AUTOMATION OF CLOSED ENVIRONMENTS IN SPACE FOR HUMAN COMFORT AND SAFETY

REPORT

FOR

ACADEMIC YEAR 1990-91

BY

Kansas State University
College of Engineering
Departments of Mechanical Engineering, Architectural Engineering,
Electrical and Computer Engineering, and Chemical Engineering
College of Education
Faculty Advisor: Dr. Allen C. Cogley

Submitted to

NASA/USRA
ADVANCED DESIGN PROGRAM

AUTOMATION OF CLOSED ENVIRONMENTS IN SPACE FOR HUMAN COMFORT AND SAFETY

REPORT

FOR

ACADEMIC YEAR 1990-91

BY

Kansas State University
College of Engineering
Departments of Mechanical Engineering, Architectural Engineering,
Electrical and Computer Engineering, and Chemical Engineering
College of Education
Faculty Advisor: Dr. Allen C. Cogley

Submitted to

NASA/USRA
ADVANCED DESIGN PROGRAM

ABSTRACT

This report presents the results of the second year of a three year design project on the automation of the Environmental Control and Life Support System (ECLSS) of the Space Station Freedom (SSF). The results are applicable to other space missions that require long duration space habitats. The report begins with a description of conceptual controls which are developed for the Water Recovery and Management (WRM) subassembly. Mathematical modeling of the Air Revitalization (AR) subassembly is presented next. The report concludes the work done by the Kansas State University NASA/USRA interdisciplinary student design team with a discussion of the expert system which was developed for the AR subassembly.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

1.1 PROJECT DESCRIPTION

The development of Environmental Control and Life Support Systems (ECLSS) for the Space Station Freedom, future colonization of the moon and Mars missions presents new challenges for present technologies. ECLS Systems which operate during long duration missions must be semi-autonomous to allow crew members environmental control without constant supervision. A control system for the ECLSS must address these issues as well as being reliable. The Kansas State University Advanced Design Team is in the process of researching and designing controls for the automation of the ECLS system for Space Station Freedom and beyond.

The ECLS system for Freedom is composed of six subsystems. The Temperature and Humidity Control (THC) subsystem maintains the cabin temperature and humidity at a comfortable level. The Atmosphere Control and Supply (ACS) subsystem insures proper cabin pressure and partial pressures of oxygen and nitrogen. To protect the space station from fire damage, the Fire Detection and Suppression (FDS) subsystem provides fire sensing alarms and extinguisher. The Waste Management (WM) subsystem compacts solid wastes for return to earth, and collects urine for water recovery. The Atmosphere Revitalization (AR) subsystem removes CO_2 and other dangerous contaminants from the air. The Water Recovery and Management (WRM) subsystem collects and filters condensate from the cabin to replenish potable water supplies, and processes urine and other waste waters to replenish hygiene water supplies.

These subsystems are not fully automated at this time. Furthermore, the control of these subsystems is not presently integrated; they are largely independent of one another. A fully integrated and automated ECLS system would increase astronaut's productivity and contribute to their safety and comfort.

1.2 THREE PHASE DESIGN PLAN

A three phase approach was implemented by the Kansas State University Advanced Design Team to design controls for the ECLSS. The first phase, completed within one year, researched the ECLSS as a whole system and then focused on the automation of the Atmosphere Revilatization (AR) subsystem.

During the second phase, the system control process was applied to the Atmosphere Revitalization subsystem. To aid in the development of automatic controls for each subsystem and the overall ECLSS, mathematical models have been developed for system simulation on a computer. Expert system control as well as conventional control methods are being tested on the models. Using the Atmosphere Revitalization subsystem control system as a "proof of concept", the other ECLSS subsystems will be automated.

Finally, during phase three, the control system of the six subsystems will be combined to form a control system for ECLSS. The expert system developed for the AR will be expanded to control the ECLSS as well as provide fault diagnosis and isolation to the astronauts.

The Kansas State University Design Team has completed phase one and two. Mathematical models for the CO_2 Removal Assembly, CO_2 Reduction Assembly and Oxygen Generation Assembly, as well as an expert system, have been developed for the AR. The mathematical models are written in the C-Language. At this time, the models function independently at the assembly level. The expert system, using a written shell program called CLIPS, monitors and controls the AR System assemblies in a hierarchical manner.

1.3 DESIGN TEAM DESCRIPTION

The Kansas State University Advanced Design Team is composed of engineering students from several disciplines, a student from general science and education, two graduate student assistants, and engineering faculty members. Architectural, chemical, computer, electrical, and mechanical engineering disciplines are represented by both students and faculty.

1.4 ORGANIZATION OF THE DOCUMENT

Chapter two of the report presents the work which was done by the conceptual controls group during the first semester of the design project. Conceptual controls were developed for the Water Recovery and Management System. These control schemes will be tested next year when the appropriate mathematical models are developed, and will aid in the development of an integrated ECLSS control scheme.

The mathematical models which were formulated for the Air Revitalization (AR) are presented in chapter three. These models are being used to test the conceptual control schemes which were developed in the first year of the project and also the expert system which was developed this year.

An expert system was created which monitors the AR system. Chapter four details the development and use of the expert system within the AR.

2.0 CONCEPTUAL CONTROLS

2.1 INTRODUCTION

This chapter outlines the progress made by the conceptual controls group toward the design of a control system for the Water Recovery and Management (WRM) System of the ECLSS. The WRM System was divided into four subsystems which are 1) Water Quality Monitor and Process Control Water Quality Monitor, 2) Urine Processor, 3) Hygiene Water Processor and 4) Potable Water Processor.

In designing the conceptual control scheme of the WRM System, several steps were taken before the final conceptual control scheme was developed. The first step was to research existing control schemes. Using this information, a tentative control scheme for each subsystem was developed. The final step involved critiqueing and revising each control scheme by group members and the faculty advisor until the best conceptual control scheme emerged.

Each section of this Chapter contains a description of the current control scheme, the proposed control scheme, the system monitoring scheme and a detailed sensor layout. A comparison between the current and proposed control schemes is also included.

2.2 WATER QUALITY MONITOR

2.2.1 WATER QUALITY MONITORING SYSTEMS DESCRIPTION

Two separate systems monitor the water quality from the various processors. The systems are called the Process Control Water Quality Monitor and the Water Quality Monitor which are discussed below in that order.

PROCESS CONTROL WATER QUALITY MONITOR

The Process Control Water Quality Monitor (PCWQM) is a system of monitors that continuously monitor the quality of product water from a particular processor--either a potable processor, a hygiene processor, or a urine processor. Each processor has a dedicated PCWQM through which all of its product water flows. The PCWQM has two distinct sections--the Continuous Section and the Intermittent Section (See Figure 2.2-1).

In the Continuous Section, product water from the particular processor flows into the PCWQM and on out through a series of four monitors. The monitors measure pH, Temperature, Iodine, and Conductivity levels every 15 seconds [5:1], [8:6]. Conductivity and pH are measured using standard conductivity and pH cells. Iodine is measured using absorption of UV/VIS light at 476 nanometers [5:1]. It is not known how the temperature is measured. The temperature reading is used to adjust the conductivity and pH readings.

In the Intermittent Section, Total Organic Carbon (TOC) is analyzed and pH and TOC are calibrated. Every 15 minutes, a 10 ml sample is diverted from the product water flow in the Continuous Section to be analyzed in the Intermittent Section [5:1-2], [8:6].

The TOC is measured using an ultraviolet reactor. The water sample is mixed with dilute phosphoric acid from the Acid Injection Module (AIM). The acid reacts with any inorganic carbon present. Any gases present in the water sample or resulting from the reaction of acid and inorganic carbon are vented. Oxygen is mixed with the water sample and ultraviolet light oxidizes the organic carbon to carbon dioxide. The infrared detector measures the amount of infrared light (4.28 micrometers) absorbed by the carbon dioxide and correlates this measurement to TOC content [5:1]. The TOC analyzer is calibrated every 24 hours by an unknown process using fluid from the Calibration Injection Module (CIM) [8:6].

The pH Calibration Monitor calibrates the pH monitor in the Continuous Section. The pH Calibration Monitor measures the pH of the 10 ml sample and compares its reading to that of the pH

monitor. If the readings are different, the controlling processor calibrates the pH monitor accordingly. The pH Calibration Monitor itself is calibrated every 24 hours using fluid of known pH from the Base Injection Module (BIM) [8:6].

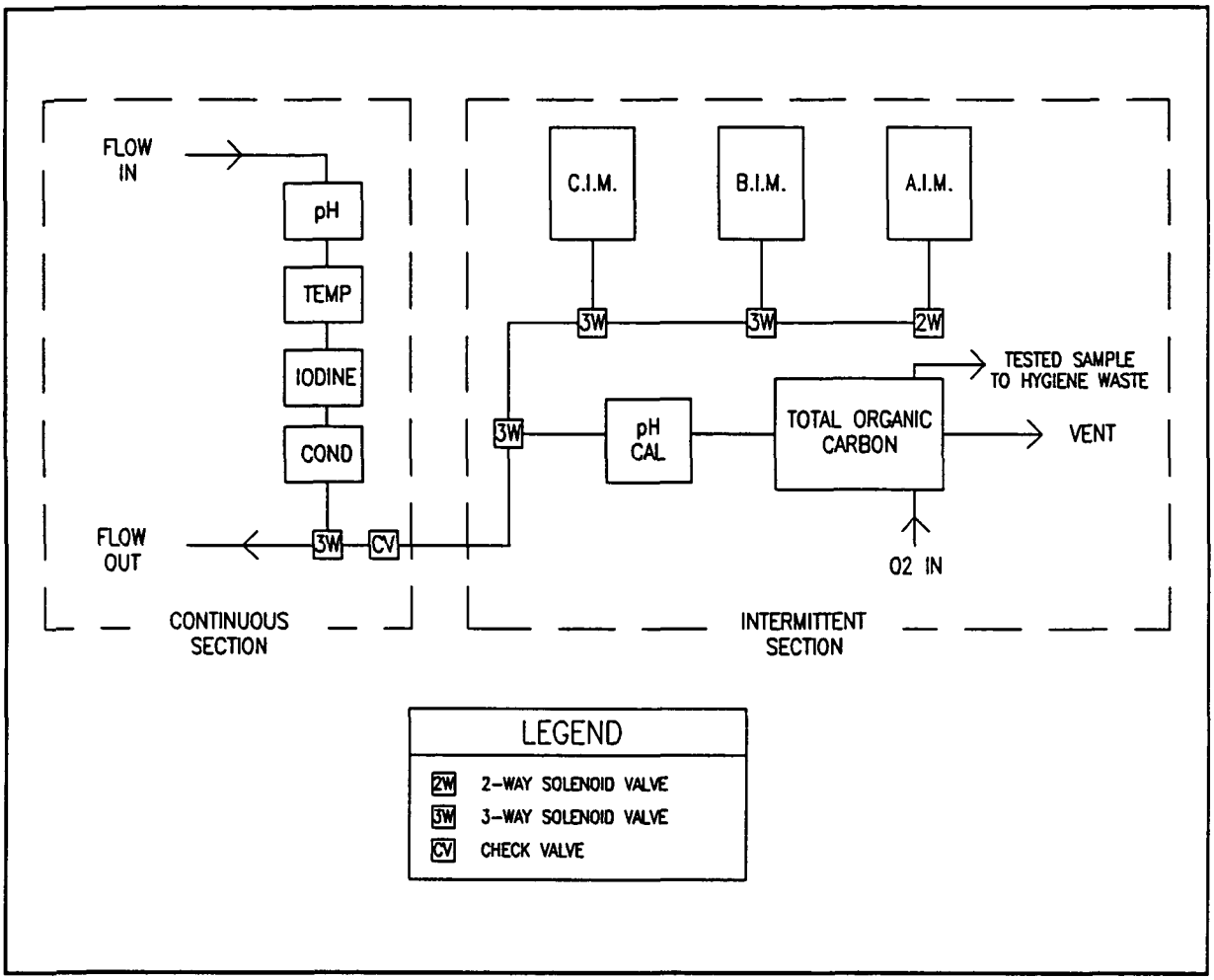

Figure 2.2-1 Current PCWQM
(adapted from [2:1])

The acid-tainted 10 ml sample and the used calibration fluid are sent to the hygiene waste water storage. The gasses from the gas/liquid separator are vented to the cabin. This minuscule amount of gas is eventually removed by the Trace Contaminant Control System.

Every 15 seconds, the PCWQM sends the pH, conductivity, raw iodine, and adjusted iodine readings to the Data Management System (DMS) [5:2], [8:6]. The raw iodine is the iodine measurement at the pH of the water. The adjusted iodine is the iodine measurement that would be present at a pH of 7 [5:2].

TOC measurements are sent to the DMS as they become available.

WATER QUALITY MONITOR

A Water Quality Monitor (WQM) analyzes each full tank of product water from a Potable Water Processor (PWP) and Hygiene Water Processor (HWP) pair. Only one product water tank from either the PWP or the HWP can be analyzed at any one time. The WQM also accepts manual samples when it is not already conducting an analysis. See Figure 2.2-2 for system layout.

The Data Management System (DMS) sends a signal to start the analysis process. The process begins by circulating water to the WQM from a selected product water tank. Circulation continues until a representative sample of the processed water can be extracted for analysis. This small sample is distributed to the various analyzers using an unknown internal sample distribution system. Each analyzer performs its own tests. As results from each analyzer become available, they are sent to the DMS. Some analyses are available almost instantaneously, while other analyses take up to 12 hours [7:2].

The various analyzers and the corresponding measured parameters for the WQM are listed in Table 2.2-1. In addition to these, the Gas Chromatograph/Mass Spectrometer can identify thousands of other compounds by sending the Mass Spectrometer scan to earth for further analyses.

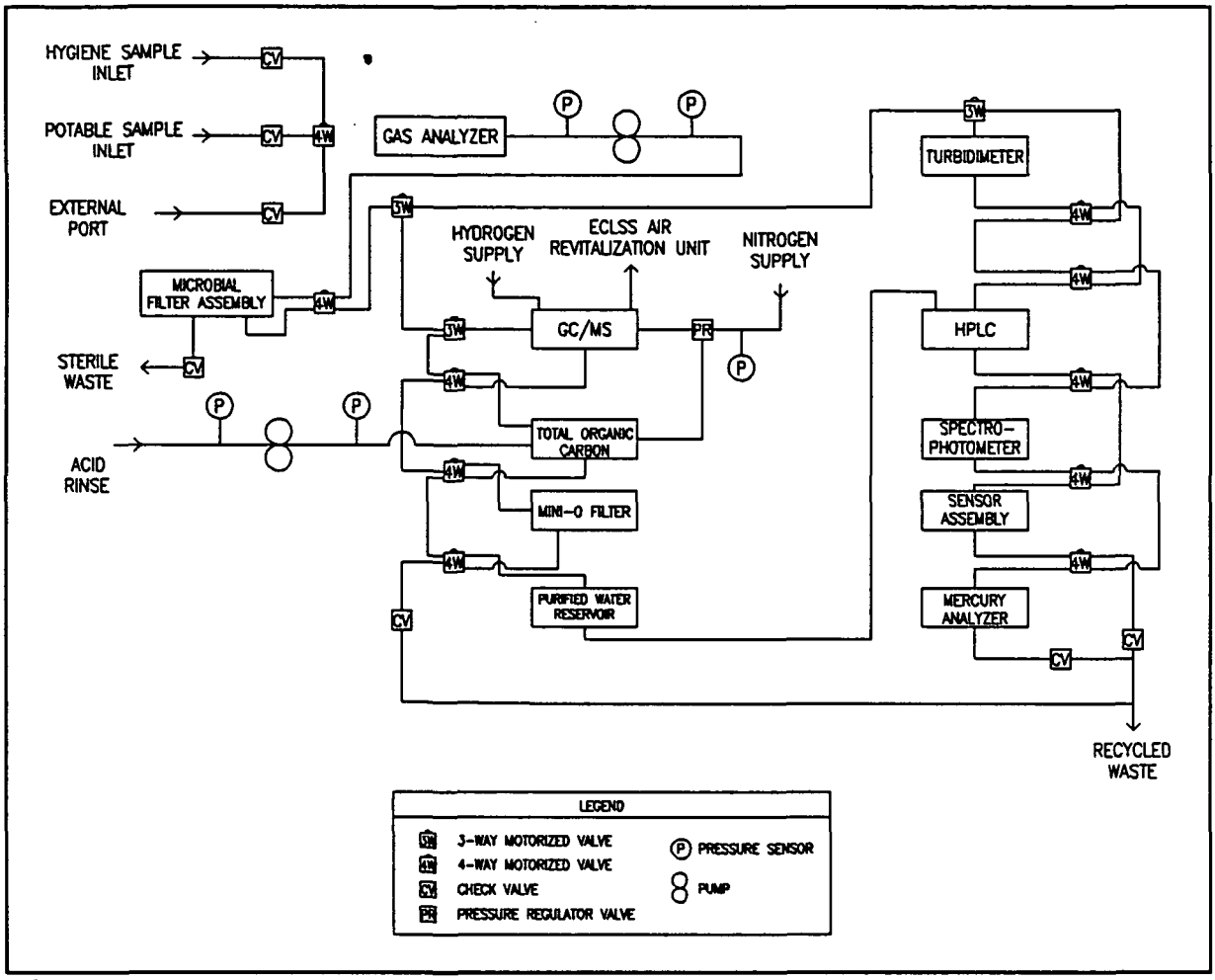

Figure 2.2-2 Current WQM
(adapted from [3:1])

Table 2.2-1 WQM Measured Parameters
(based on information from [7:1-2] and [3:1])

ANALYZER PARAMETER	BLOCK NAME (In Fig. 1.1-2)	MEASURED
UV-Visible Spectroscopy	Turbidimeter	Turbidity
pH Electrode	unknown	pH
Conductivity Cell	unknown	Conductivity
Spectrophotometer (UV-Vis. Spectroscopy)	Spectrophotometer	Color Iodine
Photometric	Gas Analyzer	Free Gas

continued

Table 2.2-1 WQM Measured Parameters

ANALYZER PARAMETER	BLOCK NAME	MEASURED
Gas Chromatograph/ Alcohol Mass Spectrometer	GC/MS	Aliphatic Carboxylic Acid Halogenated Hydrocarbons
Gold Film Analyzer	Mercury Analyzer	Mercury
UV Oxidation	Total Organic Carbon	TOC
High Performance Liquid Chromatograph	HPLC	Ammonium Barium Cadmium Calcium Copper Iron Lead Manganese Magnesium Nickel Potassium Silver Arsenic Chloride Chromium Cyanide Fluoride Iodide Nitrate Selenium Sulfate Sulfide Phenol Alcohol
Bact. Sensor Assembly (Bacteria Concentrator, Incubation Chambers, Conductivity Sensor, Chromogenic Limulus, Reagents & Reaction Chamber, & Colorimeter)	Sensor Assembly	Bacteria

2.2.2 WATER QUALITY MONITORING CONTROLS

Two separate systems--the Process Control Water Quality Monitor (PCQWM) and the Water Quality Monitor (WQM)--analyze the water quality of the product water from the potable, hygiene, and urine water processors. A PCWQM constantly monitors the quality of the continuous flow of product water from its associated potable, hygiene, or urine processor. A WQM thoroughly analyzes one sample at a time from each processor storage tank as the final test before the water is used. A WQM is responsible for monitoring a potable water processor and hygiene water processor pair. The PCWQM and the WQM will be discussed separately below.

