cooperating agents have additional responsibilities beyond command and control. The following scenarios are intended to describe the environment in which the applications will be used. The first type of scenario concerns detecting and diagnosing faults. The second type of scenario concerns selection or design of responses to faults, including recoveries and work-arounds. The scenarios imply possible human involvement as a controller.

Our approach to adjustable autonomous control is based on the use of three different types of intelligent controllers. Procedural Executive (PEs) perform operational tasks by flexibly executing procedures consistent with the perceived situation. System Analyzers (SAs) perform intelligent system state identification and diagnosis by determining expected system values and comparing them with sensed system values. Planners develop designs, plans and schedules to achieve operations goals and fault recovery goals and to manage consumables and shared resources.

Scenarios for identification of faults:

- Both PE and SA detect a problem.
- SA detects a problem that PE did not detect.
- PE detects a problem that SA did not detect.

For each fault identification scenario, we could have one of the following recovery scenarios:

- PE already has enough information to take action.
- PE does not have enough information to take action and needs help from a planner to solve the problem.

Thus, we have identified six different scenarios for fault identification and recovery based on the three different types of intelligent controllers.

Agent Roles

This architecture supports use of different agents and modules in the same role at the same time. For clarity of mapping, the labels for intelligent controller types have been reused as roles where possible. The roles include a Procedural Executive (PE), a System Analyzer (SA), a specialized Configuration Planner (CP) and a Data Acquisition and Control Manager (DACM). Implementations of each role could include software agents, human agents, or both kinds of agents working together to achieve goals and accomplish tasks. The scope of capabilities of the agents may vary, from narrow specialist to a super-agent that can do it all. Some agents can be viewed as "consultants" that provide solutions when requested. How will agents take advantage of each others' specialties during operations, and how will they coordinate their activities, whether producing information or commanding actions?

Procedural Executive—PE intelligently sequences and executes operational procedures while monitoring to maintain safety and assure successful execution. Typically, PE has some implicit knowledge of mission priorities and constraints encapsulated in its procedures.

System Analyzer—SA monitors the controlled system and PE commands. SA predicts the expected system configuration (component states), based on its models of the system. SA diagnoses the causes of detected discrepancies, and notifies PE of changes in the state and health of system components.

Configuration Planner—CP uses models of the system to identify a command set for reconfiguring a system for recovery. CP develops reconfiguration plans that meet recovery goals and constraints. CP is a specialized planner that provides a command set to move the system from the current configuration to one that meets specified goals. Future work will address interaction between CP and more generic planning and scheduling roles.

Data Acquisition and Control Manager—The DACM role is taken directly from the Skill Manager in the 3T architecture. DACM is a low-level data acquisition and control subsystem that acts as an interface between PE and the hardware. DACM takes "commands" from PE and does the translations to command the controlled hardware components. Control algorithms can include PID or fuzzy control. DACM also gives PE and SA visibility into the monitored states of the controlled hardware components.

The Agents

CAA builds on current autonomous agent technology by altering the architecture, components and interfaces for effective teamwork among autonomous systems and human agents (HAs). Modules from 3T and RA are used for the CAA roles. 3T is discussed further in Section 4, and RA MIR capabilities are discussed further in Section 5.

The assignments of roles:

- DACM: 3T Skill Manager (SKM)
- PE: 3T Sequencer (SEQ)
- SA and CP: RA mode identification and reconfiguration (MIR)

An agent is a system that can act flexibly and independently to achieve goals and accomplish tasks in an environment. In multi-agent systems, these agents cooperate to achieve goals, and are thus not completely autonomous. By this definition, the agents in CAA are SEQ, MIR and HA. SKM provides data acquisition and control service to the agents.

MIR can independently deduce two types of information that can be volunteered or provided on request to SEQ: diagnoses of component states and designs for recovery. HA can independently deduce these two types of information, as well as others, based on additional information and data that may not be available to SEQ or MIR. SEQ and SKM can handle most routine problems by using pre-defined fault detection and safing. Some problems are detected and handled conventionally by SKM at the lowest level of control. Other problems manifest themselves as task failures that SEQ handles by selecting pre-defined alternative ways to perform the task.

More complex situations can be handled by MIR and HA cooperating with other agents to design novel recoveries. MIR or HA can determine a set of actions to achieve a novel recovery configuration. MIR needs a goal statement in order to plan a recovery and cannot prescribe the order in which commands should be executed. If automated recovery is not possible, HA can reduce or change the goal set to find a partial recovery, or can develop a novel recovery or partial recovery procedure. HA can also provide command sequencing. Recommended actions need to be mapped into existing SEQ tasks or low level commands, while still meeting the overall system goals and constraints. For example, an agent should be able to submit a prospective recovery, and other agents would "review" it and identify any problems with the recovery based on their knowledge of the system.

Interoperability Processes

Our architecture emphasizes interoperability between multiple autonomous agents and humans. We have defined a set of operational processes for fault management using the agents identified in the previous section. Interoperability processes have been designed to address each identified scenario.

These are the fault identification processes.

- Both SEQ and MIR detect a problem. In this case, MIR should notify SEQ of the detected a problem. This puts all the information in the "hands" of the SEQ.
- MIR detects an error that SEQ did not detect. In this case, SEQ has either missed a fault, or does not consider the fault to be important. Again, either way we need to get the fault information from MIR to SEQ. HA can monitor the discrepancy and decide how to handle it.
- SEQ detects a fault that MIR did not detect (whether there is a real fault or not). In this case there are not any MIR-SEQ interactions. Since the role of SEQ is to analyze all its data and act, and if MIR does not have anything to contribute, then there is no interaction required between MIR and SEQ. HA can monitor the discrepancy and decide how to handle it.

These are the fault recovery processes.

- SEQ already has enough information to take action to recover, and will proceed to take the appropriate action.
- SEQ does not have enough information to take action to recover, beyond safety actions. SEQ, the human agent or another planner can ask for help. Given a goal state for the system, MIR can provide a command set to the other agents that can be used to reconfigure the system to achieve that goal.

4. 3T Agent Architecture

The 3T architecture was developed to control autonomous robots [2]. This architecture has been applied successfully to the control of life support systems as well [6]. The architecture separates the general intelligent control problem into three interacting layers or tiers as follows:

- Planner: a deliberative layer, currently consisting of a hierarchical task net planner, which predicts requirements for control tasks needed to achieve system goals, allocates resources for those tasks, and directs and monitors their execution by the sequencing layer.
- Sequencer: a sequencing layer which decomposes tasks from the planner or a human agent into procedures (reactive action packages - RAPs) for carrying out tasks reactively, by enabling and disabling sets of skills in the skills layer.
- Skill Manager: a skills layer consisting of a dynamically reconfigurable network of closed-loop control and data acquisition modules that interface to the devices of the physical world.

Figure 2 illustrates the flow of control through the 3T architecture.

The level of control task abstraction and the time constant for control is different at each tier. Tasks become more specific as one moves down through the tiers. For example, low-level planning tasks correspond to high-level SEQ tasks, and low-level SEQ tasks correspond to skills. Time constants become smaller as one moves down through the tiers, typically by an order of magnitude. The time constants are 1-10 seconds for SEQ and 0.1-1 second for SKM.

The architecture is robust since each of its tiers is cognizant of failure and changes it activities accordingly. SKM reacts to changes in instrument readings that indicate control

commands have been successful or have failed. SEQ selects different control sequences, depending upon the conditions that hold at the time of execution. SEQ also can try alternative sequences to accomplish a task if the first sequence fails to complete. The planner re-plans when a planned task fails to complete successfully. At this point in the project, the planning tier has not been included in the CAA.

The CAA SEQ for ARS coordinates three separate skill sets for controlling the VCCR, CRS, and OGS. SEQ represents tasks to control each subsystem, as well as a set of tasks to coordinate these subsystems to accomplish air revitalization.

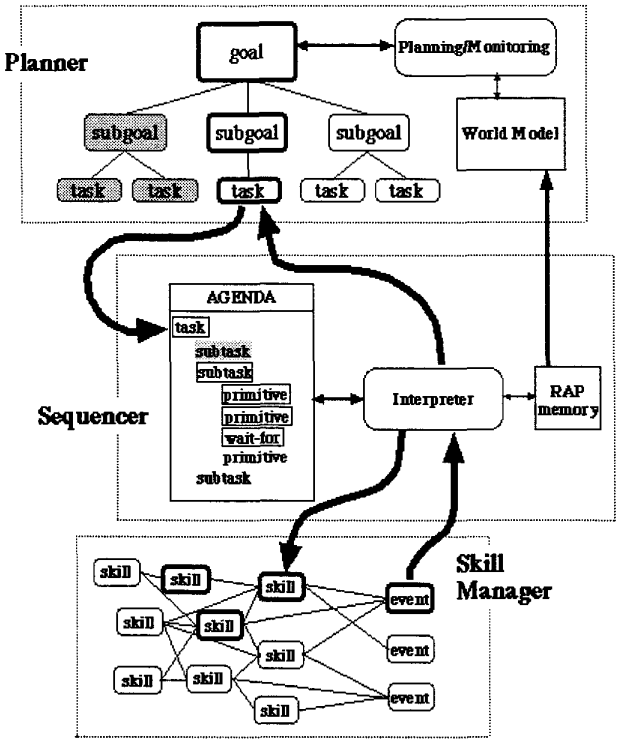


Figure 2 Three Tier control architecture [2]

5. MIR Diagnosis and Recovery Agent

The Livingstone MIR [3] agent is an expert specialist that provides model-based fault detection, identification and recovery (FDIR) capabilities. Mode Identification (MI) fulfills the CAA SA role, with a conflict resolution based strategy to track the current configuration of the system as a set of modes of the components. MIR can identify root causes that are far removed from the symptoms, and can reason in reactive time in some domains. The same model can be used to identify mode reconfigurations (MR) for fault recoveries, fulfilling the CAA CP role.

With the help of the MIR FDIR agent, SEQ and SKM can be focused on nominal control and safety. SEQ code does not need to include implicit backtracking from symptoms to unlikely failures, because failure identification should emerge from the explicit MIR models. Should multiple unlikely failures occur, MIR should be able to find a novel reconfiguration that achieves the goals of the system, by reasoning through the interactions of the components. With MIR, the CAA should be more flexible and adaptive than a control system that relies solely on human-encoded procedures, as it should be able to detect and reason about improbable combinations of problems that could not be enumerated beforehand.

MIR propositional models represent the behavior of the world as relationships between symbolic discrete values. For example, a pump may have an <on> mode where the model specifies that the pressure of the output port is equal to or higher than the pressure of the input port. Pressure is partitioned into one of four qualitative states: zero, low, medium, or high. The relationship of "pressure equal to or greater than" must be explicitly defined, and each additional qualified value added makes the calculation of this relationship combinatorially harder.

For example, the <on> mode of a pump used in the VCCR system model would be expressed as:

```
(on
  :type :ok-mode
  :model
  (and
   (negative (flow (input ?name)))
   (positive (flow (output ?name)))
   (lo-med-hi>= (pressure (output ?name)) (pressure
         (input ?name)))
   (hotter (temp (output ?name)) (temp (input ?name)))
   (gas-substances-equal (input ?name) (output ?name)
         ?substances))
  :transitions
  ((turn-off
    :when (off (cmd ?name))
    :next off
    :cost 1)
   (:otherwise :persist)))
```

The <on> mode is an "ok-mode" (nominal mode) rather than a "fault-mode". The model states that in this operational mode flow goes into the input and out of the output. The pressure of the output port is equal to or higher than the pressure of the input port, the temperature is higher at the output than the input, and all the levels of gases are transmitted through the component unaffected. Some variables, such as the direction of flow, are necessary to define operational state, while others such as temperature can define side effects that are helpful in tracking the state of the system. Finally, the model defines the nominal transitions from the <on> mode. In Livingstone, all nominal transitions must be triggered by a command. The only such transition is to the nominal <off> mode. Otherwise the pump should persist in the <on> mode. MIR continually tracks operational mode configuration for all components, and can

export this MI information automatically to other agents in the architecture.

MIR receives two types of information about system state, commands and monitors. Commands signal expected transitions between component modes and monitors indicate the actual component modes. Commands are discrete inputs from a controller to a device, e.g. <on> or <off>. Analog commands (e.g. the setpoint of a heater) are translated into discrete qualitative values. Monitors provide the symbolic state of sensed values in the system. MIR tracks the expected (commanded) state and the sensed state from the monitors. When the modes inferred from the commands and monitors are inconsistent, MIR makes a diagnosis by searching from the conflicts in the model to find the most likely consistent configuration of nominal and fault modes.

An agent may request a reconfiguration from MIR to achieve a goal, e.g., low carbon dioxide flow at ambient temperature into the crew cabin. The agent can use MIR to find a recovery from a fault-mode, to switch between nominal configurations, or to create a novel standby, safing, and fault-avoidance command set [7]. MIR finds the smallest set of actions or commands that would achieve the requested conditions by reconfiguring the system from the current state [3].

CAA MIR for ARS contains models of the VCCR and OGS subsystems. Care was taken to ensure the component models were general, so that they could be instantiated as part of new systems by providing new parameters and inter-component connections. The two systems share many components that are modeled identically, such as three way valves. Other components have slight variations. For example, the heat of the VCCR blower significantly affects the air temperature while that of the OGS heat exchanger blower does not.

The complex, continuous and time-dependent life support domain greatly complicates modeling. Whereas models of Deep Space One [7] contained flows, the flows were from a single source following at most one path down a bifurcating tree to the vacuum of space. In ARS, however, flow comes from multiple sources, is driven by various pumps, and takes a dynamic, reversible, merging, and splitting path as valves are opened and closed. This results in more complex models. Much of the modeling effort focused on achieving minimal models of ARS systems that provided sufficient coverage.

6. Cooperating Autonomy System & Testbed

The CAA software configuration used for demonstration consists of the following processes:

- Livingstone MIR (2 processes): models for the VCCR and the OGS
- Sequencer (SEQ - 1 process): RAPs for each subsystem, as well as a set of integrated RAPs that coordinate activities among subsystems
- Skill manager (SKM - 3 processes): a separate skill manager was developed for each subsystem of the ARS
- Simulation (CONFIG - 4 processes): each ARS subsystem has a separate simulation process; a fourth supervisory processes coordinates the subsystems
- Human interface (4 processes): each subsystem has a separate human interface for detailed device data; the fourth interface process displays information about control tasks and provides an overview of ARS system operations

IPC client-server software is used for communication among these processes [8]. All processes run on Unix/Linux platforms. SKM coordinates the communication of each skill set with the SKM via IPC. The skills in turn communicate via IPC with the CONFIG simulation that emulates the actual control hardware for a subsystem. SEQ activates skills corresponding to commands via SKM, and each SKM returns control events resulting from skill execution. SKMs publish information about device commands and state changes to MIR, for use in deducing device modes. SEQ subscribes to device mode information from MIR. SEQ, SKM and MIR export information to the human interface for display. SEQ and MIR also can execute commands sent from a human agent (HA) via the human interface.

We reused the message definitions developed previously for communication between SEQ and SKM, SEQ and HA [6] and SKM and HA. The following new message definitions were developed to facilitate interoperability between SEQ, MIR and HA.

SKM-Update Message—This message is published by SKM whenever a change is detected in the system being controlled. MIR subscribes to this message and uses this information to deduce operational states of system components.

MIR-Fault—This message is published by MIR whenever it detects a fault in the system being controlled. This message contains specific information about the component that has failed.

SEQ-RecoveryRequest—This message is published by SEQ or HA to request commands to achieve a system state. In the current implementation, this recovery request is generated by HA. The communicating agents must have a synchronized understanding of the current system state, since MIR bases recovery recommendations on its knowledge of current system state.

MIR-Recovery—This message is published by MIR upon request. This message contains information about the fault and the commands necessary to recover from the fault.

Figure 3 illustrates the demonstration configuration.