PCWQM CONCEPTUAL CONTROLS

CURRENT PCWQM CONTROL SCHEME

The current PCWQM control scheme is a three operational mode system that has a sensor system consisting only of valve position indicators monitoring each valve.

The three operational modes are ON, STANDBY, and OFF [4:1]. In the ON mode, all systems are powered. In the STANDBY mode, only the electronic control system is powered. In the OFF mode, all electrical power to the PCWQM is off. The PCWQM operational mode is determined by the operational mode of its associated processor. For example, if the associated processor changes from the STANDBY mode to the ON mode, the processor sends a signal telling the PCWQM to do the same.

When the PCWQM is in the ON mode and product water is available, the PCWQM monitors the product water. If the monitored values are within the normal range, the results are sent to the Data Management System (DMS). If the analyzed value in a particular monitor is out-of-range, the calibration of the monitor is checked. If the calibration is acceptable, the results are sent to the DMS. If the calibration is unacceptable the astronauts are notified.

Valve position sensors provide feedback to determine if the valves are functioning properly. Readings from the sensors are monitored by the DMS.

PROPOSED PCWQM CONTROL SCHEME

Several features where added to the initial control scheme to allow for greater fault detection and isolation (Fig. 2.2-3). The features added are:

- calibrating analyzers that are out of calibration before notifying the astronauts.
- adding seven sensors.
- adding a manual shut-off option to the solenoid valves on the Calibration Injection Module (CIM), the Base Injection Module (BIM), and the Acid Injection Module (AIM).

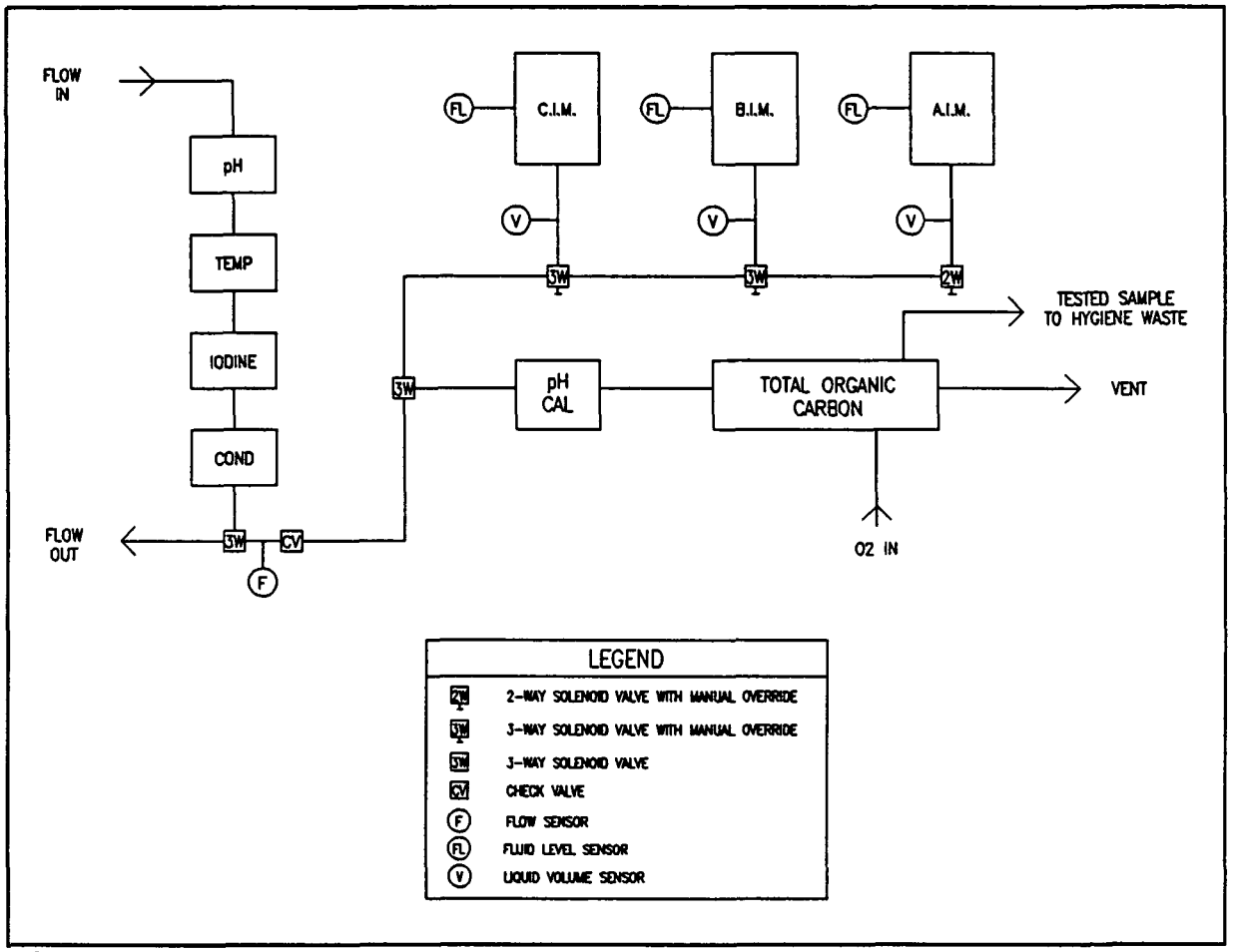

Figure 2.2-3 Proposed PCWQM
(adapted from [2:1])

PCWQM MONITORING SCHEME

System components are listed below along with a description of the monitoring scheme for each component.

Monitors

The pH and Total Organic Carbon (TOC) monitors are calibrated in the TOC section of the PCWQM. The iodine, conductivity, and temperature monitors do not need calibration.

Valves

Position indicators are used to determine if a valve is in the correct position. Whenever power is not available to the valve, the valve moves to a fail-safe position.

Check Valves

Flow sensors are placed after these valves to detect any leaks.

CIM, BIM and AIM

Fluid level sensors are used to indicate if a fluid tank is almost empty. Liquid volume sensors are used to ensure that the proper amount of fluid is drawn from the tank.

COMPARISON OF PCWQM CONTROL SCHEMES

The proposed control scheme has several advantages. It has more feedback because of the additional sensors. This increases knowledge that the system is working properly. The automatic calibration of out-of-range analyzers saves the astronauts from constantly monitoring and adjusting them. Finally, the valves with a manual override for the CIM, BIM, and AIM can be used to prevent damage to the PCWQM.

The disadvantages of the proposed system are the increased cost and complexity, the small increases in power consumption, and the increase in associated microprocessor size.

WQM CONCEPTUAL CONTROLS

CURRENT WQM CONTROL SCHEME

The current WQM control scheme is a three operational mode system with five sensors, in addition to valve position sensors, to monitor the process.

The three operational modes are ON, STANDBY, and OFF [6:1]. In the ON mode, all subsystems and analyzers are powered. In the STANDBY mode, only the ion pump, embedded controller, and related electronics are powered. In the OFF mode, all electrical power to the WQM is off. The normal WQM operational mode is the STANDBY mode, but the WQM changes to the ON mode when a sample becomes available. The OFF mode is used for maintenance of the WQM.

When the WQM is testing a sample (ON mode), each particular test result is sent to the DMS as it becomes available, as long as the

result is in the normal range. If it is not in the normal range, the calibration status of the particular analyzer is determined. If calibration is acceptable, the test results are sent to the DMS. If the calibration is unacceptable, the astronauts are notified.

Sensors provide feedback to determine if various system components are functioning properly. Readings from the sensors are monitored by the DMS.

PROPOSED WQM CONTROL SCHEME

This scheme modifies the current control scheme by:

- calibrating analyzers that are out of calibration before notifying the astronauts.
- adding ten sensors.
- stirring the particular processor storage tank before a sample is sent to the WQM.
- discontinuing any remaining WQM tests if one test is out of range.

See Figure 2.2-4 for the proposed WQM system layout.

WQM MONITORING SCHEME

System components are listed below along with a description of the monitoring scheme for each component.

Analyzers

Each analyzer block has its own calibration mechanism within the block.

Valves

Position indicators are used to determine if a valve is in the correct position. Whenever power is not available to the valve, the valve moves to a fail-safe position.

Check Valves

Flow sensors are placed after these valves to detect any leaks.

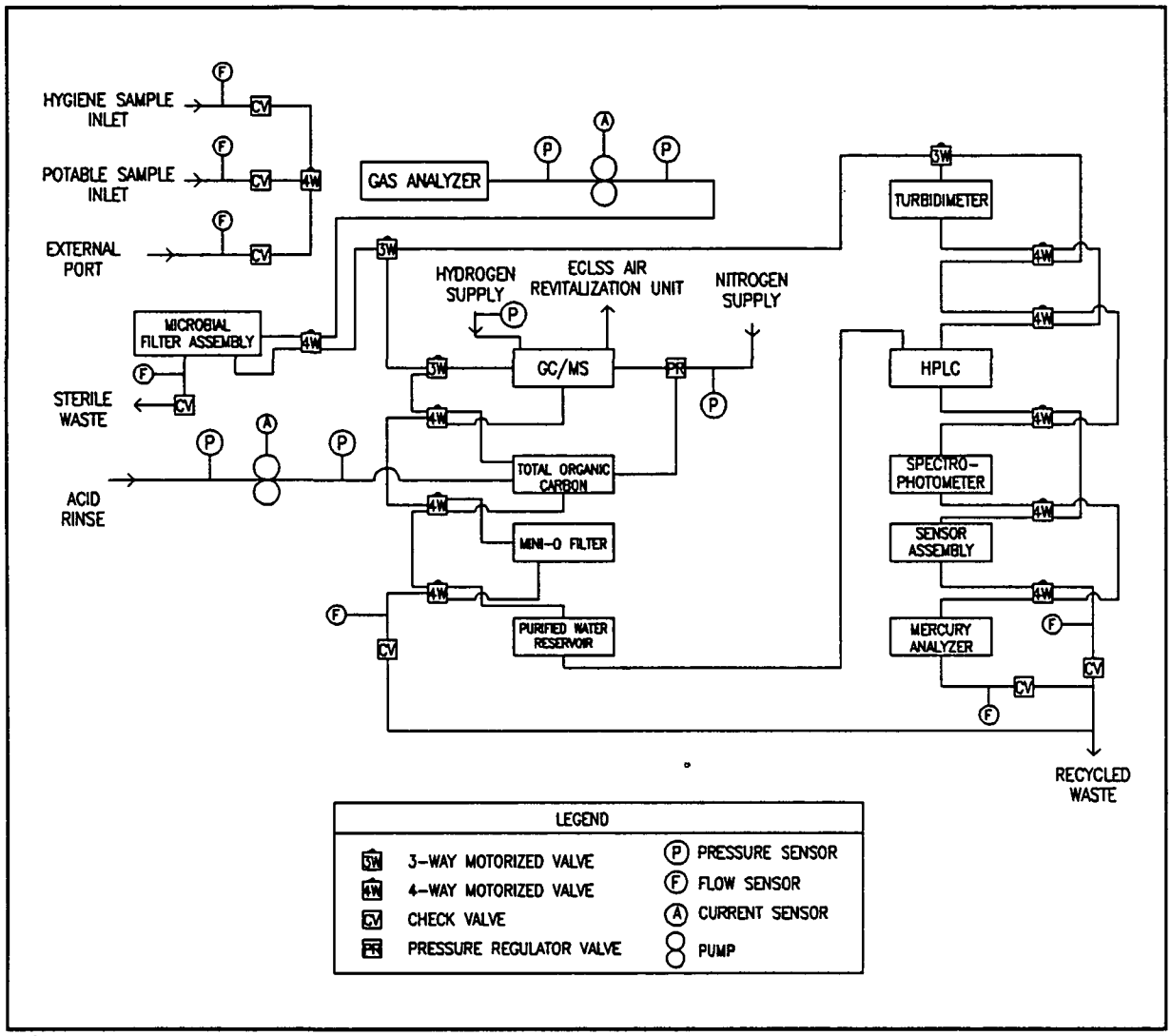

Figure 2.2-4 Proposed WQM

Pumps

Pressure sensors are placed directly before and after each pump to make sure the pump has the proper pressure differential across it.

Also, current sensors are placed on each pump to determine if the pump motor is drawing the proper amount of current. This will help determine if a pipe is plugged or if a pump fin is broken.

Pressure Regulator

A pressure sensor is placed after the regulator to determine if the pressure is being regulated properly.

COMPARISON OF WQM CONTROL SCHEMES

The proposed control scheme has several advantages. It has more feedback because of the additional sensors. This increases certainty that the system is working properly. The automatic calibration of an analyzer before notifying the astronauts saves the astronauts' valuable time by not calling them unless absolutely necessary. Stirring the processor storage tanks before sending a sample to the WQM ensures that a representative sample is analyzed. Stopping the remaining WQM tests if one test is out-of-range saves power.

The disadvantages of the proposed system are the increased cost and complexity, the increased power consumption for some of the changes, and the increase in associated microprocessor size.

2.3 URINE PROCESSOR

2.3.1 URINE PROCESSOR ASSEMBLY DESCRIPTION

The main objective of the Urine Processor Assembly (UPA) is to recover water from the urine and flush water from the urinal.[2:1] The UPA works in conjunction with the hygiene water processor by sending product water to be further processed and with the potable water processor by accepting brine to be processed.

Two urine processors will be located on the Space Station Freedom. A schematic of the UPA is shown in Figure 2.3-1.[4:1 of 1] One will be in the laboratory module and another in the habitat module. Each UPA can accept urine and waste water partially pretreated with Oxone (manufactured by Dupont) in the urinal on a 24 hour basis.[2:1] To further inhibit microbial growth and ammonia formation, a small amount of concentrated sulfuric acid will be injected by the acid injector pump into each flush from the urinal as it proceeds to the pretreated urine collection tank.[2:1]

The UPA units must be in a preheated state before a batch process is initiated. Electric heaters will operate for a sufficient period of time before each batch process, to achieve the necessary temperature for processing.

Processing will begin when the pretreated storage tank is 60-85% full depending on whether the other processor is running. [3:Fig I] Only one processor can run at a time because of power constraints. A UPA will also begin processing if it has been more than 48 hours since the last batch process. Pretreated urine should not be stored longer than 48 hours.[3:Fig I]

The pretreated urine is drawn from the storage tanks through a regenerative heat exchanger which preheats the urine and flush water. This mixture is then sent through a filter to trap any large particles in the waste water. The heated waste water is pumped into the Thermoelectric Integrated Membrane Evaporative System (TIMES). It first enters the TIMES heat exchangers which receive heat transferred from the Thermoelectric Devices (TED).[2:2] Heated waste water is pumped through bundles of hollow fiber membrane tubules whose external surfaces are exposed to a saturated gaseous environment maintained at a pressure of 2 psig.[2:2] Relatively clean product water permeates through the membranes and evaporates from their outer surface due to the high temperature of the waste water and the reduced pressure outside the tubes.[2:2] Accompanied by the evaporation is a decrease in temperature and an increase of the brine solute in the recycle flow.[2:2] The water vapor enters a condenser where it is cooled and partially condensed, releasing latent heat to the TED on either side of the condenser. The TED's act like heat pumps by

transferring the latent heat from the cooler condenser to the warmer waste water heat exchanger.[2:2]

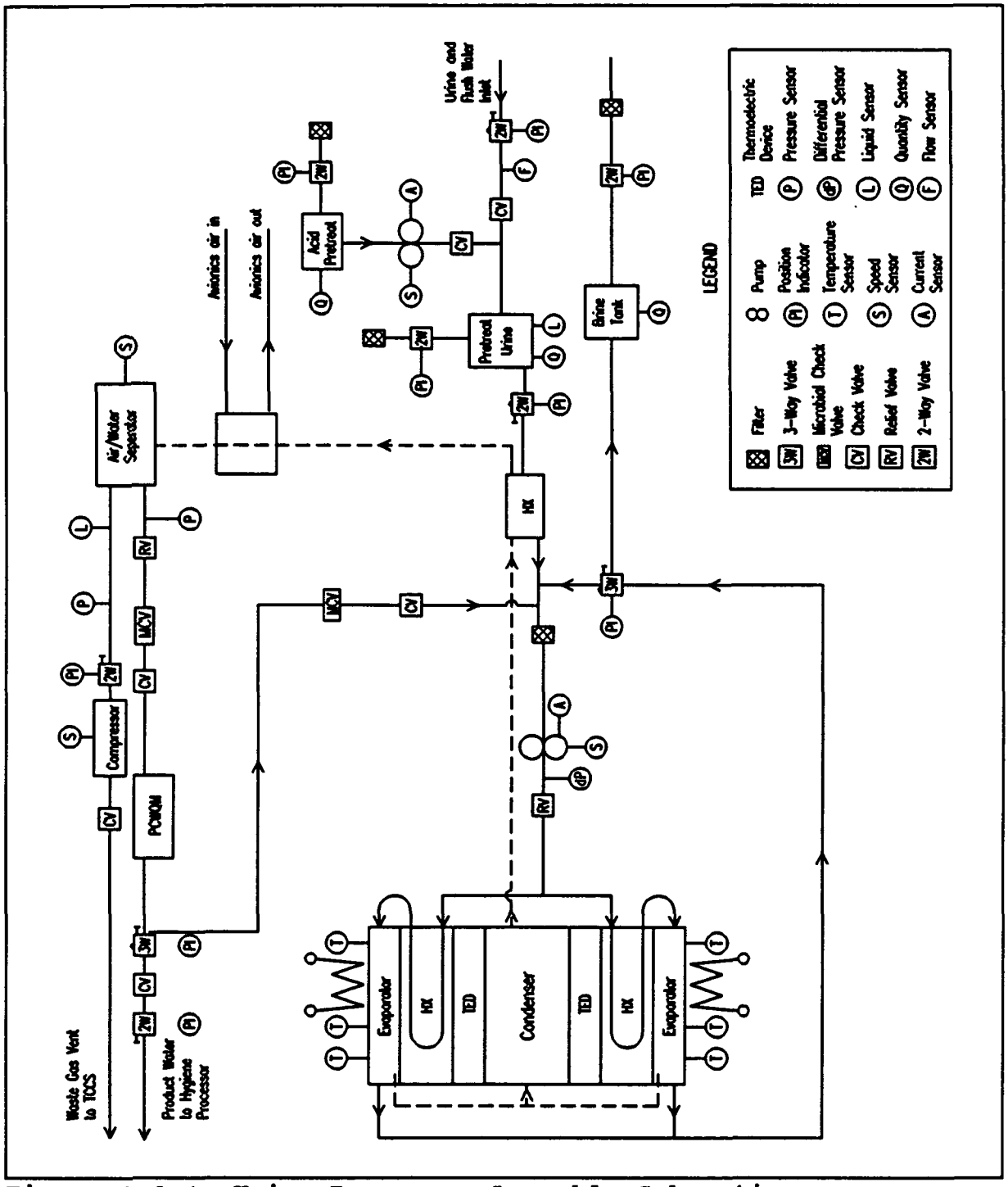

Figure 2.3-1 Urine Processor Assembly Schematic

Due to the waste heat generated by the electrical input to the TED's, only part of the steam condenses and a mixture of vapor and liquid leaves the condenser [2:2]. The mixture exits the TIMES and more vapor is condensed as the mixture passes through the regenerative heat exchanger and an air-cooled heat exchanger.[2:2] The mixture of water vapor and liquid water enters a rotary water separator where water is removed from the stream and the noncondensible gases are vented to the Trace Contaminant Control System (TCCS).[2:2] A compressor removes enough gas to maintain a pressure of approximately 2 psig at the water separator gas outlet.[2:2] The water leaving the separator is checked by the Process Control Water Quality Monitor (PCWQM) for conductivity, pH, total organic carbon (TOC) and microbes. A listing of water quality requirements is shown in Table 2.3-1.[2:Table 1]

Table 2.3-1. Product Water Quality Requirements

conductivity	$\leq$	150 micro-mho/cm
pH		3 - 8
total organic carbon (TOC)	$\leq$	50 ppm
microbes	<	1 cfu/100 ml

If the product water does not meet the PCWQM water quality requirements it remains in the recycle loop and is reprocessed. At about 25% dissolved solids the flow rate through the hollow fiber membrane tubules decreases below a minimum set point and the decreased flow rate initiates a brine dump to the brine storage tank.[2:1] Once the brine tank is full it is replaced with an empty one and the full tank is sent back to earth. Pretreated urine will displace the old recycle loop brine when processing begins again.[2:1]

2.3.2 URINE PROCESSOR ASSEMBLY CONTROLS

CURRENT URINE PROCESSOR ASSEMBLY CONTROL SCHEME

The current control scheme used for the Urine Processor Assembly (UPA) subsystem is a batch process or timed process depending on the situation. When the pretreated urine tank is 60-85% full or the time elapsed since the last processing is 48 hours then processing will begin. Pretreated urine should not be stored longer than 48 hours. See Figure 2.3-2 for a schematic of the current system.[4:1 of 1]

PROPOSED URINE PROCESSOR ASSEMBLY CONTROL SCHEME

A timed control scheme was considered for controlling the UPA, but the efficiency of the equipment would be decreased if small batches were run on timed intervals. The batch process was found to be the best method for controlling the UPA. A few additions were made to improve the system and these are explained below.

An Emergency Process mode was added to the existing modes of On, Off, Standby, and Emergency Shutdown.[5:1] The Emergency Process mode would be initiated when there is a need for waste water to be processed immediately because of another system failure. This mode would be included in the Standby mode, so if directed the UPA will begin processing even though the pretreat tank is not 60-85% full.

Another system improvement would be to add a pH sensor to the inlet of the UPA subsystem. Currently there is a small quantity of sulfuric acid being injected into the urine and waste water stream as it enters the waste water bus and flows into the pretreated urine tank. Since the pH of human urine varies from 4.8 to 8.4 depending on changes in the body metabolism or diet, the urine may need to be treated with more or less sulfuric acid than the measured amount. A pH sensor located in the waste bus between the 2-way valve and the pretreat urine storage tank could indicate how much acid pretreat is required for that particular flush. This would decrease the bacterial growth and thus decrease the amount of recycling required through the UPA.

When the recycle water has reached 25% solute, the efficiency of the water permeating through the hollow fiber membrane tubules decreases and a brine dump is initiated. There are no sensors in the current system to indicate when the recycle water has reached 25% solute. A flow sensor was considered since the flow through the hollow fiber membrane tubules decreases as the solute increases. Measuring the conductivity of the recycling water would indicate when there is 25% solute in the recycling water. Once this set point is reached a brine dump would be initiated to the

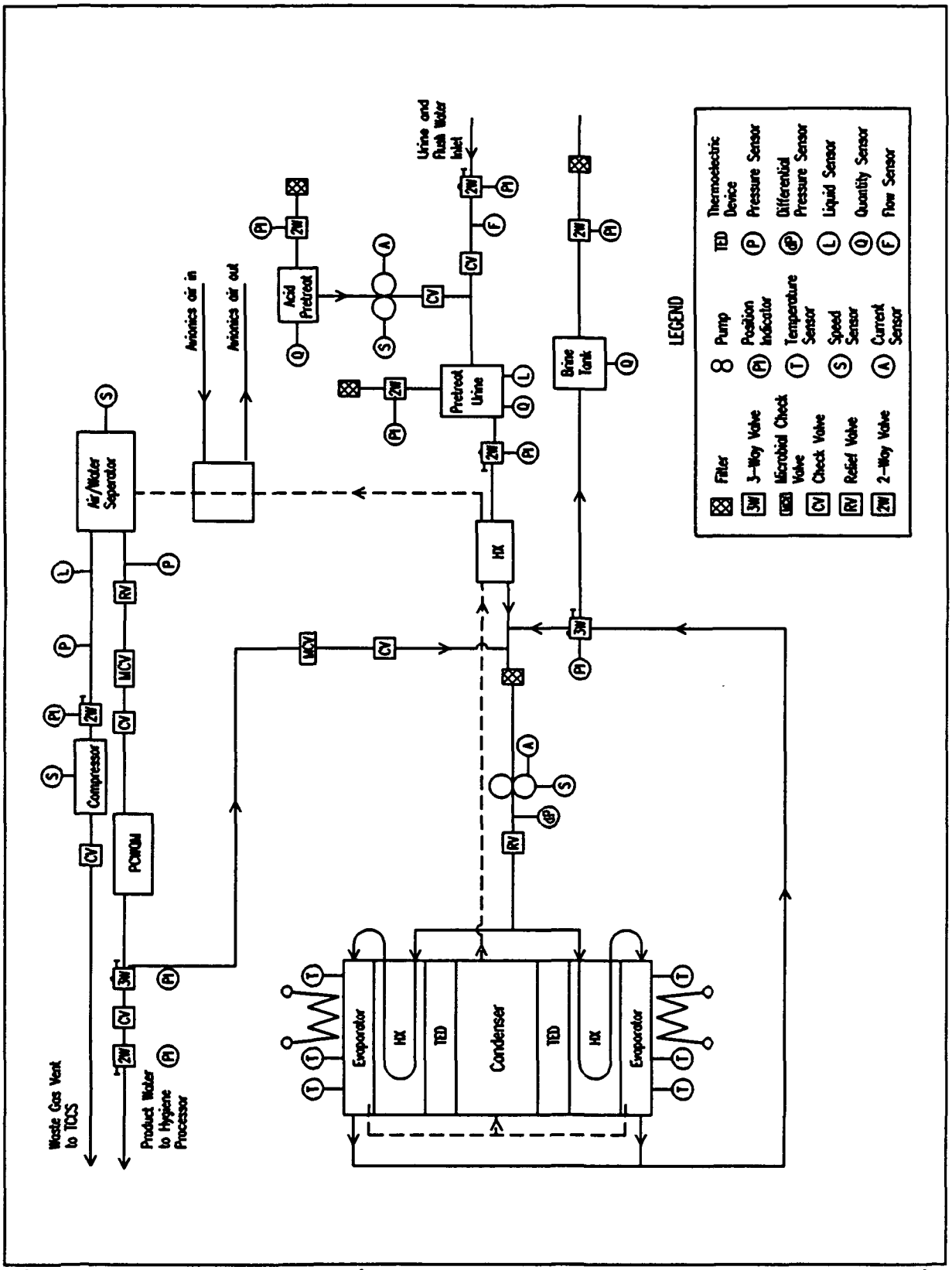

Figure 2.3-2 Current Urine Processor Assembly Control Schematic

brine storage tank. Other recommendations consist of placing additional sensors for system monitoring and failure detection.

The present UPA subsystem consists of a wide range of sensors. However, a few additional sensors could help track down specific problems that may occur. The recommended additions along with the current sensors are discussed in the following sections.

UPA MONITORING SCHEME

The following information is a list of all the major system components. Sensor methods currently being employed and suggested are included in these descriptions. For each sensor there will be a comparison made with a range of the desired output and an error signal will be generated. The controlling processor unit will determine if the sensor is giving correct output or if a failure has occurred. In the event of failure, the astronauts will be alerted. See Figure 2.3-3 for a schematic of the proposed system with the additional sensor placements.[4:1 of 1]

Acid Pretreat Tank

A quantity sensor is located in the tank to indicate when the sulfuric acid is getting low.

Pretreat Urine Tank

A quantity sensor is located in the tank to indicate when to begin processing. A liquid sensor detects any pretreated urine leaks.

Heat Exchangers

There were no sensors placed near the heat exchangers located external to the Thermal Integrated Membrane Evaporative System (TIMES). It is suggested that temperature sensors be placed at the outputs and inputs of the regenerative heat exchanger to monitor any temperature deviations that would indicate that the heat exchanger was not working properly. Flow sensors were placed in the air-cooled heat exchanger to monitor any changes in the flow of air.

Pumps

The two pumps used in the UPA are the acid injector pump and the recycle loop pump. Each one has a speed sensor and a current sensor. A differential pressure sensor was included with the recycle loop pump and it is suggested that a differential pressure sensor be added to the acid injector pump.

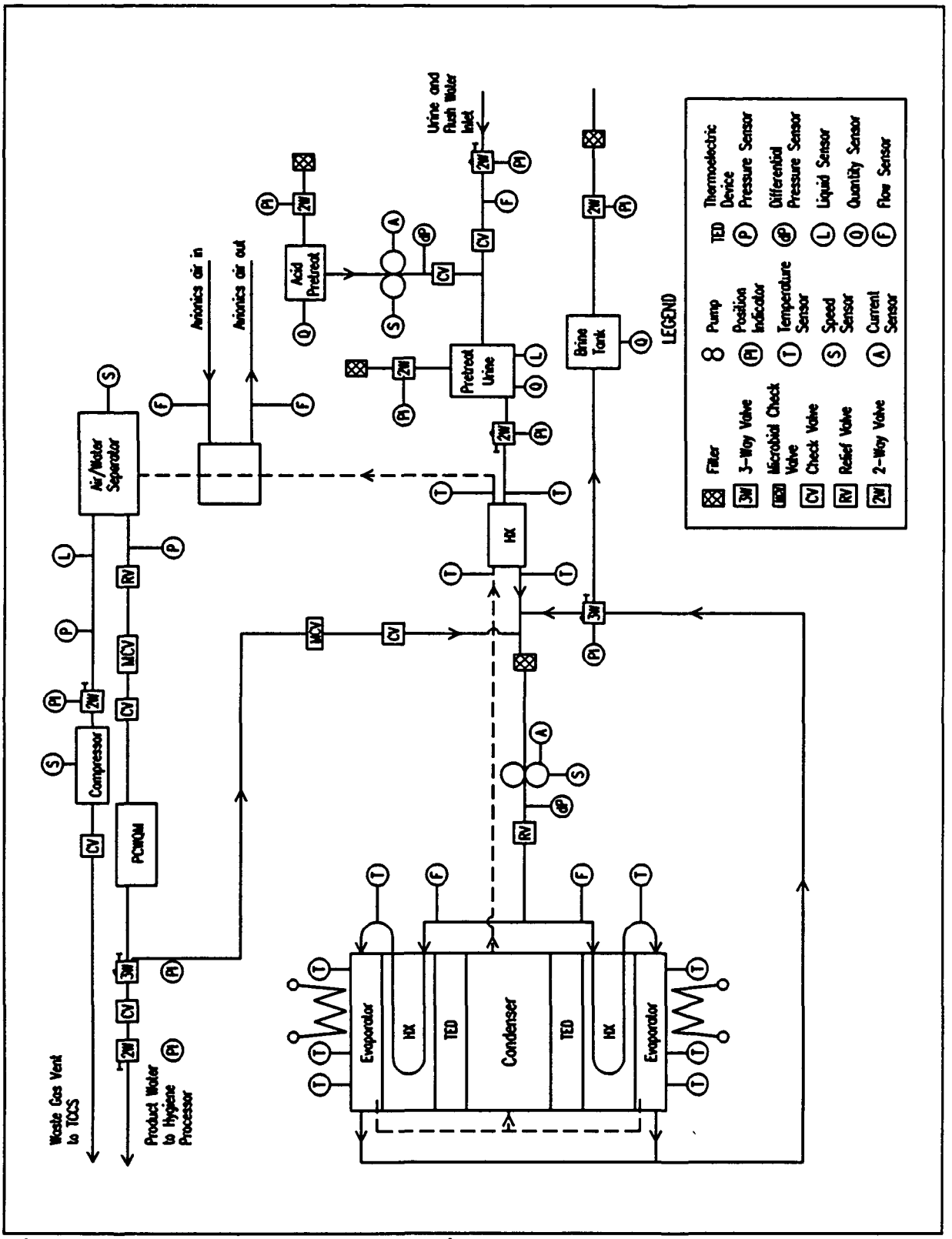

Figure 2.3-3 Proposed Urine Processor Assembly Control Schematic

TIMES

The TIMES includes evaporators, heat exchangers, thermoelectric devices, and a condenser. There are currently three temperature sensors located on each evaporator to monitor the efficiency. A temperature sensor was added to the output of each heat exchanger to check the efficiency of the thermal electric devices to transfer latent heat to the recycling waste water. Flow sensors were added to the inlets of the TIMES to monitor any obstructions that may occur in the pipes.

Air/Water Separator

A speed sensor is located on the air/water separator to monitor the compressor speed. In the product water line there is a pressure sensor which monitors the separator liquid pressure. A relief valve is included to relieve any excess pressure buildup. In the waste gas line there is a liquid sensor to sense any leaks in the separator. A pressure sensor to monitor the separator gas pressure is also in the waste gas line.

Compressor

A speed sensor is located on the compressor to determine if it is running.

Valves

All of the valves, excluding the 2-way valves located between the filters of the acid pretreat tank and brine tank, have manual back-ups. Motor driven valves are indicated by a semi-circle setting on top of the valve in Figure 2.3-3. If the valve is not motor driven then it is a solenoid valve which is electrically actuated. All the 2-way and 3-way valves have position indicators.

Check Valves

The check valves prevent backflow from contaminating other lines.

PCWQM

The Process Control Water Quality Monitor is explained in section 2.2 in this chapter.

Flow Sensors

It is suggested that flow sensors be placed in select locations along the recycle loop to indicate any obstructions in the piping.

COMPARISON OF CONTROL SCHEMES

The basic control scheme of the current control system and the proposed control system are the same. The batch process will continue to be the controlling system of the UPA subsystem. The proposed pH sensor in the waste water bus line would allow a more accurate amount of acid pretreat to be added. This would decrease the amount of bacterial growth and recycling necessary through the processor. The additional Emergency Process mode would be advantageous in emergency situations. Measuring the conductivity of the recycling water to initiate a brine dump would also be highly effective. Also, the recommended sensor additions can allow for more fault protection, detection, and isolation. Although the additional sensors of the proposed system could increase the cost of the system, it would be more productive and efficient.

2.4 HYGIENE WATER PROCESSOR

2.4.1 HYGIENE WATER PROCESSOR DESCRIPTION

The Hygiene Water Processor (HWP) consists of four major sections: gas pressurization, waste water storage, water processing and product water storage.

The Gas Pressurization section pressurizes the air on the air side of the bellows tanks (water busses). This pressure is necessary in order for the water to flow through the processor.