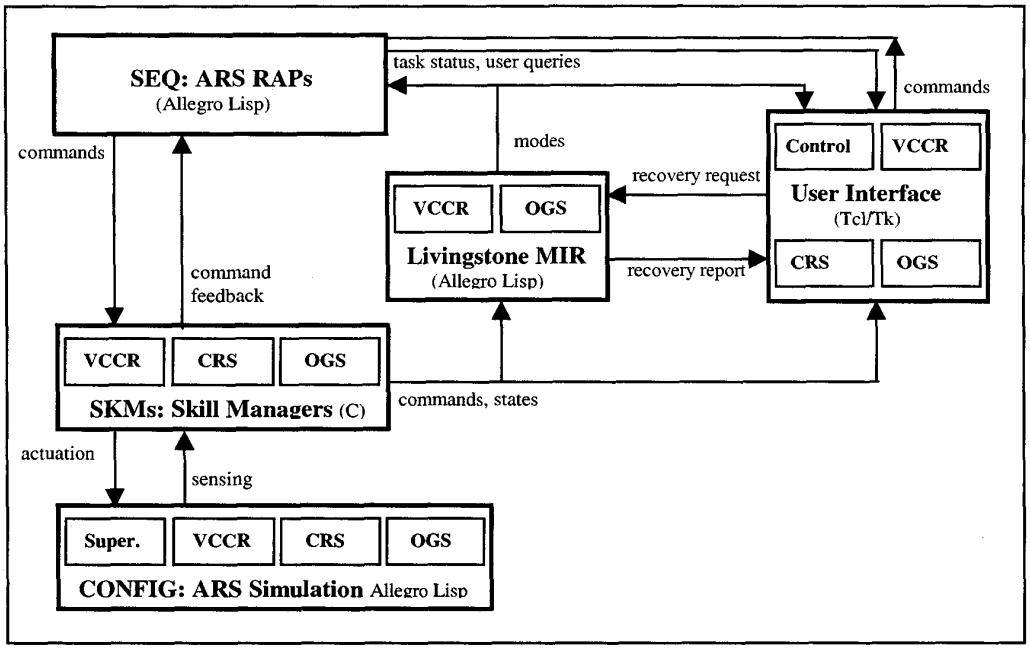


Figure 3 Software Configuration for ARS CAA

Modification of the 3T Software

Modifications were made to SEQ and SKM software to communicate with other agents of the architecture. One modification was required to incorporate failure mode information from MIR into SEQ processing. Another modification to SKM was required to provide device state information to MIR. A third modification was required to enable SEQ to request information from HA. All of these modifications were done in such a way as to be reusable across multiple applications. We also reused communication software developed for SEQ in other projects [6,9].

Diagnosis Information for SEQ—We modified the sequencer communication software to alter SEQ memory when mode information from MIR is received. We defined the RAPs memory fluent *(mir-fault ?subsystem ?component ?state)* where

?subsystem: subsystem (VCCR, CRS, OGS)
containing the affected component
?component: component that was assessed as failed
?state: the failure state

This fluent is asserted to SEQ memory when a fault-mode is received for a component, and the fluent is retracted from SEQ memory when an ok-mode is received for a component. For example, when MIR determines that VCCR heat exchanger has failed UNEXPECTEDLY-OFF, the following fluent is asserted to memory:

(mir-fault vccr hx-1 unexpectedly-off)

This fluent can be used as context to select a fault recovery method (as in the following example):

```
handle-VCCR-alerts
  method mir-faults 2
    context (and (get-VCCR-alerts vccrskm ?inlet-temp
                  ?bl-temp? hx-temp ?htr-temp-bed3
                  ?htr-temp-bed4 ?accum-press
                  ?cabin-co2  ?h2o-ps ?desorb-ps ?error)
                 (hc-valve-configuration vccrskm ?hc ?err)
                 (mir-fault vccr ?device ?fault)))
      task-net
        t1: put-out-warning vccrskm ?inlet-temp
            ?bl-temp ?hx-temp ?htr-temp-bed3
            ?htr-temp-bed4  ?accum-press  ?cabin-co2
            ?h2o-ps ?desorb-ps
        t2: handle-mir-faults
```

This fluent persists in SEQ memory until MIR assesses that the heat exchanger mode is an ok-mode (such as ON). When an ok-mode is received, the fluent is retracted from memory.

Information for MIR from SKM—MIR requires information for tracking the configuration of the ARS system so that it can detect failures. This information consists of the commanded and sensed states of the devices in the VCCR, CRS, and OGS systems. This information is available from SKM since it is this layer that reads and writes to the I/O channels that control the devices. Each SKM contains a "device" skill that interfaces to the specific I/O cards controlling the ARS devices. This device skill makes the commanded and sensed states of the devices available to other modules.

To generate the information required by MIR, a skill called

"pub2Liv" (publish to Livingstone) was added to the VCCR, CRS, and OGS skill sets. The pub2Liv skill receives the commanded and sensed values from the device skill. The pub2Liv skill, when necessary, converts the commanded and sensed values into qualitative form for MIR. For example, numeric temperature values would be converted into symbolic values such as "cool", "warm", and "hot." The pub2Liv skill publishes the data to MIR via IPC. The pub2Liv skill code is automatically generated from a specification file that describes each device controlled by SKM, including conversions from quantitative values to symbolic qualitative values. Information from this specification file is also used by the human interface and by MIR for message processing.

Information Requests to HA—SEQ communication software was modified to provide a capabilities for SEQ to request information from HA. Following the existing convention for SEQ to request information from SKM, we implemented two strategies for making such requests - one similar to a primitive event and one to a primitive query [10].

We developed the reusable primitive event AGENT-QUERY-RESPONSE. It takes three parameters – the requesting agent, the query, and the type of response. The format of the query can be one of the following

"{question?} {answer1 answer2}"
"{question?} {}"

When AGENT-QUERY-RESPONSE is executed, a message is sent to the human interface, but the execution of the RAPS event loop is not blocked while waiting for a result to return. This permits the RAPS event loop to continue processing while waiting for the response. A unique tag associated with the message is used to associate the response with the query. When the human interface receives this message, it displays the question with possible answers to HA. When a response is returned, the primitive event completes and the response is bound to a local variable in the active RAP in the same manner as other primitive events. If HA fails to give a response within the timeout period (set by default to 60 seconds), the event completes and returns a nil value.

An example using the primitive event AGENT-QUERY-RESPONSE is shown below:

```
turn-heat-exchanger-p (agent on-off timeout)
  succeed (heat-exchanger-state ?agent ?on-off))
  timeout ?timeout
  method normal
    context
      primitive (enable (tell-user-and-post "Turn ~a
                  heat exchanger ~a" '?agent '?on-off))
                (wait-for (agent-query-response ?agent
                    '(format nil "{Confirm ~a heat
                      exchanger is ~a} {yes no}"
                      '?agent '?on-off)
                    "symbol"
                    ?response)
                    :succeed(user-results ?response))
                (disable :above)
```

We also developed the function AGENT-PRIMQUERY-REQUEST that is used to embed queries to HA into memory queries. This function can be used in defining a primitive query. It takes four arguments - the human interface agent, the query, the type of response, and the timeout period. The query format is the same as that used in the primitive event AGENT-QUERY-RESPONSE.

When AGENT-PRIMQUERY-REQUEST is executed, a message is sent to the human interface and execution of the RAPS event loop is blocked while waiting for a result to return. When the human interface receives this message, it displays the request in the same manner as the display generated for a request made via primitive event. When HA responses, the primitive query completes and the response is bound locally as with any memory query response. If no response is made within the amount of time specified by the TIMEOUT parameter, the query returns nil.

Simulation Testbed

Models of the ARS subsystems are interfaced to the control software for testing and demonstration in interactive simulation. The demonstration uses CONFIG, a general-purpose discrete event simulation tool that has been adapted for interactive testing [11, 12].

Computer simulations of the physical system interact with the control software in lieu of the actual system. They can assist in uncovering and resolving a number of problems in design of intelligent autonomous software for controlling large complex systems. In such large systems, even the most knowledgeable engineers may not foresee every important contingency that should be reflected in the software requirements. A procedure to control one part of a system may fully satisfy the software requirements yet have unwanted effects on other parts of the system. Such ramifications may be difficult for the engineer or software developer to envision. Model-building itself has uncovered significant omissions in the knowledge that should be encoded in the software. Among the benefits are:

- simulation at speeds much faster than real time of the behavior of systems whose states change very gradually
- concurrent but independent software and hardware testing prior to integration
- testing software responses to system faults too dangerous or expensive to induce in the actual hardware

In CONFIG, models are constructed from object-oriented libraries of model parts. The user defines data structures and writes behavior descriptions for classes of devices. Models are constructed by connecting instances of the defined device classes to each other. Continuous processes are

simulated using Euler integration for computing values of rate-influenced variables.

Design parameters for the modeled system are stored in a "Global System Device" object associated with each model. Dependencies between the design parameters and the values of state variables in the devices can be expressed in the behavior description language. These stated dependencies are used to compute the values of the dependent component variables upon initializing the simulation. This capability, similar to some spreadsheet applications used in the sizing of hardware systems, has proven to be a necessity for modeling complex systems such as the ARS.

A linear circuit analysis module automatically constructs a computational network that updates the rates of flow and changes in potential (pressure) throughout the model as the magnitudes of resistances and motive forces change during simulation. For nonlinear flows, the linear representation provides a useful qualitative approximation that helps reveal how devices sharing paths of flow interact.

For large systems composed of subsystems with well-defined interfaces, the subsystems of interest can be modeled separately and simulations can be run concurrently on separate computer platforms. A supervisory process manages the synchronization of events in the different simulations and mediates data exchanges between them.

The CONFIG ARS Subsystem Models

Models for the three ARS subsystems were constructed and tested separately prior to integration with each other and with the multi-agent software. With the exception of the flow paths connecting simplified sources and sinks, each physical flow indicated by an arrow in Figure 1 corresponds to a data flow from one subsystem model to another during the distributed simulation. The crew chamber was represented as a limitless source of CO_2-laden air and a sink for the recycled air. The WRS was also represented as a limitless source of water in the OGS model and as a sink in the CRS model.

A screen capture of the CONFIG model of the CRS is shown in Figure 4. In this schematic view of the model, active flow paths are indicated by small triangles overlaying the connections between devices. The values of device variables in the model can be changed during simulations to induce failures for evaluation of the responses of the control software and other elements of the architecture.

Model Interfaces

An interface specification file records the mapping from the "transmitting" state variables in each model to "receiving" variables in the other models. Variable values are transmitted only when they change. The connections use specialized connector objects such as those represented by the icons labeled "H_2-OGS" and "CO_2-VCCR" in Figure 3.

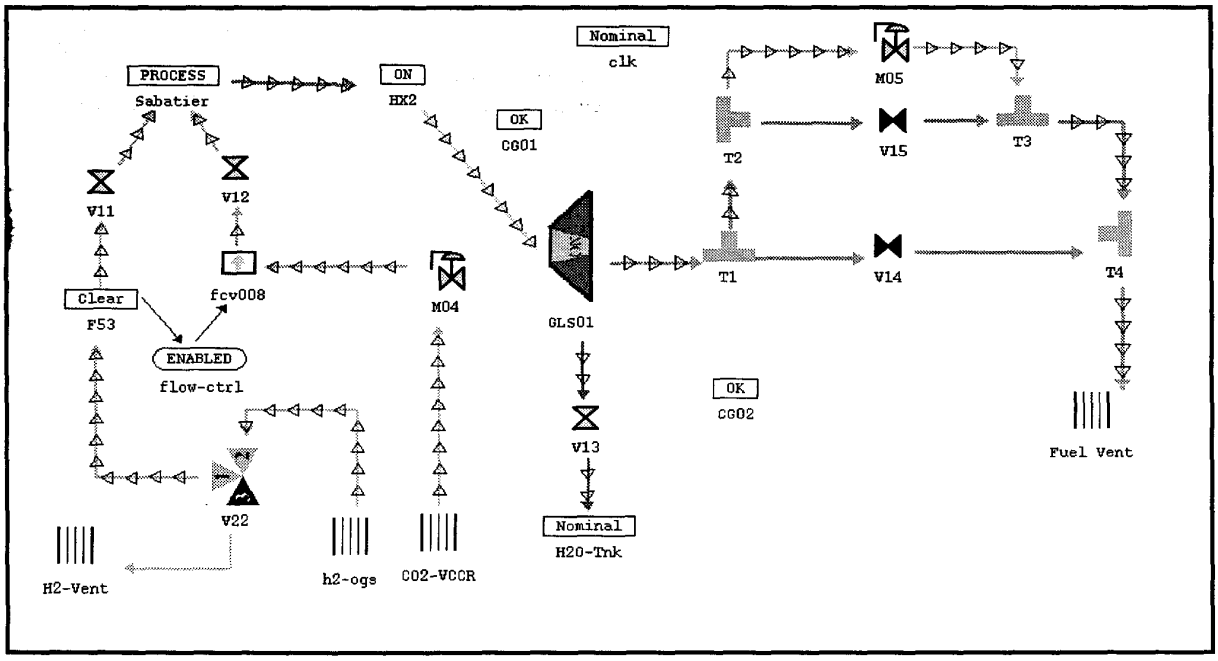


Figure 4 CONFIG CRS model

These objects receive data on the motive forces and resistances on the connecting paths of flow to and from the other models. The flows of H_2 and CO_2 from the concurrently executing OGS and VCCR simulations to the CRS are computed from state information received by corresponding connector objects in the CRS model. The interfaces between each CONFIG model and the corresponding SKM are also defined in a specification file. Unlike the inter-model interface, however, the communications with the control software are not entirely data driven. The values of model variables corresponding to sensor inputs are transmitted to the control software whenever a value changes or a specified maximum period of time has elapsed since the last data transmission. In discrete event simulation, time does not normally advance at a rate proportional to real time. While this is one of its benefits, allowances must be made for the real-time nature of the control software. For this reason, the control software interface specification includes statements indicating the conditions under which a real-time approximation mode is to be imposed so that the control software is given sufficient time to respond to critical system states.

7. Support for the Human Agent

In multi-agent diagnosis and control, the human agent (HA) performs a variety of new tasks, to exchange information and trade control:

- Supervising autonomous fault management tasks
- Interacting with the control architecture at the task level, to adjust control autonomy
- Monitoring the health of the multi-agent control system

To support supervising autonomous fault management, an overview display shows the integrated operation of ARS subsystems and detailed displays provide data from each subsystem. To adjust control autonomy, HA can perform manual tasks selected by the autonomous control software, or select and activate tasks for other agents to perform autonomously. HA can monitor the health of the distributed, multi-agent control system by using a data communication monitor that issues alarms when data rates are unusual. We describe each of these capabilities in this section.

Supervising Autonomous Fault Management

The displays integrate information from multiple agents in the distributed system. An overview of ARS operation summarizes the current state of control operations and provides a history of high-level-control actions. Schematic views of the physical configuration and gas flows are overlaid with sensor data, commands from the control software, and failure modes identified by MIR.

Overview of Operations—Two types of information summarize the current state of ARS control operations: an assessment of the operating mode for each subsystem from SEQ, and a summary of failures detected by MIR. The operating mode implies a physical configuration of the subsystem and the failures identify components of the subsystem that are anomalous. For example, when the VCCR is in HALF-CYCLE-1, the system is configured to desorb CO_2 from bed 3 and to adsorb CO_2 into bed 4. In Figure 5, the VCCR subsystem is operating in a steady state configuration (half-cycle-1 air_save) with a failed heat exchanger (HX1).

The top portion of the summary display shows the operating mode and phase of operation for each of the 3 ARS systems.

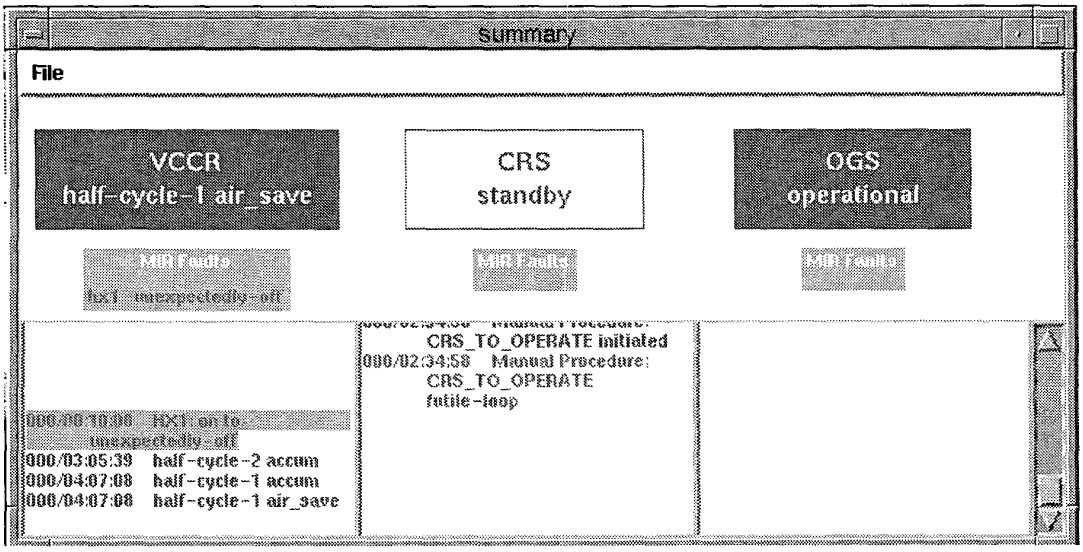


Figure 5 Overview of ARS Operations

When viewed together, operating mode information gives insight into the start up and shut down sequence for each system, and the dependencies between the operation of these systems. Phases are defined as follows:

- Operational: system is powered and operating in an equilibrium configuration (dark green background)
- In-transition: system is in a transient state as it moves toward steady state: operational or shutdown (white background, green text)
- Shutdown: system is not operating (gray background)

In Figure 5, the VCCR and OGS are operational and the CRS is in-transition. The system operating modes are determined by SEQ.