Water enters the HWP at a pressure of 34.5-82.74 kN/m^2 (5-12 psig). This pressure range allows the waste water storage section to maintain a waste water bus pressure between 34.5-82.74 kN/m^2 (5-15 psig) during both static and flow conditions. The gas pressurization section maintains this pressure by monitoring the water inlet pressure and by opening a valve on the gas side of the tank to lower pressure if the pressure exceeds 82.74 kN/m^2 (12 psig) and opening a valve from the higher pressure section if the pressure falls below 34.5 kN/m^2 (5 psig). If the waste water storage tank level exceeds 80% capacity. The inlet valves will be closed to prevent potential damage to the water storage tanks. A check valve is placed before the inlet valve on the waste water storage tank. This prevents the delivery of water from the waste water storage tank back to the waste water bus in the event of a waste water bus depressurization due to external leakage.[1:1]

When water is available in the waste water storage section, and the product water tanks are not full, the processing section will process and deliver water from the waste water storage section to the product water storage section. A water pump draws the waste water from the waste water storage tank to the water processing section. In this water processing section the water goes through a sterilization loop that sterilizes and filters the waste water. Sterilization is accomplished by heating and maintaining the water at a temperature greater than 121° C (250° F) for 20 minutes. For energy efficiency, a regenerative heat exchanger is used in conjunction with the particulate filter electrical heater to achieve the sterilization temperature. After heating, the water goes into the combination filter-residence tank, which uses a 40 micron filter and provides for a residence time greater than 20 minutes. Water leaves the filter and exits from the sterilization section by flowing through the hot side of the regenerative heat exchanger. The pressure of the cold inlet and cold outlet water sides of the regenerative heat exchanger are monitored to detect an expended filter and insure proper pressure maintenance to maintain water in the liquid state.[5:4]

The water then enters the recycle loop which circulates the water through two membrane-based filtration modules. The first membrane filtration is done in the Ultrafiltration (UF) module. Suspended

solids and dissolved macromodules (i.e. skin cells and soaps) are rejected while lower molecular-weight species pass through the membrane and become feed for the second stage of membrane filtration. The Reverse Osmosis (RO) module, is capable of reversing this feed flow within the UF module periodically to backflush the module and prolong membrane life. A valve also directs the outflow from the UF module to the brine processor to flush out the concentrated brine solution and replace it with fresh waste water. The reservoir tank placed before the recycle loop reduces the brine concentration in the recycle loop. This will allow an increase in the time between backflushes.[1:2]

The permeate water from the UF module passes through an air/water heat exchanger to cool the water to a temperature below 45° C (113° F) and prevent the delivery of water with a temperature in excess of 45° C (113° F) to the RO module. The permeate water then enters the second membrane-based filtration module, the RO module. The RO module removes low-molecular weight species (i.e. salts and low molecular weight organic material) which passed through the UF module, while allowing water to pass through and return to the recycle flow upstream of the reservoir tank. A check valve prevents backflow of fresh waste water through the RO module. In this way the RO module separates a relatively cleaner (compared to the UF permeate) water, which is then delivered to the unibeds. The RO permeate water is monitored by both a pressure sensor and a conductivity sensor. The pressure sensor will determine if flow blockage downstream of the RO has occurred. The conductivity sensor is used to determine when to backflush the recycle loop. A check valve prevents the backflow of water from the unibeds into the RO module. This backflow could damage the RO module.

The RO permeate then flows to the post-treatment loop. The post-treat loop consists of two unibeds that use the principle of adsorption to remove contaminants. Adsorption is a process in which contaminant particles are deposited on the surface of solid particles. The solid particles known as adsorbents, are chosen to preferentially adsorb the contaminants expected in the waste stream. The amount and type of material adsorbed varies from adsorbent to adsorbent. The unibeds in the HWP are arranged in a series configuration, such that when one bed is expended, the next bed is used. A new bed is then placed in the spot vacated by the expended bed. These beds will be replaced in staggered intervals of 30 days.[5:4]

The Process Control Water Quality Monitor evaluates the quality of the processed water and determines if it meets the required specifications. If it does not, the water must be redirected to the water pump inlet to be reprocessed. The PCWQM is discussed in detail elsewhere in this report.

The product water storage section stores the processed water. If the product water tanks are full, the inlet valves are closed and the HWP enters a standby state. The only other time the inlet valves are closed is during sterilization. If the product water tanks are empty, the outlet valves are closed. Otherwise, the outlet valves are kept open, and the water bus pressure is controlled between 124.1-206.9 kN/m^2 (18-30 psig) by the pressurization section. This pressure range allows the product water storage section to maintain a product water bus pressure of between 102.4-206.9 kN/m^2 (15-30 psig) during both static and flow conditions. The pressurization section maintains this pressure by monitoring the water pressure and by operating a valve to ambient pressure if this pressure exceeds 206.9 kN/m^2 (30 psig) and by turning on the compressor if this pressure falls below 124.1 kN/m^2 (18 psig). The product water tanks can accept processed water and at the same time deliver stored water to the hygiene water bus.[1:3]

2.4.2 HYGIENE WATER PROCESSOR CONTROLS

CURRENT HYGIENE WATER PROCESSOR CONTROL SCHEME

The present system uses a continuous cycling of waste water. The water processing section will process water when the waste water storage tank is full and there is room in the product water tanks for the processed water. The mode of operation is determined from a computer used by the astronauts. The water is sterilized and filtered in the processing section. All valves and major system components are controlled by electrical signals that arise from sensors placed throughout the system. The hierarchy of this control scheme is yet to be done. Detection of failure in the current scheme is limited to pressure, temperature and conductivity sensors and the PCWQM system. Again, the hierarchy of this control scheme is for future work [3:1-2].

PROPOSED HYGIENE WATER PROCESSOR CONTROL SCHEME

This control scheme will be based on the amount of water to be processed and the amount of water in the product water storage tanks. The waste water will be processed when the tank is 60% full and the product water tanks are only 60% full. The mode of operation will be determined by the ratio of the amount of water to be processed to the amount of water in the product water storage tank, thus allowing the system to operate independently of an external command. The modes of operation are standby, on, off and emergency shutdown.[1:4]

Hygiene Water Processor Standby Mode

When the waste water storage tank is less than 60% full or the product water tank is full, the processing section will be in standby mode. This mode is used for equipment preheat, tank pressurization and setting the valves to the ready position.

Hygiene Water Processor On Mode

When the waste water storage tank is over 60% full and the product water tank is less than 60% full the processing section will begin processing waste water. The processing occurs as discussed in the operational description of the HWP. A submode of the on mode is backflush. During backflush normal processing is stopped and the filters are backflushed to clean and extend their useful life.

Hygiene Water Processor Off Mode

When the waste water storage tank is less than 30% full and the product water tanks are greater than 20% full the processing section will go into the off mode. During off mode processes are gradually stopped and the power is removed from the system. While the system is in this mode the system is isolated and can be serviced.

Hygiene Water Processor Emergency Shutdown Mode

When either a part of the hygiene water processing system breaks or the power consumed by this system is needed elsewhere the processor will go into emergency shutdown mode. This mode is similar to off mode except the system is stopped as quickly as possible.

SYSTEM MONITORING SCHEME

The major components and the sensors associated with each component are listed in the following paragraphs. The monitoring scheme will allow the system to be controlled, and allow component failures to be detected. The output of the sensor will be compared with a desired output and an error signal will be generated if the output is not within the determined limits. A microprocessor will monitor and determine, through the use of an artificial intelligence program, if a component failure has occurred. The astronauts will be alerted via a message displayed on a computer terminal in the event of a failure. The necessary steps to correct any failure can then be taken.[4:1-12] Figure 2.4-1 shows a schematic of the proposed system.[1:4]

Gas Pressurization Section

Pressure sensors are placed on the waste water inlet line to determine if the water pressure is between 34.5-82.74 kN/m^2 (5-12 psig). If the pressure is either high or low then air is either released or added to the gas side of the storage tank. Quantity sensors are located on the water busses (bellows tanks) to determine if they are pressurized. Quantity sensors are used to measure the amount of gas or liquid contained in a tank.

Solenoid Valves

Position indicators are located on each valve. The position of the valve is compared to the position required during the process. If a valve becomes stuck open or shut the system will alert the astronauts and begin emergency shutdown mode.

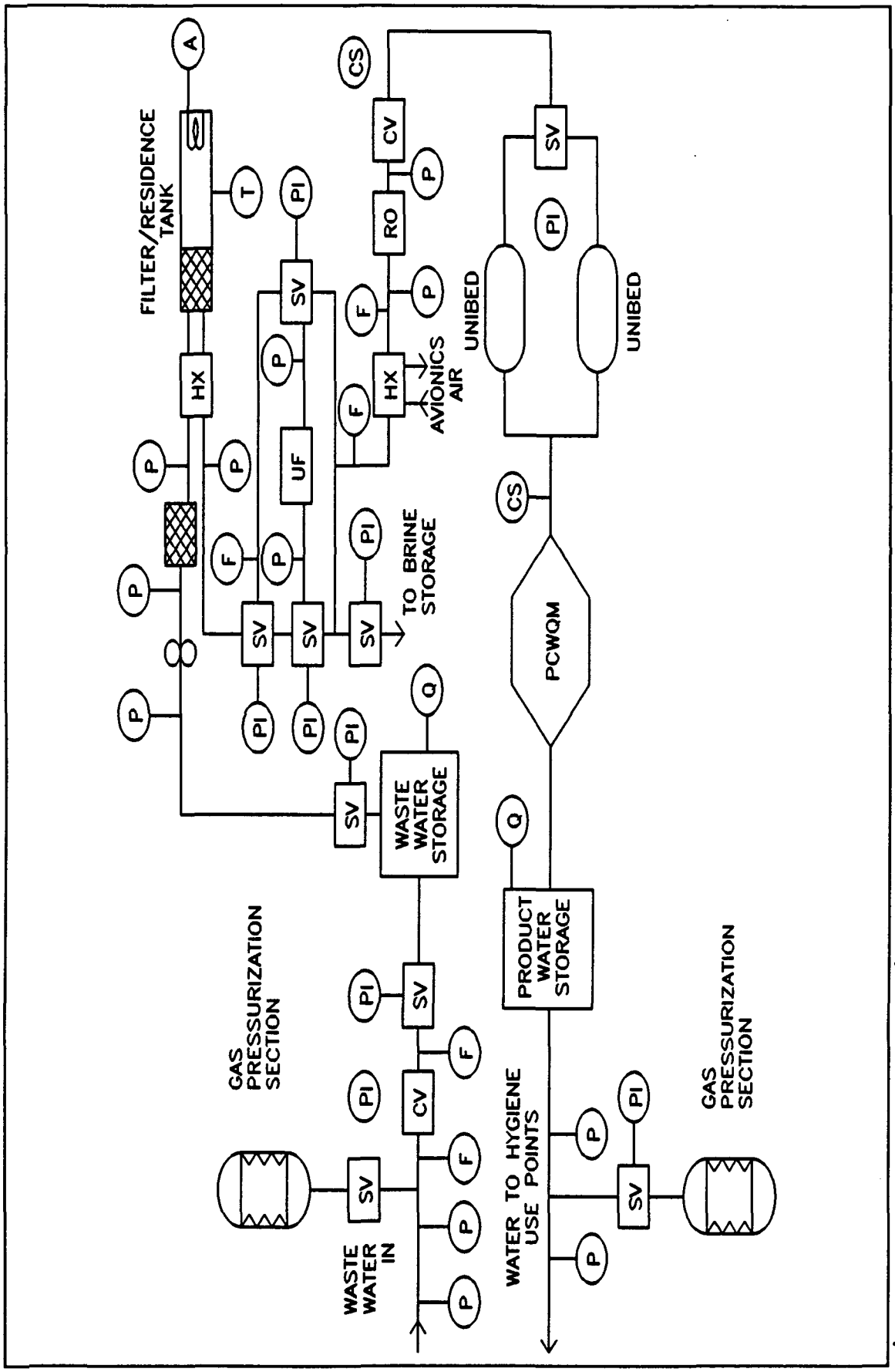

Figure 2.4-1 Proposed Hygiene Processor Schematic

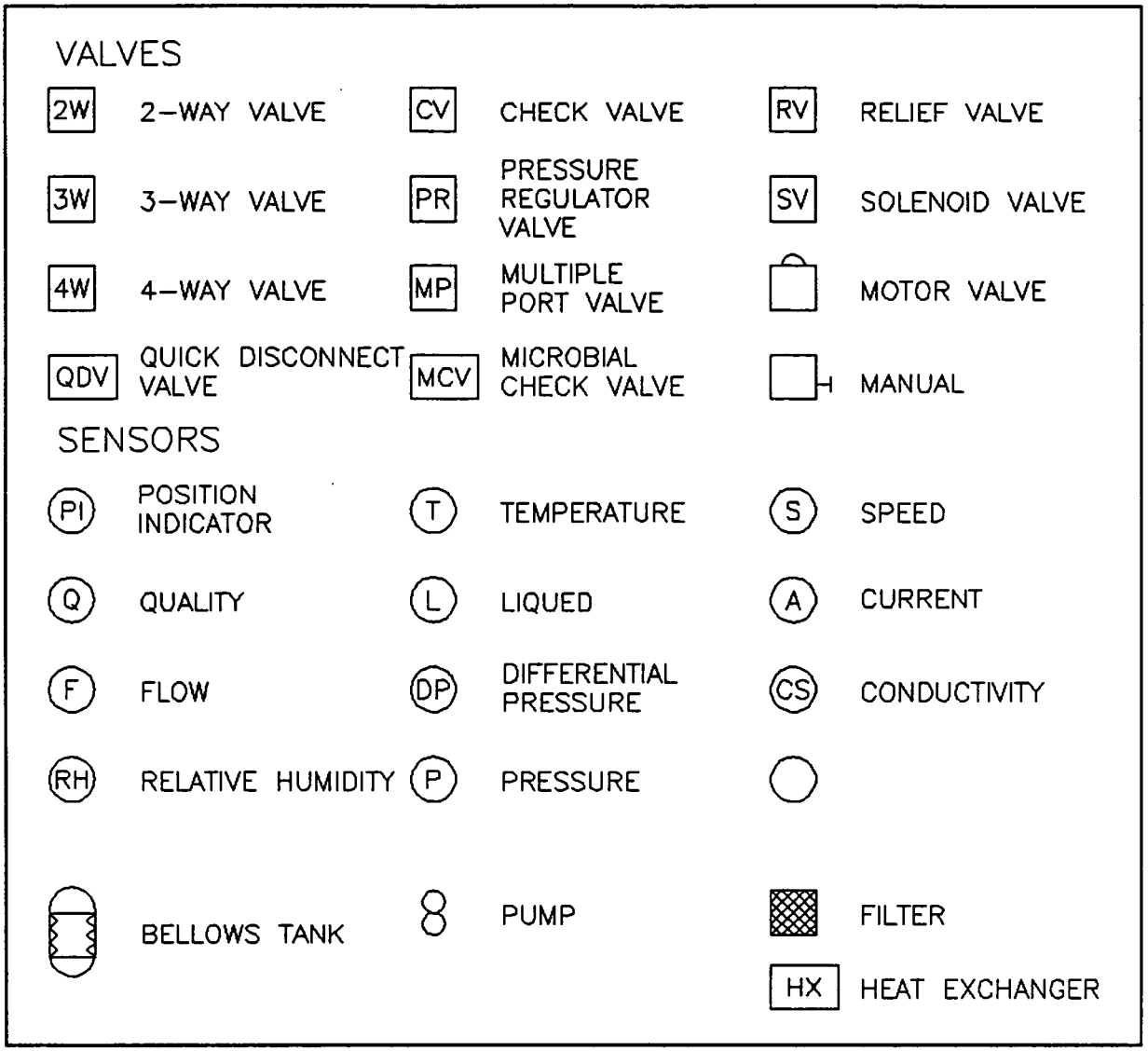

Figure 2.4-2 Legend of the Proposed Hygiene Water Processor

Check Valves

A flow sensor will be placed on either side of a check valve to determine if the valve is leaking. Check valves assure that water will not flow backwards in the system. Backwards flow could possibly damage the system.

Waste Water Storage

A quantity sensor is located on the storage tank to determine if the processing cycle should begin, the system should go to standby mode or if the system should proceed to off mode. The quantity sensor also determines if a solenoid valves placed on the inlet and outlet sides of the tank should be open or closed. These valves either allow the water to be processed or stored.

Water Pump

Pressure sensors are placed on either side of the pump to determine if it is working properly. Values for the pressure on either side of the pump are checked with predetermined values in the microprocessor program. If these values are not within the predetermined range the system will go into emergency shutdown mode.

Heat Exchanger

Pressure sensors are placed on the cold inlet and outlet sides of the heat exchanger to detect an expended filter and ensure the water is at the proper pressure to remain in the liquid state.

Filter/Residence Sterilization Tank

Temperature sensors are located in the tank to ensure that the water has reached its sterilization temperature of 121.1° C (250° F). A current sensor is also located on the power supply to the heater to determine if it is working properly.

Ultrafiltration Module

Pressure sensors are located on either side of the UF module to determine if the filter is expended. Values for pressure still need to be determined. A flow sensor is also located before the UF module to determine if it is being backflushed or not.

Heat Exchanger

Flow sensors are placed on both the inlet and outlet sides of the heat exchanger to determine if it has become clogged.

Reverse Osmosis Module

Pressure sensors are located on either side of the RO module to determine if the module is functioning properly and determine if blockage has occurred. A conductivity sensor will be used to determine when to flush the recycle loop.

Unibeds

A conductivity sensor is placed between the two unibeds to determine if they are expended. Each unibed will last approximately 60 days according to current data.

Process Control Water Quality Monitor

This is a separate subsystem of the WRM and is discussed at length in section 2.2 of this report.

Product Water Storage Tank

A quantity sensor is placed on the product water storage tank to determine if the system should begin processing, go to standby mode, or switch to off mode. The quantity sensor determines the position of the solenoid valve on outlet of the tank. If the tank becomes empty these valves would be closed.

Gas Pressurization Section

Pressure sensors are located on the exit side of the product water storage tank to determine if the pressure is between 34.5-82.74 kN/m^2 (15-30 psig). Quantity sensors are located on the water busses to determine if they are pressurized.

COMPARISON OF CONTROL SCHEMES

The new control system has advantages over the current control scheme. The primary advantage is that the new control system has feedback to determine more accurately the condition of the system. It operates on an as-needed basis, processing water only when there is a sufficient amount to be processed and when water is needed. This enables the new system to operate more efficiently by consuming less power. The new system also frees up the astronauts to perform their duties and not be bothered with controlling the system. This control scheme also contains many fail safe features. This allows the system to shut itself down if a problem arises that would endanger the astronaut's safety.