The middle portion of the display summarizes the MIR fault-modes that are active for each system. The name of the component and the fault mode are identified. The display of fault mode persists until MIR determines that the mode is an ok-mode. This display also provides access to a table showing current mode values for all components of a subsystem.

The bottom portion of the overview display contains a history of high-level control actions that indicate significant system changes. The history is organized chronologically into scrolling message lists for each system. Significant changes include a change in operating mode, the detection of a failed component, the repair of a failed component, and HA activation of control procedures. These changes can result from commands issued by HA or SEQ, or from modes detected by MIR. Background colors distinguish the different sources of information (SEQ–white, HA–yellow, MIR–blue).

State and Configuration of Devices—The graphic display of device data in a schematic diagram simultaneously provides a quick overview of a given system (VCCR, OGS, or CRS), and quick access to detailed device data. The visual forms were designed so that the display of device details would not detract from the integrated overview perception of system configuration. Each device icon in the schematic has four display regions described in Table 1.

Table 1. Display Regions in Device Icons

DATA	REGION	DESCRIPTION
command	inner border	command issued from skill manager
state	inner area	sensor value received by skill manager
status	outer perimeter	health of device, inferred by MIR
mode	L shaped center	Livingstone MIR mode inference

The regions associated with command, state, and mode can assume the colors of black (unknown state), gray (off state), or blue (on state). The outermost perimeter shows status (health) assessed by MIR. When MIR identifies an alarm mode, this border becomes red. Otherwise it is clear (no color). These colors are illustrated with the pump device in Figure 6. When the command value, the sensed value, and the MIR mode value all agree, the device is one solid color. When there is a disagreement among these sources of information, the device has multiple colors. As a consequence, under nominal conditions, HA is not overloaded with redundant information. Additional information appears as color change only as it becomes relevant (command differs from state, or sensed state differs from MIR assessment of state).

When devices are connected to one another in a schematic view, as illustrated in Figure 7, the blue "on" indicators

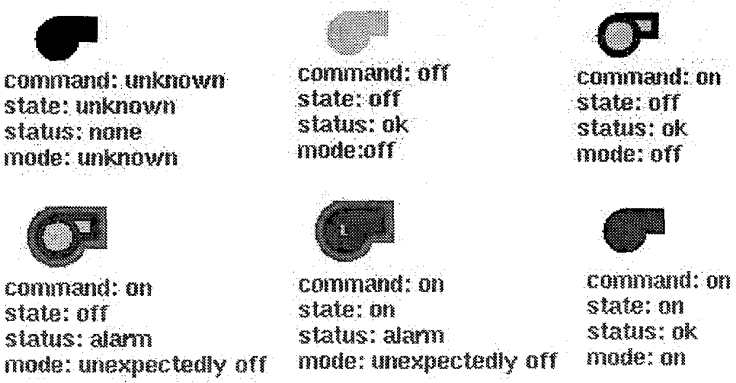


Figure 6 Device State and Status Displays

show the expected gas flow through the system. To achieve this configuration display, the individual devices (pumps, valves) display the commands, states, statuses, and modes from the skill manager and from MIR. A simple flow trace algorithm evaluates whether the "pipes" connecting these devices can have gases flowing through them, based on the device states and the sources and sinks of the gases.

Numerical values are shown for sensor devices, providing data such as temperature, pressure and gas analyzer values. Additional details are also available. From the "Data" pulldown menu, tables of device data from both the skill manager and MIR can be displayed graphically on the schematic. A left click on a device provides the device name.

A right click on a device

- provides the tabular display for a single device
- begins a real-time plot of the last few sensor values
- begins a textual readout "probe" of any device value
- displays the MIR mode transition diagram, described below.

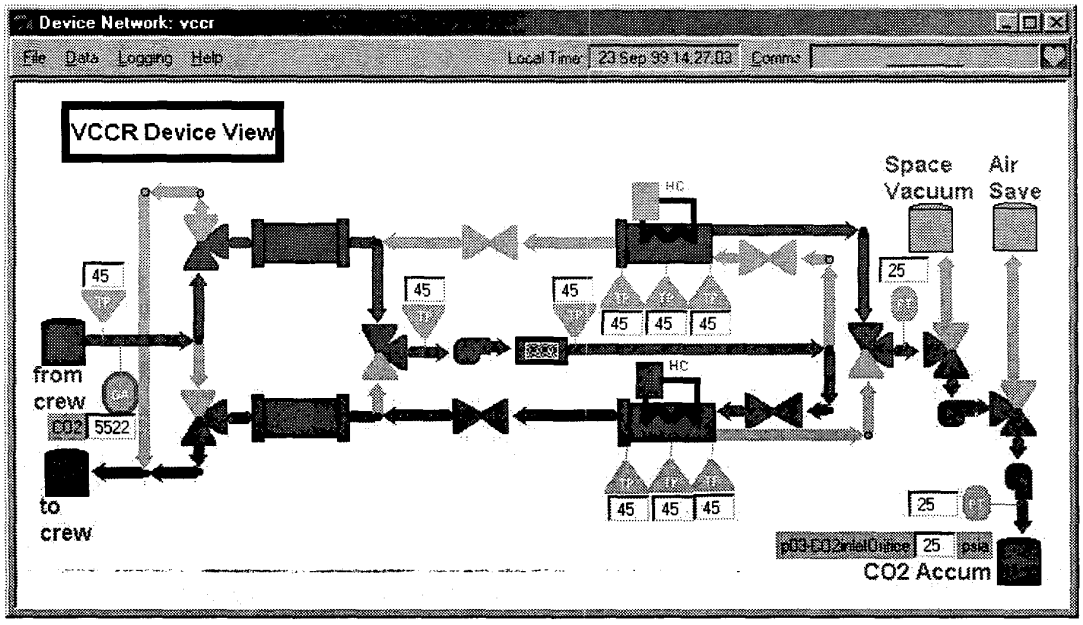


Figure 7 VCCR Device View

The human interface provides access to a mode diagram for each device represented in the physical configuration display, as shown in Figure 8. The MIR modes of the device are represented as nodes in a mode transition diagram showing all possible modes and mode transitions. The lines between the nodes represent mode transitions. Nodes shown in green represent ok-modes and nodes shown in red represent fault modes. The current MIR mode assessment is highlighted using a bright background color. The previous mode assessment is highlighted using a less intense background color of the same hue. Figure 8 shows an example of a mode diagram for a three-way valve that has transitioned from a 2-3 state (allowing flow through the upper portion of the valve) to a closed state (allowing no flow through the valve).

Additional descriptive information about each mode is accessible from a menu selection from the HELP pulldown:

- description of the mode
- model associated with the mode
- probability of the mode occurring

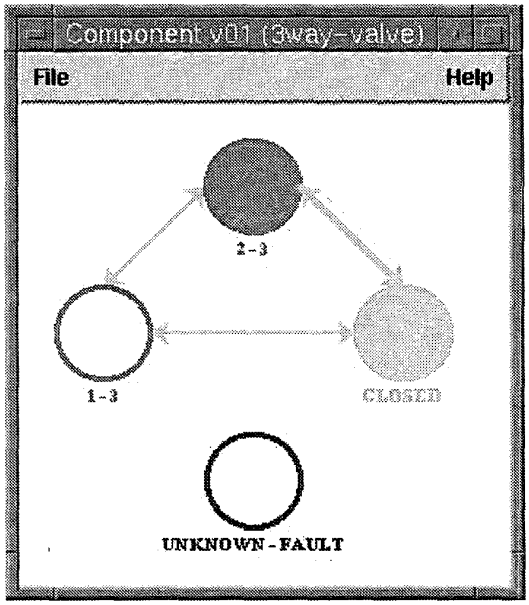


Figure 8 VCCR V01 Mode Transition Diagram

Adjusting Autonomy for Fault Management

Two approaches support the human agent in adjusting control autonomy for fault management. First, the human agent can execute autonomously selected tasks. Second, the human agent can select and activate tasks that are autonomously executed. Multi-step procedures can be defined and manually activated, and each step can be performed by any agent in the control architecture. We discuss each of these approaches in this section.

Autonomously Selected Manual Procedures—Manual procedures are selected autonomously by SEQ, and manual methods are executed when a manual level of autonomy has been specified. This approach uses new sequencer and human interface capabilities to support this type of autonomy adjustment [13]. SEQ can ask HA to perform a manual task, and HA can indicate when the task is complete [14]. Figure 9 shows an example where SEQ asks HA to confirm that the heat exchanger in the VCCR subsystem is turned on. Turning on the heat exchanger is done manually using control knobs on the device.

The SEQ request is an information query to HA, where the information requested is confirmation that a manual task has been completed. The request is posted to the USER NOTIFICATION window, with response options or a text entry box. HA completes the task and makes the appropriate response, which is returned to SEQ.

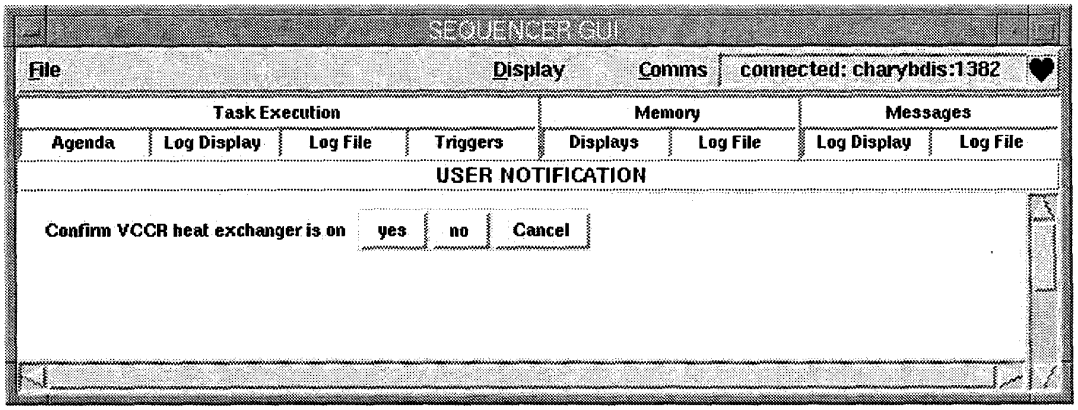


Figure 9 Sequencer Requests Information from User

HA-Activated Autonomous Procedures—An agent can select and manually activate multi-step procedures. Each step of these procedures results in the execution of a autonomous task. These task steps can be activated by any agent in the control architecture. For the ARS, the SEQ and the MIR agents performed the multi-step procedures. These procedures correspond to standard operating procedures (such as system startup and shutdown) and malfunction procedures (such as request recovery actions from MIR).

We defined a human interface for manually activated autonomous procedures. The control engineer specifies the procedures that define how HA is expected to interact with the control software. In the past, we have developed paper procedures for users that describe the typical interaction with the autonomous system, for example:

Startup ARS:

1. Initialize ARS by executing the function INIT-RAP-SYSTEM.
2. Startup the VCCR by installing the RAP START-UP-VCCR. The VCCR operating mode will change to HALF-CYCLE-1 AIR_SAVE.
3. After the VCCR is operational, start up the OGS by installing the RAP OGS-STARTUP. The OGS operating mode will change to OPERATIONAL.
4. When the OGS is operational, increase the OGS current setting to 45 amperes by installing the RAP (SET-CURRENT OGSSKM 45 30).
5. When the OGS current setting has been changed, start up the CRS in a STANDBY mode, by installing the RAP CRS-STARTUP. The CRS operating mode will change to STANDBY.
6. When the CRS is in STANDBY mode, change to operational by installing the RAP BEGIN-OPERATION. The CRS operating mode will change to OPERATIONAL.

A computer-based checklist interface captures much of the same information as the paper procedures, but provides the ability for HA to execute actions and monitor their effects from the checklist display.

Each procedure consists of one or more steps that are activated manually. Steps correspond to the execution of a capability in one of the active control processes. Each step includes additional information describing how to perform the procedure, identifying the conditions that should hold at the start of execution, and characterizing the effects that should be achieved by the step. These expectations are shown next to a display of actual data that characterizes the condition or effect. The execution status of an active step is indicated using both a text message in the step header and a

change in background color of this header (executing–blue, completed successfully–green, failed–red). An example of the ARS Startup procedure is shown in Figure 10.

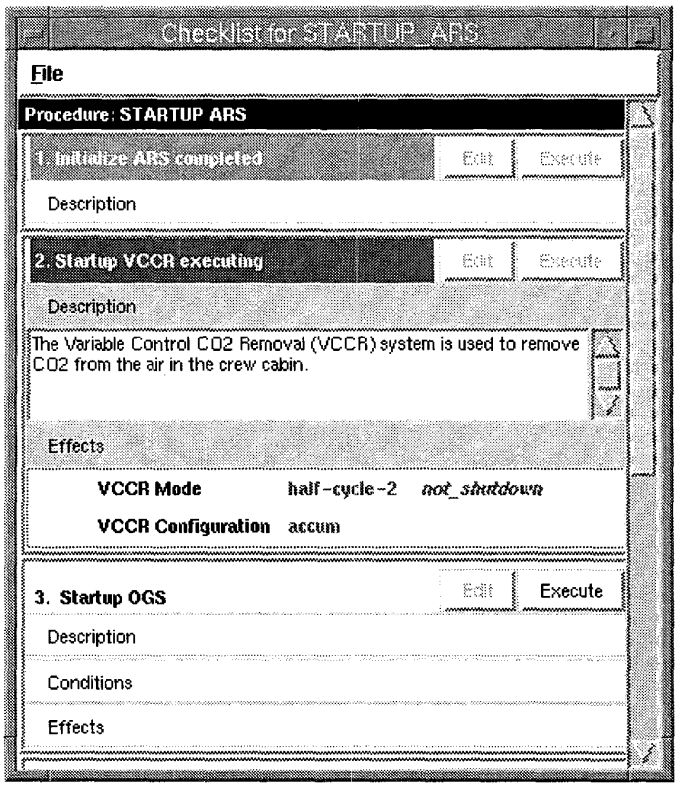


Figure 10 ARS Startup Procedure in Checklist Display

This checklist procedure interface provides an integrated display for controlling multiple software agents. HA can use the checklist to interact with SEQ to add a task, change a proposition in memory, or execute a function. The checklist can also be used to interact with MIR to make a recovery request. We plan to add the capability to issue high level commands to the planner and low-level commands to SKM. The capability to issue low-level SKM commands will be useful when SEQ control is unavailable because of problems or because of incomplete coverage in its defined sequences.

Monitoring Multi-agent Systems

Monitoring the general status of distributed processes is accomplished in two ways. One is by displaying data received from the distributed processes (discussed earlier) and the other is through the use of a "heartbeat" that signals the receipt of a new message from a remote process. The "heartbeat" signals the receipt of new messages and annunciates an alarm when too much time has elapsed since the last message. Figure 11 illustrates the dialog box that provides individual heartbeats for each agent.

For each agent, a separate heartbeat indicator, timer, and time-checker can be started. Each message from that agent restarts the timer and toggles the heartbeat display (swapping background and foreground colors). If there is a

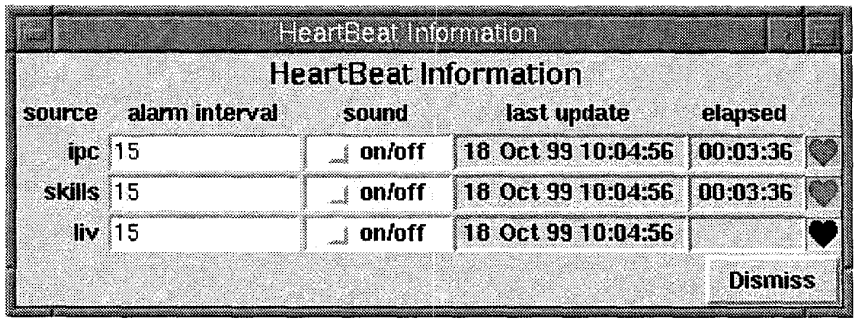


Figure 11 Heartbeat Dialog Box Shows Detail for Each Agent

timeout for a source, the heartbeat alarm is annunciated, changing the color of the heartbeat display from blue to red. HA can also set an option to sound an audible alarm. The timer display for each heartbeat tells how long it has been since the last message from that source.