The proposed control scheme has three main disadvantages; it will be more complex, consume more space and be more costly to implement. The new method also requires further research into the proper operating pressures of each point and a complex hierarchy of control.[3:1-2]

2.5 POTABLE WATER PROCESSOR

2.5.1 POTABLE WATER PROCESSOR DESCRIPTION

Figure 2.5-1 [2:8] is a diagram of the Potable Water Processor (PWP). The purpose of this subsystem is to produce potable water for consumption by space station astronauts. Condensate from the Temperature and Humidity Control (THC) and Atmosphere Revitalization (AR) subsystems provide feed water for the PWP. The Potable Water Distribution Bus (PWDB) distributes processed water about the space station. Two processors, one in the Habitation Module and one in the Laboratory Module, comprise the PWP. Each has a capacity based on 2.84-5.16 kg H_2O/man-day with each processor operating for forty hours[3:1]. Two twenty hour processing periods exist in the forty hour cycle. The break allows the Condensate Storage Tank to refill should the Condensate Storage Tank empty before the end of the forty hour cycle.

During a normal processing cycle, condensate from the Condensate Storage Tank enters the Filter/Heater at a nominal rate of 1.81 kg/hr[3:1]. The Condensate Storage Tank and Main Pump bring system pressure to 103.4-206.8 kPa[3:1]. The Filter/Heater raises the water from ambient temperature to 121°C. The water remains at this temperature for at least 20 minutes[3:1]. The Heat Exchanger pre-heats water entering the Filter/Heater and allows the effluent to move to the Unibeds at ambient temperature. The Filter/Heater destroys microorganisms and filters particulate.

The Unibeds are multifiltration beds composed of ion-exchange resins and activated carbon. They remove inorganic and high molecular weight organic compounds. They also maintain a residual iodine level of 2-4 ppm[2:3]. The Unibeds have a 30 day life-time[3:1]. Replacing the filters involves rotating the second filter into the lead position and placing a new filter in the second position. The pressure and conductivity sensors monitor the Unibeds for clogging. The controlling CPU can automatically backflush the beds, based on bed age and sensor information. The Urine/Flush Water Processor receives backflush brine.

The water moves on to a Volatile Removal Apparatus (VRA Reactor). Hydrogen peroxide is the oxidant for a catalytic oxidation reaction in the VRA Reactor. Alcohols and low molecular weight organic compounds are oxidized for removal by a set of Polishing Ion-Exchange beds. The oxidation reaction takes place at 121°C or higher[3:2]. A regenerative Heat Exchanger pre-heats water entering the VRA Reactor and brings VRA Reactor effluent to ambient temperature. A Phase Separator removes oxidation process gas by-products and vents them to the Gas Rack. A gas check valve prevents backflow from the Gas Rack to the PWP. The Phase Separator is assumed to be passive and does not need monitoring.

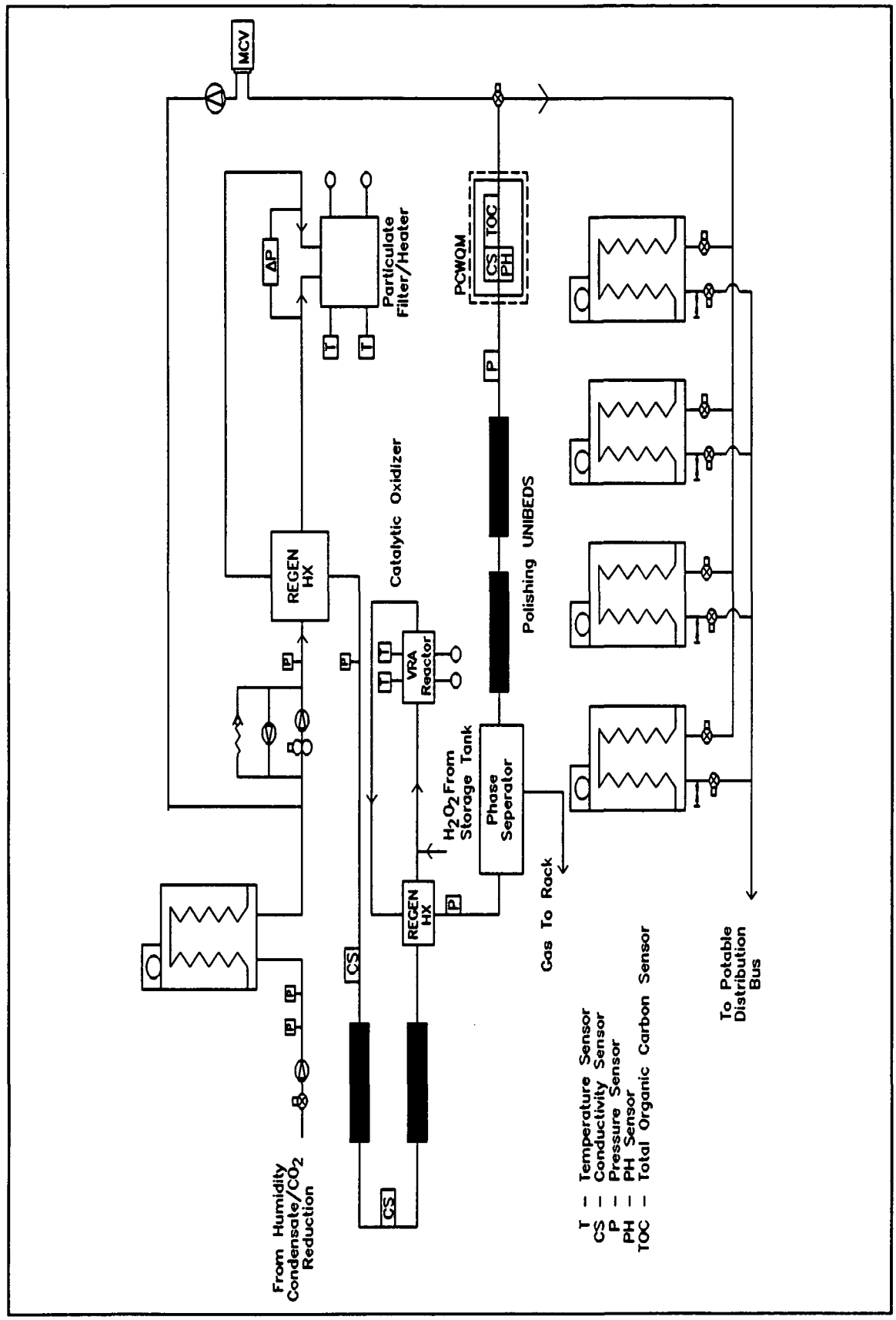

Figure 2.5-1 Multifiltration Potable Subsystem Schematic

Polishing Ion-Exchange beds follow the Phase Separator and remove the alcohols and low molecular weight organic compounds oxidized by the VRA Reactor. Pressure and conductivity sensors monitor for filter clogging. The controlling CPU can backflush these filters automatically with the Urine/Flush Water Processor receiving the brine.

The Process Control Water Quality Monitor monitors water quality during processing. Extensive batch testing by the Water Quality Monitor provides calibration information to the controlling CPU about the PCWQM. The PCWQM samples every 15 seconds to find pH, conductivity, temperature and total organic carbon content [Table 2.2-1]. These factors determine if the processed water will enter the Potable Water Holding Tanks or if the water should be recycled through the PWP. To recycle the water, a three-way solenoid valve redirects flow to the Main Pump inlet.

On its way to the Main Pump, the water passes through a Microbial Check Valve (MCV) that at present adds iodine or another biocide to the water. The MCV is assumed to be passive and lacks monitoring. A check valve on the MCV outlet prevents water flowing into the PWP from the Condensate Storage Tank from entering the MCV.

Each PWP has four Potable Water Holding Tanks. For each half cycle (20 hours) they have different functions. One tank is filling with processed water for later astronaut use while a second tank is filling with processed water for batch testing by the WQM. A third tank fills with potable water for standby and emergencies and the fourth tank supplies potable water to the astronauts. The tank supplying the astronauts is the tank that was filling for standby in the previous half cycle. These tanks have bellows to maintain an outlet pressure of 103.4-206.8 kPa on the PWDB. Control of bellows pressure is similar to that used on the Condensate Storage Tank. The two-way solenoid valves attached to the inlet and outlets of the Potable Water Holding Tanks have fail-safe positions reachable by spring force. They will close in the event of power failure or power removal. Manual movement of the valves is possible.

Each PWP processor has a controlling CPU. The controlling CPU makes decisions on processor mode, start and stop times and emergency control based on sensor information. The controlling CPU reports to a master CPU, which then communicates with the space station Systems Software. Providing each PWP with its own controller relieves the Systems Software of processor control. This allows a more sophisticated control scheme for the processor, capable of near autonomous operation.

2.5.2 POTABLE WATER PROCESSOR CONTROLS

CURRENT POTABLE WATER PROCESSOR CONTROL SCHEME

The current Potable Water Processor (PWP) has five modes of operation with sensors to report on the system status. The five modes of operation are: Initialization, Standby, Operate, Shutdown and Stop[3:3]. Space Station Freedom will have two PWPs, one in the habitation module and one in the laboratory module. They operate one at a time. Each processor operates for forty hours. The operating processor will continue to process condensate stored in a tank until the tank is empty or the Systems Software signals a stop.

PROPOSED POTABLE WATER PROCESSOR CONTROL SCHEME

The proposed PWP control scheme uses additional sensors for monitoring of storage tank pressure and three-way solenoid valves to allow backflushing of individual filters. Figure 2.5-2 shows placement of sensors and control valves. In the proposed system, each PWP processor will be controlled by a dedicated CPU with a third CPU monitoring both processors and reporting to the Systems Software. Changes and additions to the processor modes provide greater control of the system during normal operation and emergency situations. The proposed modes of operation are: Initialization, Update, Standby, Process, Normal Stop, Emergency Stop, Backflush, Manual Valve Control, Receive, and Off.

The Initialization mode is the primary start-up mode for the PWP. The controlling electronics of the PWP and the heating element of the filter/heater receive power. If pumps that can be engaged and disengaged while maintaining a constant motor speed are present, then they will be powered. Bellows Storage Tank Compressors are powered. The valve positions will not be changed during this mode.

Update mode will usually follow Initialization and the Controlling CPU will find all valve positions to prepare for moving into Process, Backflush or Manual Valve Operation modes. The CPU also checks fluid levels in the Condensate Storage Tank, Potable Water Storage Tanks, and the H_2O_2 Bladder Tank. If there is no condensate to operate on, or if the Potable Water Storage Tanks are full, then the system will move to the Off and Receive modes to conserve power.

In the Standby mode, all electronics are powered, as are heaters, pumps and compressors. The Controlling CPU will know all valve positions and fluid levels in condensate storage, potable water storage and H_2O_2 bladder tanks. Standby is a staging point for the processor to move to Process or Backflush modes. The processor

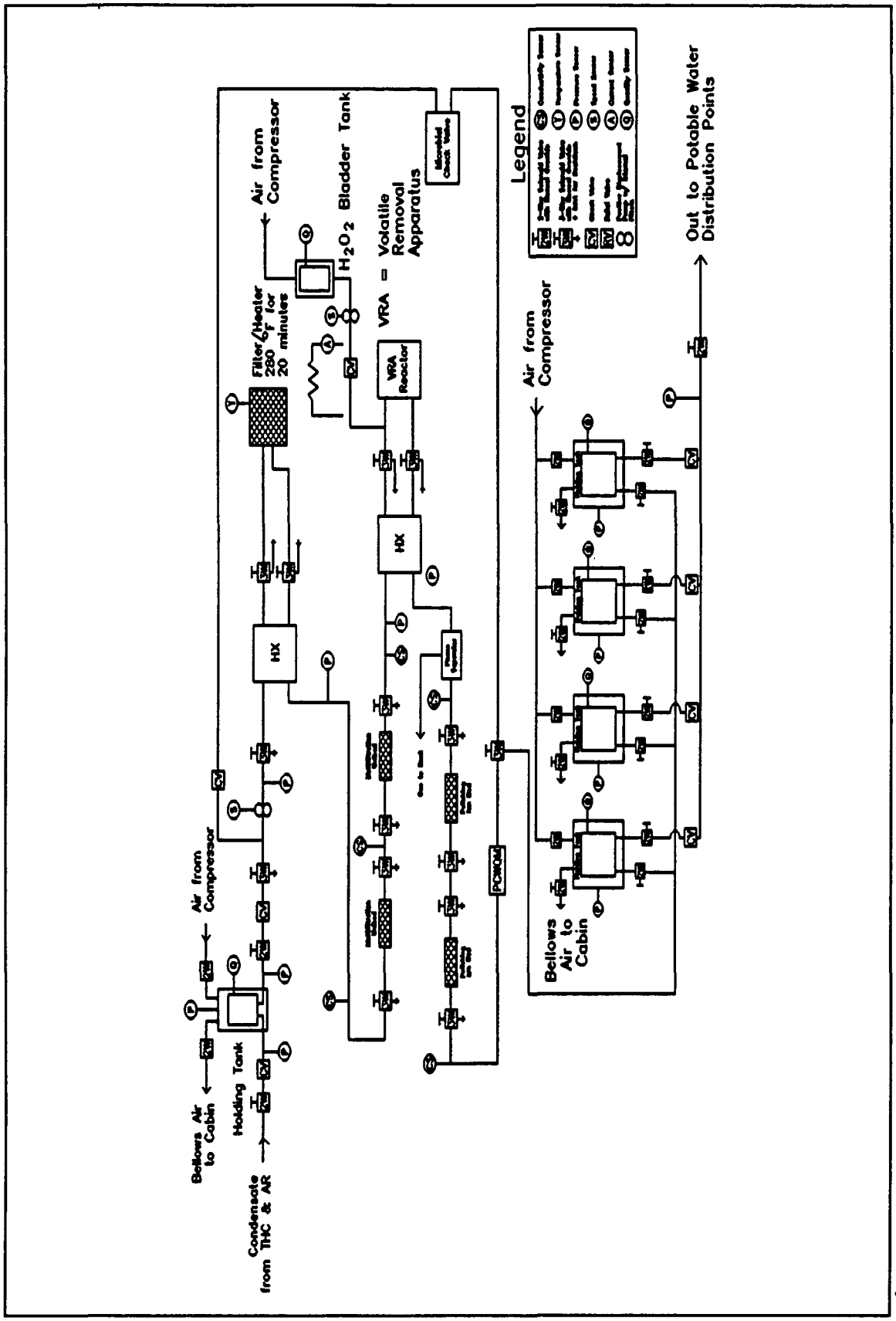

Figure 2.5-2 Proposed Potable Water Processor Control Schematic

will not remain in this mode for longer than it takes the Controlling CPU to change valve positions for the next mode of operation.

The PWP will spend most of its forty hour duty cycle in the Process mode. During this mode, the PWP is actively processing Condensate Storage Tank water. The Condensate Storage Tank will not receive condensate from the Atmosphere Revitalization (AR) or Temperature and Humidity Control (THC) sub-systems while Process mode is active. The pressure of the Condensate Storage Tank is monitored and maintained such that the outlet pressure is sufficient for passing water through the system. When the bellows pressure is within the proper range, the outlet valve will open and the main pump will engage to begin processing. The temperature of the filter/heater is monitored and adjusted as necessary. Line pressures, water conductivity, quality testing PCWQM and holding tank levels are monitored.

The Normal Stop mode of the PWP will be selected at the end of the processor's forty hour cycle, if all water from the Condensate Storage Tank has been processed, or if the Potable Water Holding Tanks are approaching capacity. The Controlling CPU will close the Condensate Storage Tank outlet valve but continue to power the Main Pump until the Potable Water Storage Tank quantity sensors report no addition of water to the receiving tank(s). The three-way valve following the PCWQM is then set to recycle water back to the main pump inlet and power will be removed from pumps, heaters, and compressors. If the processor must be restarted after the Normal Stop mode, the Initialization and Update modes must be activated. The Off mode will generally follow the Normal Stop mode.

Emergency Stop provides a way to stop flow from the processor to the Potable Water Storage Tanks quickly. Upon receiving the signal to move to the Emergency Stop mode, the Controlling CPU will change the three-way valve after the PCWQM to recycle the processed water back to the main pump inlet. The outlet valve of the Condensate Storage Tank also will be closed. Emergency Stop removes power from the pumps, heaters and compressors. The Emergency Stop is nearly the reverse of the Normal Stop, except that water will be trapped in the system. Most of the trapped water can be removed by the Backflush mode or by operating the PWP in the Manual Valve Operation mode. To begin processing water after an Emergency Stop, the Initialization and Update modes must be used.

The Manual Valve Operation mode allows the astronauts to start and stop pumps and compressors manually, to have manual control of heater temperature and manual control of valve positions. The valve positions can be changed at the valve or via the Systems Software, Master CPU and dedicated CPU. This mode allows the astronauts complete control over the system during repairs or routine maintenance (i.e., changing of filters). This mode can only be reached after Initialization. If the system fails and

cannot be initialized, then the Systems Software must inform the Master CPU to use only the undamaged PWP processor.

The Condensate Storage Tank cannot receive water from the AR and THC sub-systems during Update, Process, Normal Stop, or Emergency Stop modes. The Receive mode is a background task for the Initialization, Standby, Backflush and Off modes. When the processor is in the Receive mode, the bellows pressure is reduced below the inlet line pressure (but no lower than ambient pressure) and the inlet valve to the Condensate Storage Tank will be opened allowing condensate to flow into the storage tank. Process mode requires repressurization of the bellows.

The Off mode removes power from the electronics and generally will be accompanied by the Receive mode. The Off mode follows the Normal Stop and Emergency Stop modes. In an emergency or loss of system control, the Off mode can be executed by the astronauts via Systems Software. Executing the Off mode out of sequence (i.e., because of astronaut intervention), removes all power from the system. All solenoid valves in the PWP have fail-safe positions (inhibit flow into and out of system). In the event of abnormal power removal, these fail-safe positions are reached using springs. The Off mode cannot be executed by the Controlling CPU except following a Normal Stop mode or Emergency Stop mode.

The Controlling CPU will monitor sensors by multiplexing them onto fewer lines. If a sensor begins to send information requiring the attention of the CPU (i.e., pump speed dropping below a set level) the CPU will poll that sensor more frequently until the problem has been resolved. If a problem is not resolvable by the Controlling CPU, it will report to the master PWP CPU, which will decide whether to switch processing to the dormant PWP, continue processing with the ailing PWP, halt all processing, signal the astronauts via the Systems Software or some combination of the above.

MODES FOR PROPOSED CONTROL SCHEME

Potable Water Processor Power-up Mode

The normal Power-up Sequence begins with the master PWP CPU selecting the Habitation or Laboratory PWP. Processor condition (damaged or healthy) and last processor to operate determine PWP activation. If both processors are healthy, the dormant processor will begin a power-up sequence as the operating processor is following a normal stop sequence. The mode order for a normal power-up sequence is: RECEIVE (background mode, operating since last normal stop sequence) -> INITIALIZATION -> UPDATE -> STANDBY -> PROCESS. If the filters in the processor are nearing the end of their lifetime (approximately 30 days), the normal power-up sequence could be modified to contain a Backflush mode before

Initialization. The Backflush could be performed on each section, or on separate sections. The modified mode sequence is: RECEIVE -> BACKFLUSH -> INITIALIZATION -> UPDATE -> STANDBY -> STANDBY -> PROCESS.

Potable Water Processor Power-down Mode

The normal power-down sequence execution occurs at the end of a processor's forty hour processing cycle, if the Condensate Storage Tank is empty or if the Potable Water Storage Tanks are reaching capacity. The mode order for a normal power-down sequence is: PROCESS (current mode) -> NORMAL STOP -> OFF -> RECEIVE. If the filters are near the end of their lifetime and a backflush was not performed during the normal power-up sequence, a Backflush mode may be inserted following the Normal Stop mode. The modified normal power-down sequence is: PROCESS -> NORMAL STOP -> BACKFLUSH -> OFF -> RECEIVE.

Potable Water Processor Emergency Power-down Mode

In an emergency (i.e., power must be diverted to another space station function or there is an uncorrectable failure in the PWP) the Controlling CPU can initiate an emergency power-down sequence. The sequence of modes is: PROCESS (current mode) -> EMERGENCY STOP -> OFF. No Backflush or Receive mode is part of the emergency power-down sequence. The master PWP CPU also will alert Systems Software to the emergency power-down and provide a reason for the power-down.

SYSTEM MONITORING SCHEME

Water Storage Tanks (Condensate and Product)

A pressure sensor monitors the bellows pressure of each tank. If the bellows pressure is not in range for the current mode of operation, two solenoid valves provide a means for altering the pressure. One solenoid valve connects the bellows to a compressor to increase pressure. The fail-safe position for this valve is closed. The second valve can open to vent air to release pressure. The Gas Rack receives vented air. The fail-safe position for this valve is open. Level sensors monitor the quantity of water in the tanks. The Controlling CPU places this value in memory so that as the water level in the tank begins to decrease rapidly (i.e., leak) the CPU can change bellows pressure accordingly and, if necessary, initiate the Emergency Power-down Sequence.

Condensate Bus Inlet

A pressure sensor on the Condensate Bus Inlet monitors the water pressure so that the Condensate Storage Tank bellows pressure is lower than the bus pressure. If the bellows pressure has been

reduced to ambient pressure but the bus pressure is still lower, the Controlling CPU will alert the Master CPU. The Master CPU will then attempt to increase the bus pressure. A check valve placed between the two-way solenoid valve and the Condensate Storage Tank prevents backflow from the Condensate Storage Tank to the bus. Integral flow sensors in the check valve monitor for valve failure.

Condensate Storage Tank Outlet

The Condensate Storage Tank Outlet has a pressure sensor to monitor outlet pressure. Information from this sensor is used to adjust the bellows pressure. A two-way solenoid valve controls outlet of water from the Condensate Storage Tank. A check valve follows the two-way solenoid to prevent backflow into the storage tank. A second check valve appears on the Recycle loop of the PWP to prevent flow of water from the Condensate Storage Tank from entering the outlet of the recycle loop. The check valves have integral flow sensors to monitor for valve failure.

Pumps

Both the Main Pump and the H_2O_2 Pump have speed sensors. These sensors monitor for proper operation of the pumps and provide an estimate of the pump outlet pressure. The estimated outlet pressure provides a check on both the pump performance and the pressure sensors following the pump. Three-way solenoid valves surround the main pump. Forcing potable water backward through the pump to the Urine Processor Assembly backflushes the pump.

Filter/Heater and 1st Heat Exchanger

A set of pressure sensors surrounds the Filter/Heater and Heat Exchanger. These monitor input and output pressures to alert the Controlling CPU to leaks or the need for a backflush of the Filter/Heater. Three-way solenoid valves surround the Filter/Heater for backflushing. A temperature sensor in the Filter/Heater provides information for maintaining the 250° temperature required. A mechanical thermostat backs up the temperature sensor. The heating element has a current sensor to monitor power consumed.

Multifiltration and Polishing Unibeds

Conductivity sensors surround the two Multifiltration and two Polishing Unibeds followed by a pressure sensor. These sensors inform the Controlling CPU about the condition of the Unibeds. A marked increase in water conductivity from previous sampling suggests Unibed clogging. Each Unibed can be backflushed individually.

Volatile Removal Apparatus (VRA) Reactor and 2nd Heat Exchanger

A set of pressure sensors surrounds the VRA Reactor and Heat Exchanger. The H_2O_2 Bladder Tank has a controllable pump to release known amounts of H_2O_2 into the VRA Reactor. A level sensor on the H_2O_2 Bladder Tank and speed sensor on the pump inform the Controlling CPU about H_2O_2 release. The CPU releases H_2O_2 to match the flow of water into the VRA Reactor, as monitored by a flow sensor. Pressure sensors surround the VRA Reactor and Heat Exchanger to monitor for proper line pressure. Insufficient line pressure suggests a clogged VRA Reactor. Three-way solenoid valves surround the VRA Reactor for backflushing.

Phase Separator

The Phase Separator vents gasses to the Gas Rack. The Phase Separator is assumed to be a passive system and currently lacks a control scheme.

Process Control Water Quality Monitor

This is a separate subsystem of the PWP and is discussed at length in section 2.2 of this chapter.

Potable Water Storage Tanks

During potable water processing, each of the four tanks has a different function. One tank is filling with processed water for later distribution, another is filling with processed water for further testing by the Water Quality Monitor subsystem, another tank is on standby (empty) and the fourth tank is delivering water to the Potable Water Distribution Bus. Each tank has a level sensor to monitor fluid level inside the bellows. Bellows pressure is monitored and used to control the inlet two-way solenoid valve and vent two-way solenoid valve. The same compressor that supplies air to the Condensate Storage Tank is used to supply the Potable Water Storage Tank bellows with air. The tank designated as the Potable Water Supply Tank maintains a higher pressure to deliver water to the Potable Water Distribution Bus at 103.43 to 206.86 kPa gage. A pressure sensor on the distribution bus is also used to control the Potable Water Supply Tank bellows pressure. The two filling tanks maintain a pressure below line pressure (not to fall below ambient pressure) to allow filling. Check valves on the outlet lines of the Potable Water Storage Tanks prevent backflow from the distribution bus into the tanks. The check valves contain integral flow sensors to monitor for valve failure. A two-way solenoid valve on the distribution bus can be closed to isolate the storage tanks from the bus. The valve's fail-safe position is closed (isolate tanks). There are two of these valves present in the event of device failure.

Microbial Check Valve

The Microbial Check Valve (MCV) is used to add an additional biocide to the processed water as it is recycled to the main pump inlet. A check valve after the MCV prevents flow of water from the Condensate Storage Tank into the MCV. The MCV is assumed to be a passive system and lacks a control scheme.

Check Valves

All check valves in the PWP have integral flow sensors (not shown on the system diagram). These flow sensors notify the Controlling CPU of a valve failure. In the event of a pressure sensor failure, the check valve flow sensors indicate high line pressure. Depending on the sensor location, number of sensors reporting out of range, water waiting to be processed, Potable Water Storage Tank levels and time remaining in the processing cycle, the Controlling CPU can opt to continue processing or initiate the Emergency Stop Sequence. The Master CPU will notify Systems Software in the event of Emergency Stop Sequence.

Two and Three-Way Solenoid Valves

All solenoid valves in the PWP contain position sensors. These sensors report to the Controlling CPU the current valve position. If a valve is stuck, the Controlling CPU will initiate the Emergency Stop Sequence. The Master CPU will notify Systems Software of the Emergency Stop Sequence.

COMPARISON OF CONTROL SCHEMES

The proposed control scheme places the PWP under the control of a dedicated CPU which reports to a Master CPU. The Master CPU then reports to the Systems Software and ultimately the astronauts. By placing these "buffers" between the PWP and the astronauts, the PWP can 1) be self-contained, 2) self-repairing (in a limited way) and 3) portable. Because this control scheme is portable, the same PWP can be implemented on different craft, each with different Systems Software. The software of the Master CPU can be rewritten to interface with the host Systems Software, rather than modifying the host Systems Software and running the risk of losing control of another vital system. The current control scheme implies that the same software that is monitoring other life support systems also will be required to monitor the PWP.

Because the PWP will have limited self-repair capabilities, it can continue to run and ensure an emergency supply of potable water for the astronauts. The inclusion of a Manual Valve Operation mode makes the testing and repair of the system easier for the

astronauts. With this mode, they can perform dynamic testing on the system. The current control scheme only allows for static testing.

A self-contained system eases repair and upgrading of the system. Addition of sensors requires only the Controlling CPU software to be modified rather than the Master CPU software and/or the Systems Software.

Drawbacks to this system result from increased system complexity. The proposed control scheme has ten modes of operation, twice the current scheme. The addition of sensors to storage tank bellows and check valves will increase system cost. The writing of control software for Controlling CPUs and Master CPUs also will increase the cost. The number of sensors will produce a system that is impossible to test exhaustively. Computer simulation of the system must supplement physical tests. Reporting insufficient information to the Controlling CPU, the Master CPU or the Systems Software could result in a system failure and notification of the astronauts too late to correct the failure.

3.0 MATHEMATICAL MODELING

3.1 INTRODUCTION

The modeling of the Air Revitalization (AR) system is divided into three distinct areas of concentration. First is the Carbon Dioxide Removal Assembly. Next, the Carbon Dioxide Reduction Assembly, and finally the Oxygen Generation Assembly (OGA).

Carbon Dioxide Removal Assembly uses a series of molecular sieve beds to extract CO_2 from the cabin air of the Space Station. The CO_2 is removed from the air and is pumped to a CO_2 storage tank.

Stored CO_2 is drawn from the storage tank into the Carbon Dioxide Reduction Assembly for conversion to methane (CH_4) and water vapor. The methane/water vapor mixture is cooled and water is removed by a centrifugal separator. The methane is placed in a storage tank for later disposal.

The Oxygen Generation Assembly converts water to oxygen and hydrogen gas in electrolysis cells. An electrical current passes through electrodes in each cell, causing the "cracking" of water into hydrogen and oxygen gas at the electrode surface.

Math modeling serves two purposes. The first purpose is to provide a means of learning how each system operates. Second it provides the Expert System with data for its knowledge base.

The following subsections give an analysis of the three Air Revitalization subassemblies.

3.2 CO_2 REMOVAL ASSEMBLY MODEL

DESCRIPTION

The CO_2 Removal Assembly is part of the Air Revitalization (AR) Subsystem. Its purpose is to remove CO_2 from the cabin atmosphere, deliver CO_2 to the CO_2 Reduction Assembly and return humidified air to the cabin. This is done using a four bed molecular sieve consisting of: two desiccant beds to remove water vapor from incoming air, two CO_2 adsorption beds, a blower to force air through the system, a CO_2 pump, a CO_2 accumulator, a pre-cooler and five multiple-flow selector valves [1:8]. Figure 3.2-1 illustrates these major components.

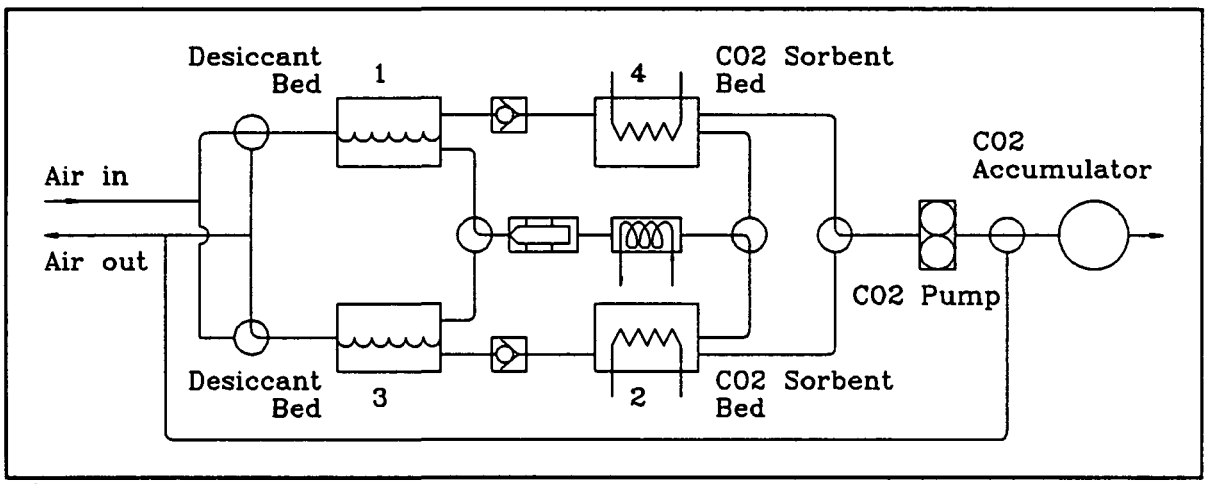

Figure 3.2-1 CO_2 Removal Assembly

MATH MODEL

Assumptions

1. All beds modeled as lumped systems.
2. For simplicity, we have neglected the fact that the adsorbing/desorbing processes vary as a function of distance through the bed.
3. Assumed thermal equilibrium which negates the dependence on bed length.
4. Thermal equilibrium assumed for the CO_2 desorbent bed.
5. Pump operation is 100% efficient, isentropic, and adiabatic.

6. The model can be improved by sub-dividing each bed into many smaller beds. This method results in plug flow operation in which the simulation takes place several times for each bed.

7. To implement the improved modeling scheme, the overall bed volume and sorbent mass will be divided by the number of plugs.

All four beds have been modeled as lumped systems. For simplicity, we have neglected the fact that the adsorbing/ desorbing processes vary as a function of distance through the bed. In our model, we have assumed instantaneous thermal equilibrium, which in effect, negates the dependence on bed length. We intend to improve the model by sub-dividing each bed into many smaller beds. Plug flow operation results, in which the simulation takes place several times for each bed. To implement this scheme, the overall bed volume and sorbent mass will be divided by the number of "plugs". The simulation will most likely be modified using 2-D arrays.

The relationship equations and state variable equations are as follows:

Equations

1) For the CO_2 desorbent bed assuming thermal equilibrium, one obtains

$$P_b = k_1 \left(\frac{m_d}{m_b} \right) (T_b - T_{ref}), \tag{1}$$

$$T_b = T_g, \tag{2}$$

$$P_g = \frac{m_g R T_b}{V_g}, \tag{3}$$

$$\frac{dm_d}{dt} = (P_b - P_g) k_2, \tag{4}$$

$$\frac{dm_g}{dt} = (P_b - P_g) k_2 - m_o, \tag{5}$$

$$\frac{dT_b}{dt}=\frac{\left(\frac{dm_d}{dt}S_c+Power\right)}{m_b c_{vb}}. \tag{6}$$

The required definitions are given as:

P_b = CO_2 equilibrium pressure of bed (kPa),
k_1 = constant (picked to be $0 \leq K_1 \leq 1$),
m_d = mass of CO_2 in sorbent material (kg),
m_b = mass of sorbent in bed (kg),
T_b = temperature of the bed (K),
T_{ref} = reference temperature (K),
T_g = temperature of CO_2 gas (K) (substitute T_b),
P_g = pressure of CO_2 gas in bed void space (Kpa),
m_g = mass of CO_2 gas in void space (kg),
R = CO_2 gas constant (kPa.m^3 / kg.K),
V_g = void space of bed, also volume of CO_2 (m^3),
k_2 = transfer coefficient ($0 \leq K_2 \leq 1$),
m_o = CO_2 gas mass flow rate, determined by pump (kg/sec),
S_c = heat of sorption of CO_2 (J/kg CO_2),
Power = power applied to bed (J/sec),
c_{vb} = heat capacity of sorbent material (J/kg.K),
dm_d/dt = rate of CO_2 desorbed (kg CO_2/sec),
dm_g/dt = change in mass of CO_2 in void space (kg CO_2/sec),
dT_b/dt = change in temperature of bed (K/sec).

Pump operation has been assumed to be 100% efficient, isentropic, and adiabatic. In actuality, no pump exhibits such behavior due to friction, windage, heat loss and pressure losses. For our first attempt, however, we will neglect these inefficiencies to simplify the model. At this time, this is an acceptable model as it provides the correct overall response.

2) For the rotary vane pump assuming 100% efficiency, isentropic and adiabatic operation, we have

$$m_i=m_o, \tag{7}$$

$$m_o=(\omega k_3)\rho_g, \tag{8}$$

$$\rho_g=\frac{P_g}{RT_b}, \tag{9}$$

$$T_p = T_b \left(\frac{P_p}{P_g} \right)^{\left(1 - \frac{1}{k}\right)}. \quad (10)$$

The symbols are defined as:
m_o = mass flow rate of CO_2 gas stream (kg / sec),
w = angular velocity of pump (rad / sec),
k_3 = transfer coefficient of pump ($0 \leq K_3 \leq 1$),
p_g = density of CO_2 gas (kg / m^3),
R = CO_2 gas constant (kPa.m^3 / kg.K),
P_g = entering CO_2 gas pressure to pump (kPa),
T_b = entering CO_2 gas temperature to pump (K),
P_p = exiting CO_2 gas pressure from pump (kPa),
T_p = exiting CO_2 gas temperature from pump (K),
k = specific heat ratio of CO_2.