The heartbeat information helps HA assess whether information from a source is stale. This is important when displays are designed to highlight unexpected values (sensed values that do not match inferred values from MIR). For example, consider the case of a valve that is commanded closed and inferred closed by MIR (based on temperatures upstream and downstream of the valve), but sensed as open. In this case, HA might suspect that the sensor on the valve is malfunctioning. However, if the human interface had lost communications with the SKM, the heartbeat would annunciate an alarm, and HA would instead suspect that the valve sensor had simply not changed before communications were lost. Instead of taking steps to accommodate or repair a valve with a bad sensor, HA would take more appropriate action to re-establish communications between the display and SKM.

We plan to further automate the heartbeat function, extending support for "management by exception" [15]. One extension would automate the process of re-establishing communications and restarting agents. Another extension would use a "ping" message to automatically check on an agent whose expected time-between-messages has been exceeded. This capability is especially useful for agents with irregular, change-only data communications.

8. Demonstration Cases

We have exercised and demonstrated the integration using two cases. In the fault diagnosis case, MIR agent identifies a fault that cannot otherwise be detected by SEQ. The other is a fault recovery case, where MIR handles a recovery SEQ cannot handle. MIR agent responds to a recovery request from HA by generating a report describing the suggested recovery actions. HA identifies and executes SEQ procedures that implement the suggested actions.

MIR Diagnosis with SEQ Recovery

A fault diagnosis case illustrates how SEQ can use fault identification information from MIR to recover from a system failure autonomously. In this case, SEQ uses MIR failure information to detect a control anomaly. Once the anomaly has been detected, SEQ executes a failure recovery method. HA is notified of fault management activities, but does not participate in them.

Our approach to autonomous fault recovery consists of the following steps:

1. MIR detects a fault mode and notifies HA and SEQ of the failure
2. SEQ uses the fault identification information to select a fault recovery method
3. After execution of the fault recovery method, MIR notifies HA and SEQ that the device is no longer faulty and nominal control resumes

Our case concerns the air blower in the VCCR. The blower moves air into the heat exchanger for cooling prior to CO_2 removal, and thus should be turned on whenever the VCCR is operating. In this scenario, the blower has shut off due to a power glitch some time after the VCCR has been started. SEQ does not detect this failure because once a physical device is turned on, it is assumed to stay on until commanded otherwise. The VCCR MIR software serves as a separate monitoring process that notifies SEQ and HA when the VCCR air blower goes off unexpectedly. SEQ matches this failure condition to an appropriate recovery method, which turns the blower back on automatically. Once the blower is on, MIR detects that the blower is no longer faulty and notifies HA and SEQ of its nominal operating state.

HA and MIR Handle a Failed Procedure

A fault recovery case illustrates how HA could use adjustable control autonomy to recover from a system failure. In this case, HA actively participates in fault recovery, because novel recovery actions should not be executed without considering the impact they have on other systems currently operating. This is a role for planners, and HA fulfills this role.

The approach to human-assisted fault recovery consists of the following steps:

1. MIR detects a fault mode and notifies HA and SEQ of the failure.
2. Since SEQ does not already have fault recovery methods to handle this failure, SEQ takes no recovery action.
3. Once HA sees the failure persist, HA asks MIR agent to suggest recovery actions (i.e., makes a recovery request)
4. MIR generates a report describing the suggested recovery actions. HA is notified of each action.
5. HA identifies procedures that can implement the suggested actions.

HA executes the recovery action via SEQ, and nominal control resumes.

The fault recovery case concerns the N_2 tank in the OGS. Nitrogen is used to purge the gas lines when the OGS is shut down. In this scenario, a slow leak in the N_2 tank has reduced the tank pressure. The OGS MIR software notifies HA and SEQ that the N_2 tank has a low pressure failure. SEQ initiates a safety shutdown of the OGS, but the nitrogen-purge step fails to complete and the OGS shutdown task fails. SEQ has no more actions to handle the problem. HA is notified that the nitrogen-purge step of the OGS shutdown failed. HA notices that the N_2 tank has a low-pressure anomaly, and postulates that the low tank pressure prevented the purge from completing. HA then asks MIR to

suggest a set of actions to recover from the low tank pressure. MIR determines that the tank pressure can be raised by injecting facility nitrogen into the tank. Since the use of facility nitrogen requires human approval, HA then initiates nitrogen injection into the tank. When the tank pressure is raised above a threshold, MIR notifies HA and SEQ that tank pressure is nominal. At this point, HA re-initiates the OGS shutdown, which completes successfully.

9. CONCLUSIONS

We have designed information exchange and control sharing in a multi-agent architecture for fault management that includes a human agent. We are exercising this architecture in cases that use the Livingstone model-based diagnosis and recovery agent as a specialist that assists a human and procedural executive in handling anomalous situations.

We have learned some limitations of the current Livingstone MIR engine and interface, which can lead to enhancements of that agent. Certain assumptions enable the speed and power of MIR at the expense of ease of expressiveness. For example, it is assumed that all nominal transitions are commanded and that a second command cannot be asserted while a first is being processed. Thus, the process of filling a buffer tank to a desired level can only be modeled with a pseudo-command asserted when the tank has reached the desired level. Future implementations of Livingstone can help solve this problem.

An enhancement of Livingstone was provided to support smooth MIR operations during startups and shutdowns, when many low level components change states at about the same time. Prior to the change, MIR generated false diagnoses when processing overlapping commands, indicating diagnosis over an inconsistent data set. The new software provides control over the suppression of monitors. Using this software, we suppress the processing of monitors during multi-command transient periods, delaying diagnosis until the system has reached steady state.

We have gained insight into the human role in fault management of systems controlled by autonomous software. As expected, MIS improves fault detection. It provides both the human and the autonomous controller with a capable new source of failure state information. Because failure states detected by Livingstone may not be consistent with sensed states, a means of disambiguating conflicting state assessments is needed. Initially, we propose that the human should perform this task. This approach takes advantage of human knowledge of underlying life support hardware and keeps the operator informed of state uncertainty.

The human role during fault recovery is more complex, particularly when there is no general planner in the control architecture. The operator may be involved in both the formulation of the recovery request and the execution of the recovery actions. Human expertise is needed in formulating the recovery request to ensure that all situation constraints have been considered and to evaluate trade-offs in goal selection. The operator can assist the execution of recovery in a couple of ways - determining when to perform the recovery action (i.e., how to interleave the recovery actions with other actions, ongoing and planned) and determining in what order to perform recovery actions. We expect that adding the planner to the architecture will automate much of the task of interleaving recovery actions with ongoing actions. Planned upgrades to Livingstone to add the ability to sequence recovery actions will automate action ordering. We expect, however, that the human will continue to be involved in verifying the generated recovery procedure.

Our next efforts will focus on integrating the 3T planner into the architecture. We also plan to expand the architecture to handle various autonomous agents, each with different but not necessarily exclusive roles. We plan to provide more support for review of proposed novel procedures.

We will continue to expand the life support systems models in our testbed, adding to the complexity of the system that will be autonomously controlled. The resulting test bed can be used to support the development of hardware and software systems. It could also be used to provide a simulator for training operations personnel.

REFERENCES

[1] Douglas E. Bernard, Gregory A. Dorais, Chuck Fry, Edward B. Gamble Jr., Bob Kanefsky, James Kurien, William Miller, Nicola Muscettola, P. Pandurang Nayak, Barney Pell, Kanna Rajan, Nicolas Rouquette, Benjamin D. Smith and Brain C. Williams, "Design of the Remote Agent Experiment," *1998 IEEE Aerospace Conference Proceedings*, March 1998.

[2] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David Miller and Marc Slack, "Experiences with an Architecture for Intelligent, Reactive Agents," *Journal of Experimental Theory of Artificial Intelligence* **9**, *237-256*, 1997.

[3] Brian C. Williams and Pandurang P. Nayak, "A Model-based Approach to Reactive Self-configuring Systems," *Proceedings of the 13th National Conference on Artificial Intelligence*, Cambridge, MA: AAAI Press, 971-978, 1996.

[4] Gregory A. Dorais, R. Peter Bonasso, David Kortenkamp, Barney Pell, and Debra Schreckenghost, "Adjustable Autonomy for Human-Centered Autonomous Systems on Mars," *1st International Mars Society Convention Proceedings*, August 1998.

[5] Emilie M. Roth, Jane T. Malin, and Debra L. Schreckenghost, "Paradigms for Intelligent Interface Design." In M. Helander, T. K. Landauer, and P. Prabhu (eds.), *Handbook of Human-Computer Interaction*, 2d Ed.,

Elsevier North-Holland, 1997.

[6] Debra Schreckenghost, Dan Ryan, Carroll Thronesbery, Pete Bonasso and Dan Poirot. "Intelligent control of life support systems for space habitats." *IAAI Proceedings,* July 1998.

[7] Barney Pell, Dan Bernard, Steven Chien, Erann Gat, Nicolla Muscettola, Pandurang Nayak, M. Wagner and Brian Williams. "An Autonomous Spacecraft Agent Prototype," *Autonomous Robotics*, **5**, 1-27, 1998.

[8] Reid Simmons and D. James. *Inter-Process communications: A reference Manual (for IPC Version 2.6).* Pittsburgh, PA: Carnegie-Mellon University, 1997.

[9] Debra Schreckenghost. Usable Autonomy: Adding Human Interaction to 3T. *Proceedings of the Adjustable Autonomy Workshop, IJCAI-99.* July 1999.

[10] R. James Firby. *The RAP Language Manual.* Chicago, IL: University of Chicago, March 1995.

[11] Jane T. Malin, Brian D. Basham, and Richard A. Harris, "Use of Qualitative Models in Discrete Event Simulation to Analyze Malfunctions in Processing Systems." In M. Mavrovouniotis, (ed.), *Artificial Intelligence in Process Engineering*, Academic Press, 37-79, 1990.

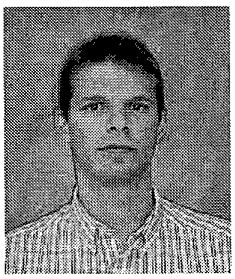
[12] Jane T. Malin, Land Fleming and Thomas R. Hatfield, "Interactive Simulation-Based Testing of Product Gas Transfer Integrated Monitoring and Control Software for the Lunar Mars Life Support Phase III Test," SAE Paper No. 981769. *28th International Conference on Environmental Systems*, July 1998.

[13] Pete Bonasso, David Kortenkamp and T. Whitney. "Using a Robot Control Architecture to Automate Shuttle Procedures." *Proceedings of 9th Conference on Innovative Applications of Artificial Intelligence*, Providence, R.I. 949-956, 1997.

[14] Debra Schreckenghost. "Checklists for Human-Robot Collaboration during Space Operations." *Proceedings of Human Factors and Ergonomics Society 43rd Annual Meeting*, September 1999.

[15] Carroll Thronesbery and Debra Schreckenghost. "Human Interaction Challenges for Intelligent Environmental Control Software." *Proceedings of 28th International Conference on Environmental Systems.* July 1998.

Jane T. Malin *is Technical Assistant and Computer Engineer in the Intelligent Systems Branch in the Automation, Robotics and Simulation Division in the Engineering Directorate at NASA Johnson Space Center, where she has led artificial intelligence (AI) research projects for 15 years. She manages the Adjustable Autonomy Testbed project where she has developed designs, design principles and technology for intelligent systems for collaboration with human operators. She has led development of the CONFIG simulation tool for evaluating intelligent software for operation of space systems. She has led research on intelligent user interface and expert systems for real-time monitoring and fault management of space systems. She received a Ph.D. in Experimental Psychology, focusing on cognitive psychology, from the University of Michigan in 1973.*



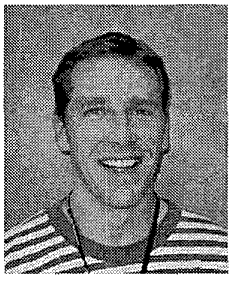
Jeff Kowing *is a Software Engineer in the Intelligent Systems Branch in the Automation, Robotics and Simulation Division at NASA Johnson Space Center. He has been involved in autonomous robotics, neural network, and expert system technologies. He has supported development and maintenance of the skill layer of the 3T architecture. He received his M. A. in Cognitive and Neural Systems at Boston University in 1995. He received a B. S. in Electrical Engineering from Wichita State University in 1988.*

Debra Schreckenghost *received a Masters Degree in Electrical Engineering at Rice University in 1982. After graduating, she worked at NASA Johnson Space Center as a Shuttle flight controller. For the last 10 years she has applied advanced software technology to aid humans in performing space operations. She was technical lead for the development of an adjustable autonomy control system to manage product gas transfer during the Lunar/Mars Life Support Technology Program Phase III test. Recent areas of research include architectures with adjustable levels of control autonomy, mixed initiative planning, intelligent user interfaces, and visualization of situated action.*

Pete Bonasso *is a senior staff consultant for AI & Robotics at Metrica's Texas Robotics and Automation Center Laboratories (TRACLAbs). Based at NASA's Johnson Space Center, he currently supports the Automation, Robotics and*

Simulation Division investigations of intelligent monitoring and control using layered architectures. He is the co-developer of the 3T Three Tiered Robot Control Architecture, and has applied that architecture to the control of robotic and life support machines. He received his B. S. in engineering from the U. S. Military Academy at West Point in 1968. After serving in the Army, he received Masters degrees in operation research and computer utilization from Stanford in 1974. Before joining the MITRE Corporation in 1979, he served as Assistant Professor of Computer Science at West Point, and as the software manager for the Army's Battery Computer System. At MITRE he was a principal investigator for AI research and the creator of MITRE's Autonomous Systems Laboratory. Research areas included rule-base systems for sensor data fusion, cooperating knowledge systems, and reactive control of autonomous robot systems. He was formerly the department head of the MITRE Washington AI Technical Center, overseeing studies and experiments in the application of AI techniques to military intelligence systems, strategic and tactical planning, and advanced battlefield simulations.

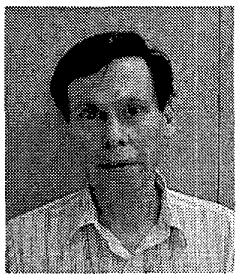
***Joe Nieten** is a Software Engineer & Researcher and an adjunct faculty member at the University of Houston --Clear Lake, where he teaches object oriented programming. For the last 10 years he has been doing research and development of intelligent systems. He has developed machine learning systems to assist in the verification of simulators, automated planners that can reassemble complex images broadcast from deep space missions, knowledge repository technologies to enable cataloging online content, and, most recently he has been working to define an autonomous agent architecture that can be used to provide autonomous control of production hardware. This last year, he was awarded a patent for his research on a collaborative software component architecture called "AgentNation". Joe has a Bachelors Degree in Mathematics from the University of South Florida.*



***Jeffrey S. Graham** obtained a B.S. in Computer Science from Louisiana State University in 1987. His graduate work specialized in robotics and AI. He has expertise in object oriented analysis and design, C++, and Unix. Mr. Graham has worked as a contractor in the NASA Johnson Space Center Automation, Robotics and Simulation Division since 1995. He is presently researching software architectures for autonomous agents in the "EVA Robotic Assistant" project.*

***Matt MacMahon** is a computer scientist specializing in AI and systems integration. He earned a B. S. in Symbolic Systems, focusing on AI, from Stanford University in 1993. He has developed intelligent systems for the Integrated Ground Demo of the Autonomous EVA Robotic Camera. He is continuing work on model-based fault detection, isolation, and recovery for the Adjustable Autonomy Testbed project, as well as a machine learning project. His research interests center on cognitive architecture design and implementation for smart agents in complex real world situations.*

***Carroll Thronesbery** has been developing intelligent systems and human-computer interaction software since receiving his Ph.D. in cognitive psychology in 1981. His projects at NASA Johnson Space Center include intelligent systems to help flight controllers monitor flight data, collaborative software to support anomaly response teams, and human interaction with intelligent life support controls software. His research interests include human-intelligent system interaction to support the control and monitoring of engineered systems (e.g., life support systems, Shuttle hardware sub-systems).*



***Land D. Fleming** is a Computer Systems Specialist working as a contractor in the NASA Johnson Space Center Automation, Robotics, and Simulation Division since 1990. He has been involved in both the development of computer simulation tools and their application to space systems. He received his M. S. degree in Computer Science from De Paul University, Chicago, Illinois in 1987.*

ACKNOWLEDGEMENTS

This work was performed under U.S. Government contracts. We wish to thank James Kurien for help in using, integrating and enhancing Livingstone MIR for this project.