3) The accumulator, during continuous operation, is governed by

$$T_a = \frac{u_a}{m_a c_v} + T_o, \quad (11)$$

$$P_p = \frac{m_a R T_a}{V_a}, \quad (12)$$

$$\frac{dm_a}{dt} = m_o - m_{out}, \quad (13)$$

$$\frac{dU_a}{dt} = m_o (T_p - T_o) c_v - m_{out} (T_a - T_o) c_v - Q_{loss}. \quad (14)$$

The needed quantities are defined by:
T_a = temperature of accumulator (K),
U_a = internal energy of accumulator (J),
c_v = specific heat of CO_2 gas (J / kg.K),
T_o = thermodynamic reference temperature for internal energy (K),
P_p = CO_2 pressure at pump exit (kPa),
R = CO_2 gas constant (kPa.m^3/kg.K),
V_a = volume of accumulator (m^3),
m_a = mass of CO_2 in accumulator (kg),
T_p = exiting temperature of pump (K),
dm_a/dt = change in total mass of CO_2 in accumulator (kg/sec),

dU_a/dt = change in internal energy of accumulator (J/sec),
m_o = CO_2 mass flow rate through pump (kg/sec),
m_{out} = CO_2 mass flow rate to CO_2 Reduction Assembly (kg/sec),
Q_{loss} = heat loss of accumulator (J/sec).

TRANSFER COEFFICIENTS AND INITIAL CONDITIONS

The final values of the adsorbing cycle determine the initial conditions $T_b[0]$ and $m_d[0]$ for the desorbing cycle. From these values, the initial conditions are found. From Equation (1) one obtains

$$P_b[0]=k_1\frac{m_d[0]}{m_b}(T_b[0]-T_{ref}) \tag{15}$$

Where the index of zero refers to the time relative to start of cycle. At $t_{cycle} = 0$ with $P_b[0] = P_g[0]$, using equations (1), (3) and (15) yields the initial condition $m_g[0]$ as

$$m_g[0]=\frac{V_g P_g[0]}{RT_b[0]} \tag{16}$$

Since the initial amount of CO_2 in the desorbing bed is known and will be desorbed in <u>about</u> 60 minutes, the coefficient k_2 can be estimated by (5) to obtain

$$k_2\approx\frac{m_d[0]}{60\cdot\min}(1)=0.0006 \tag{17}$$

Next, to get k_3, calculate $p_g[0]$ from (9) as

$$\rho_g[0]=\frac{P_g[0]}{RT_b[0]} \tag{18}$$

Taking the constant mass flow rate m_o to be dm_d/dt for w=200 rad/sec., k_3 can be found from (8) as

$$k_3\approx\frac{m_d[0]}{60\cdot\min}\left(\frac{1}{\rho_g\omega}\right) \tag{19}$$

$$k_3\approx0.0023$$

The k values are physical properties of the components and do not change. The values calculated above are not exact, but they are of the proper order of magnitude to allow model operation over a wide range of parameters.

MODELING TECHNIQUES

Solution of the governing equations relies on solving differential equations. This simple integration procedure gave results accurate only for small time steps. The original simulation program used a six second time step. In regions where the solution has a steep slope, the large time step forced the program to over or under estimate the answer. The over and under-shoot appeared as transients on graphs of state variable equations. These transients also caused negative absolute pressures and negative absolute temperatures. The erroneous negative numbers were fatal errors in program statements with exponential functions. Decreasing the time step to one second for the AT&T 3B1 model and 0.1 second for the IBM models alleviated the problems caused by steep solution slopes.

Decreasing the time step introduces new difficulties. Because data was saved in arrays, decreasing the time step increased the number of array elements. The limited memory of the AT&T 3B1's prevented modeling of more than about ten minutes of the cycle. To overcome this problem, temporary registers were used to hold state equation variables and other variables until about thirty seconds of the model has been simulated. After the thirty seconds, the temporary register values are placed in plotting arrays. In the case of CLPMODEL.C, the temporary registers are used to collect data for five (length is arbitrary) minutes after which time it is written to CLIPS sensor files.

Temperatures of the desorbing bed, CO_2 accumulator and CO_2 pump reached very high (>1000 K) levels in initial modeling attempts. Simple on/off control of the desorbing bed heater and cooling of the accumulator were used to keep the temperature within reasonable values. The controllers produce oscillations in bed temperature, accumulator temperature and associated components. Future versions of the program should incorporate control schemes reflecting actual controllers.

SAMPLE OUTPUT

Several simulations were made using the AT&T 3B1 version of the model, each time changing a single parameter. The model seems most sensitive to pump speed and reference temperature. The following set of graphs shows the model running with the listed parameters:

Mass flow rate of air into model	= 0.2 kg/s
H_2O concentration into model	= 0.01 kg H_2O/kg air
CO_2 concentration into model	= 0.001 kg CO_2/kg air
Temperature of air into model	= 300 K
Angular velocity of CO_2 pump	= 200 rad/s
Reference temperature of model	= 250 K

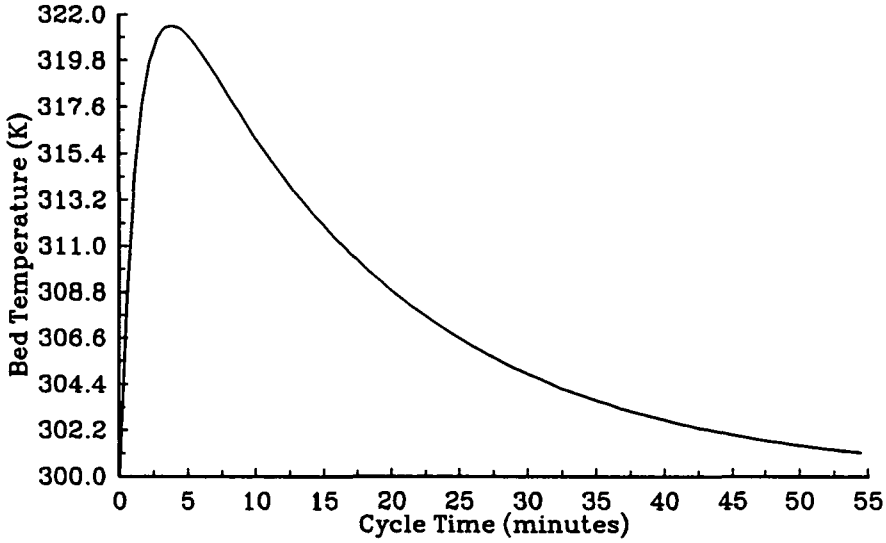

Figure 3.2-2 Temperature of Desiccant Bed

Figure 3.2-2 gives indication that a half-cycle time of 55 minutes is sufficient for the desiccant bed to stabilize at or near the ambient temperature. The temperature of the adsorbing CO_2 bed follows the same curve as the desiccant bed. Because of simplifications made to the model, the maximum adsorbing bed temperature was nearly the same as the maximum desiccant bed temperature or approximately 322 K.

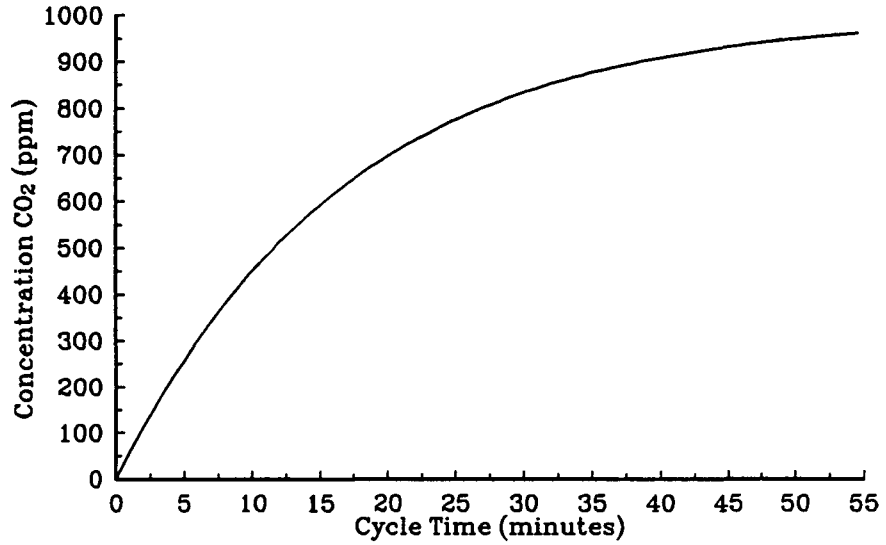

Figure 3.2-3 Concentration of CO_2 Leaving Sorbent Bed

Figure 3.2-3 shows that most of the CO_2 is adsorbed in the first third of the cycle. The graph actually shows the return air concentration, but the amount of CO_2 adsorbed would be the initial concentration minus the graph value for a given simulation time.

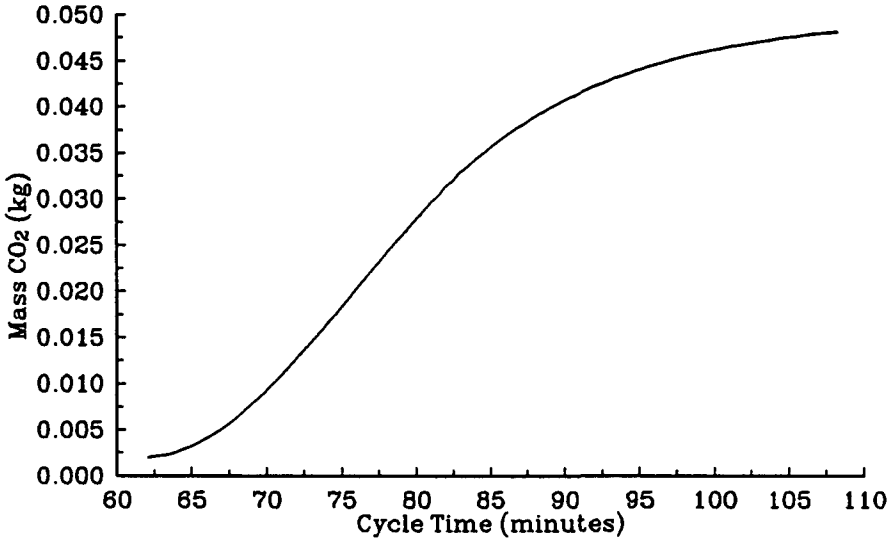

Figure 3.2-4 Mass CO_2 in Accumulator

The CO_2 Removal model assumes that the CO_2 Reduction assembly is drawing CO_2 from the storage tank at a constant rate. Figure 3.2-4 shows the accumulation of CO_2 in the tank. It was important to note that the Removal and Reduction assemblies could both place demands on the storage tank. As long as the Reduction assembly does not draw CO_2 from storage at a greater rate than the Removal assembly could produce CO_2, the system will function properly. The control system must be able to resolve conflicts such as an empty CO_2 storage tank or a Reduction assembly that is drawing too much CO_2 from the tank.

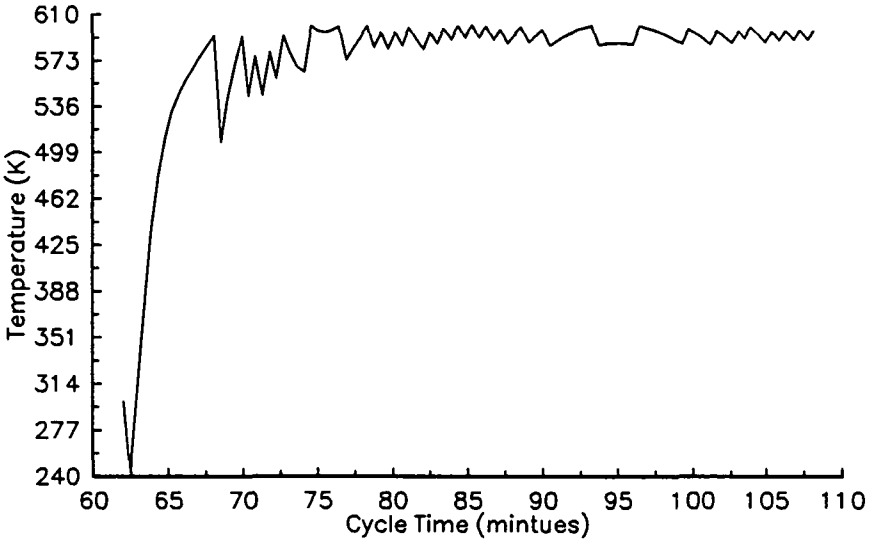

Figure 3.2-5 Temperature of Accumulator

The accumulator tank was modeled with a cooling jacket in place to remove a constant amount of heat when the tank reaches a set temperature. The jacket is controlled by a thermostat as the ECLSS control system should not bother with this low level control requirement. An interesting result of this on-off control of the tank temperature is the reflection of the graph shape in any other variable which has accumulation tank temperature as part of its controlling equation. The pump outlet pressure shown in figure 3.2-6 below illustrates this point.

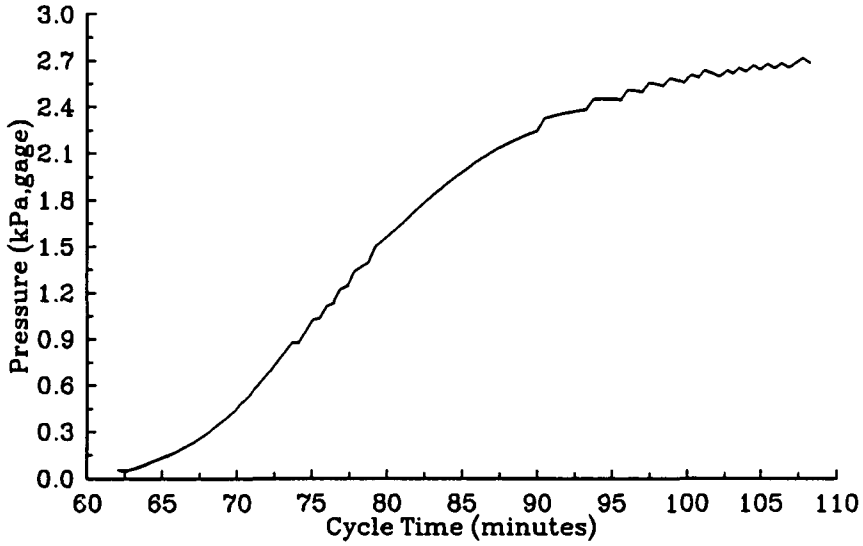

Figure 3.2-6 Outlet Pressure of CO_2 Pump

3.3 CO_2 REDUCTION ASSEMBLY MODEL

DESCRIPTION

The CO_2 Reduction Assembly of the ECLSS reduces the excess CO_2, provides oxygen for use in the cabin and hydrogen for use in the Oxygen Generation Assembly. This is accomplished with a system comprised of a Sabatier Methanian Reactor (SMR) which converts CO_2 and H_2 to methane gas (CH_4) and water vapor (H_2O), a Condenser/Separator (C/S) to remove product water, and a Carbon Formation Reactor (CFR) to break down methane into Carbon and Hydrogen. A block diagram of the CO_2 Reduction Assembly is shown in Figure 3.3-1.

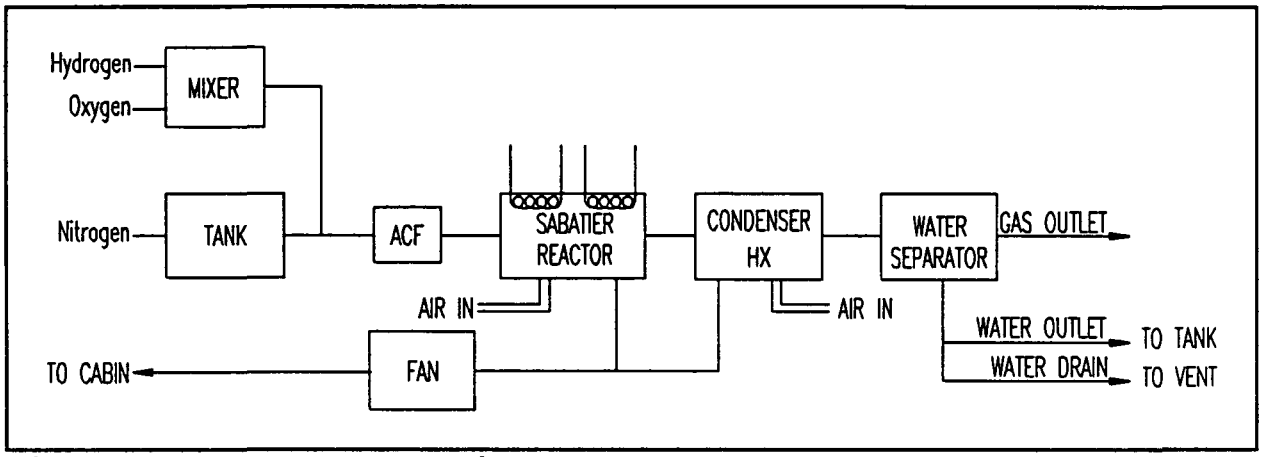

Figure 3.3-1 CO_2 Reduction Assembly

MATH MODEL

The following assumptions are made:

1. Steady-state behavior.
2. Plug flow behavior.
3. No convection.
4. No work is done by the system.
5. Heat capacity is constant over the temperature range.
6. No CH_4 or H_2O enters the gas stream.
7. The reactions occur in stoichiometric amounts.
8. Catalyst does not deactivate.

The CO_2 Reduction Assembly takes carbon dioxide and reacts it with hydrogen over a catalyst to produce methane and water. The reaction is given by the Sabatier reaction:

$$CO_2 + 4H_2 \text{ <==> } CH_4 + 2H_2O. \quad (1)$$

To find the volume of the catalyst (i.e. the reactor) necessary to carry out this reaction, a mass balance needs to be formulated.

Mass Balance

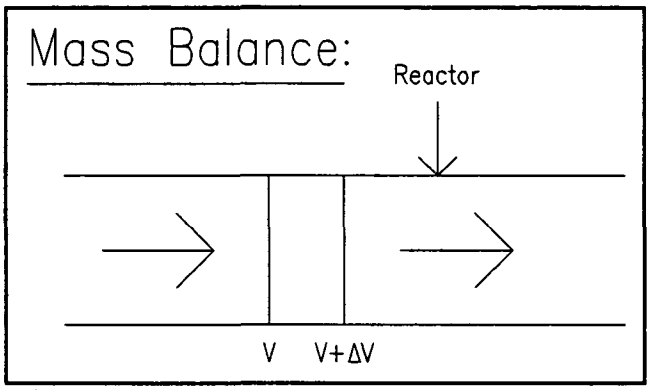

Figure 3.3-2 Mass Balance for Sabatier Reactor

In words, the mass balance is as follows:

Rate of accumulation of A in the system	=	rate at which A enters	-	rate at which A leaves	+	rate of production of A by chemical reaction

In this derivation, A will refer to CO_2.

At this point, two assumptions are made. The system exhibits 1) steady-state behavior and 2) plug flow behavior. Assuming plug flow behavior allows radial variations to be neglected. These requirements give

$$0 = F_A|_V - F_A|_{(V + \Delta V)} + r_A \Delta V, \quad (2)$$

$$\frac{F_A|_{(V + \Delta V)} - F_A|_V}{\Delta V} = \frac{dF_A}{dV} = r_A, \quad (3)$$

where
$F_A \mid_v$ = molar flow rate of CO_2 entering reactor (moles/sec),
$F_A \mid_{v+Dv}$ = molar flow rate of CO_2 leaving the reactor (moles/sec),
r_A = reaction rate (moles/cm^3),
dV = change in volume of the catalyst (liters).

The molar flow rate of CO_2, F_A, can also be represented in terms of the conversion of CO_2 as

$$F_A = F_{Ao}(1 - X_A), \tag{4}$$

where
X_A = conversion of CO_2 and F_{Ao} = inlet flow rate of CO_2.
This gives

$$dF_A = d(F_{Ao} - F_{Ao}(X_A)) = -F_{Ao}(dX_A). \tag{5}$$

Substituting this expression for dF_A back into equation (4) yields

$$dV = -\frac{F_{Ao}dX_A}{r_A}. \tag{6}$$

Integrating both sides of equation (6) gives

$$V = F_{Ao}\int_0^{X_A} \frac{dX_A}{-r_A}. \tag{7}$$

Equation (7) is the mass balance integral that gives the volume of the catalyst necessary to carry out the reaction to be determined. In order to find the volume of the reactor, a reaction rate r_A for this reaction must be determined. A reaction rate was found in a SAE technical paper [1:all]. The reaction rate used is given by

$$r_A = \frac{A}{T} e^{\left(\frac{-E_a}{RT}\right)} \left[P_{CO_2}^{0.25} P_{H_2} - \frac{P_{CH_4}^{0.25} P_{H_2O}^{0.5}}{K_{eq}} \right], \quad (8)$$

where
A = 1.112393 x 10^6

$$\frac{E_a}{R} = 8553^o K \quad (9)$$

$$K_{eq} = \exp\left[0.5033\left(\frac{5600}{T^2} + \frac{34633}{T} - 16.4\ln(T) + 0.00557T\right) + 33.165\right] atm^{-2} \quad (10)$$

with
T = Temperature
P = Partial Pressure (atm).

In order to evaluate the integral in the mass balance (7), r_A must be exclusively in terms of X_A. Thus, temperature and partial pressures must be in terms of X_A. Temperature can be represented in terms of conversion through the evaluation of an energy balance.

Energy Balance

In words the energy balance is as follows: The rate of accumulation of energy equals the rate of flow of heat from surroundings, minus the rate of work done by system on surroundings, plus the rate of energy added by mass flow in, minus the rate of energy out by mass flow out. The standard energy balance for this system can be represented by the expression

$$\left(\frac{dE}{dt}\right)_{system} = Q - W_S - F_{Ao} \sum_{i=1}^{n} \int_{T_{io}}^{T} \Theta_i C_{\pi} dT - \left(\Delta H_R^o [T_R] + \int_{T_R}^{T} \Delta C_p dT\right) F_{Ao} \cdot X_A \quad (11)$$

where
Q = Heat flow from surroundings to system (cal/sec),
W_s = Work done by system on surroundings (cal),
Θ_i = Ratio of the number of moles of species i entering initially to the number of moles of A entering initially (= F_{io}/F_{Ao}),
C_{pi} = Heat capacity of species i (cal/gmoleK).

The following expression represents the heat of reation:

$$\Delta H_R^o\,[T_R] + \int_{T_R}^{T} \Delta C_p\, dT = \textit{Heat of reaction (cal/gmole)}. \qquad (12)$$

This derivation requires the following approximations: there is no convection (Q = 0), there is no work done by the system on the surroundings (W_s = 0), the heat capacity, C_{pi}, stays constant over the temperature range in question, and the system is in steady-state.

The heat of reaction for this system was found in [1:1] to be given by

$$\Delta H_R[T] = 16.4 \cdot T - 0.00557 \cdot T^2 + \frac{112000}{T} + 34633 \frac{cal}{gmole}. \qquad (13)$$

For this system the term,

$$\sum_{i=1}^{n} \int_{T_{io}}^{T} \Theta_i\, C_{p\,i}\, dT \qquad (14)$$

expands to:

$$F_{Ao}\left(\int_{T_{Ao}}^{T} \frac{F_{Ao}}{F_{Ao}} C_{pA}\, dT + \int_{T_{Bo}}^{T} \frac{F_{Bo}}{F_{Ao}} C_{pB}\, dT + \int_{T_{Co}}^{T} \frac{F_{Co}}{F_{Ao}} C_{pC}\, dT + \int_{T_{Do}}^{T} \frac{F_{Do}}{F_{Ao}} C_{pD}\, dT \right), \qquad (15)$$

where
A = CO_2
B = H_2
C = CH_4
D = H_2O.

It is assumed that there is no CH_4 and H_2O in the entering stream, thus F_{Co} and F_{Do} are zero and the last two terms in equation (8) are also zero. The heat capacities for CO_2 and H_2 were represented by linear functions with respect to temperature. The heat capacity for CO_2 is given by [3:1]

$$C_{pA} = a + bT, \tag{16}$$

where
a = 5.316
b = 0.014285
for a temperature range of 273 K to 1800 K.

The heat capacity for H_2 is represented by

$$C_{pB} = a + bT, \tag{17}$$

where
a = 6.952
b = -0.0004576
for a temperature range of 273 K to 1800 K.

It is also assumed that the reaction occurred in stoichiometric amounts, that is,

$$\frac{F_{Bo}}{F_{Ao}} = 4. \tag{18}$$

Substituting Equations (16) and (17) into (15) and integrating from $T_{Ao} = T_{Bo} = 293$ to T yields (19):

$$F_{Ao}[(5.316 \cdot T + 0.0071425 \cdot T^2 - 2170.8) + (27.808 \cdot T - 0.0009152 \cdot T^2 - 8069.2)]. \tag{19}$$

Substituting Equations (13) and (19) into the energy balance (14), along with the assumptions made, yields:

$$-F_{Ao}[33.124 \cdot T + 0.0062273 \cdot T^2 - 10240] + \left[16.4 \cdot T - 0.00557 \cdot T^2 + \frac{112000}{T} + 34633\right] F_{Ao} \cdot X_A = 0. \tag{20}$$

Solving for X_A then gives

$$X_A = \frac{33.124 \cdot T + 0.0062273 \cdot T^2 - 10240}{16.4 \cdot T - 0.00557 \cdot T^2 + \frac{112000}{T} + 34633}. \tag{21}$$

After conducting a linear regression analysis, equation (21) reduces to:

$$T = 1088X_A + 297.59 \tag{22}$$

Equation (22) now allows the temperature to be expressed in terms of X_A in the mass balance.

The next step is to relate the partial pressures of each species in terms of X_A. This can easily be done by assuming ideal gas behavior and using the ideal gas law. The ideal gas law for component A (CO_2) is given by

$$P_A = C_A R T, \tag{23}$$

where
P_A = partial pressure of CO_2 (atm)
C_A = concentration of CO_2 (gmole/cm^3)
R = gas constant
T = temperature (°K).

The CO_2 concentration C_A can be represented by

$$C_A = \frac{N_A}{V}, \tag{24}$$

where
N_A = number of moles of CO_2
V = volume.

The number of moles N_A can then be represented by

$$N_A = N_{Ao}(1 - X_A). \tag{25}$$

Since there is a change of moles in the Sabatier reaction, the volume can best be represented by

$$V = V_o(1 + \epsilon \cdot X_A)\frac{T}{T_o}. \tag{26}$$

Epsilon, ϵ, is defined as the fraction change in volume per mole of A reacted resulting from the change in the total number of moles. Epsilon can be expressed by

$$\epsilon = \frac{-\sum_{i=1}^{n} \nu_i}{\nu_A}\frac{N_{Ao}}{N_{To}}, \tag{27}$$

where

v_i = stoichiometric coefficients which are negative for the reactant and positive for the products,

N_{Ao} = initial moles of CO_2,

N_{To} = total moles initially present.

Again the assumption that the reaction occurs in stoichiometric amounts is made so N_{Ao}/N_{To} is equal to 1/5 and ϵ becomes -0.4. Substituting Equations (25) and (26) into (24) and using ϵ = -0.4 yields:

$$C_A = \frac{N_{Ao}(1 - X_A)T_o}{V_o(1 - 0.4X_A)T} \tag{28}$$

Substituting Equation (28) into (23) and noting that N_{Ao}/V_o is equal to C_{Ao} yields

$$P_A = C_{Ao}\left(\frac{1 - X_A}{1 - 0.4X_A}\right)\frac{T_o}{T}RT. \tag{29}$$

Noting that $C_{Ao}RT_o$ is equal to P_{Ao} and also noting that the temperature cancels out, Equation (29) becomes

$$P_A = P_{CO_2} = P_{Ao}\frac{1 - X_A}{1 - 0.4X_A}. \tag{30}$$

With the assumption of reactions occurring in stoichiometric amounts, P_{Ao} is equal to 0.2. Thus, the partial pressure of A (CO_2) is now a function of conversion (X_A). Following a similar derivation, the partial pressures for the remaining species can be represented as follows:

$$P_B = P_{H_2} = 0.2\left(\frac{4 - 4X_A}{1 - 0.4X_A}\right), \tag{31}$$

$$P_C = P_{CH_4} = 0.2\left(\frac{X_A}{1 - 0.4X_A}\right), \tag{32}$$

$$P_D = P_{H_2O} = 0.2\left(\frac{2X_A}{1 - 0.4X_A}\right). \tag{33}$$

Since all partial pressures and temperatures for all the species are in terms of conversion, the mass balance integral can now be integrated where r_A is exclusively in terms of conversion.

In equilibrium reactions, a maximum conversion can be achieved. This maximum conversion becomes the upper limit when evaluating the mass balance integral. The maximum conversion can be determined by setting the reaction rate, r_A, equal to zero and solving for X_A. Doing this yields a maximum conversion of approximately 0.52. This maximum conversion corresponds to a reactor operation temperature of approximately 860 K. Thus, the volume of the catalyst can be found by:

$$V = F_{Ao} \int_0^{0.52} \frac{dX_A}{-r_A}, \tag{34}$$

where
r_A = equation (8)
temp. = given in terms of conversion by Equation (22)
pp. = given in terms of conversion by Equations (30) through (33).

As can be seen by the complexity of the equations, this integral is difficult to evaluate analytically. Therefore a different method was employed. To evaluate this integral, a plot of $1/-r_A$ vs. X_A was made and the area under the curve measured from $X_A = 0$ to 0.52. Using a software package, this area was found to be 1.652×10^7. Therefore the volume is

$$V = \left(1.652 \times 10^7 \frac{sec \cdot cm^3}{gmoles}\right) F_{Ao}. \tag{35}$$

Equation (35) now allows us to determine the catalyst volume necessary to carry out the reaction for a specified inlet CO_2 rate. Below is a table representing the reactor volume necessary for various man-loadings.

Table 3.3-1 Reactor Volumes for Various Man-loadings

Man Loading	Inlet CO_2 Flow Rate	Catalyst Volume
3 - man	7×10^{-4} mol/sec	3.1 gallons
5 - man	1.3×10^{-3} mol/sec	5.7 gallons
8 - man	2.1×10^{-3} mol/sec	9.2 gallons
10 - man	2.8×10^{-3} mol/sec	11.3 gallons

* Numbers obtained from SAE technical paper, # 840936.

Throughout this derivation there was one significant assumption that was made but not incorporated in the mass balance. This assumption was that the catalyst did not deactivate (i.e. the catalyst remained active indefinitely).

Based on this analysis, a reactor size of 11.3 gallons is recommended operating at a temperature of 860 K. A size of 11.3 gallons was chosen because any man-loading below 10-man could easily be handled by this reactor.

At this point in the modeling, a computer program was developed that would simulate the Sabatier reactor and it is listed in Appendix 3b. Output from this program correlated with the previously recommended reactor size. For a set reactor volume, any flow rate of CO_2 equal to or below the flow rate that corresponds to maximum conversion is fine. Yet, for flow rates that are just slightly larger, conversion drops exponentially. One item to note is that small changes in flow rates in moles/sec correspond to fairly big changes in grams/sec. Never-the-less, a reactor size corresponding to a 10-man loading will be sufficient for most worst-case scenario's.

MODELING REVIEW

- There are no controls in our model.
- We did not consider transient responses in our model. This is because the Sabatier reactor has heaters to heat the catalyst at the entrance. We assumed that the reactor would be heated up to the temperature relating to maximum conversion before any species entered the reactor.
- Obviously, our model is only a part of the overall WRM. We decided to assume that the CO_2 and H_2 coming into the Sabatier was actually from an external tank and not from another sub-assembly. Therefore, our model does not "communicate" with anything else in the WRM.
- As far as the expert system is concerned, they can control our whole model since the only output that is vital is conversion, which relates to the production of H_2O forthe OGA. Depending on the needs for water for the OGA, or conversely the amount of CO_2 that the CO_2 Removal sub-assembly is producing, the expert system can vary conversion to account for different flow rates of CO_2 in or of H_2O out.

3.4 OXYGEN GENERATION ASSEMBLY MODEL

The Oxygen Generation Assembly (OGA) of the AR provides the daily requirements of oxygen to the cabin compartment of the space station. It accomplishes this by using electrolysis to convert water into its two components, oxygen and hydrogen. This water is supplied to the OGA from the CO_2 reduction sub-assembly and the hygiene water supply. A by-product of this reaction is hydrogen gas which is used in the CO_2 reduction sub-assembly of the ECLSS.

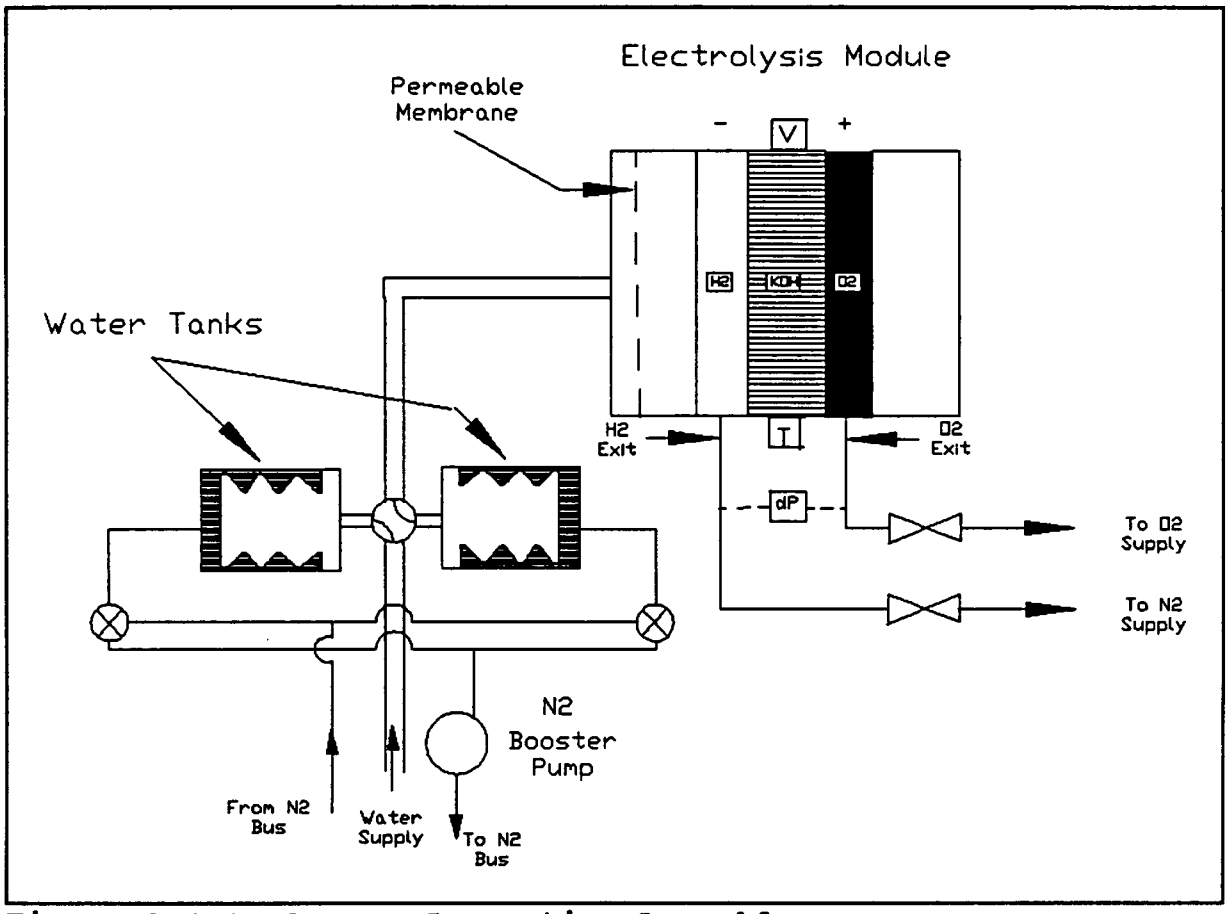

Figure 3.4-1 Oxygen Generation Assembly

The OGA assembly consists of two main components, a water storage and supply system and an electrolysis module. The storage and supply system of the OGA consists of two water storage tanks pressurized by nitrogen. These tanks supply water to the electrolysis module in alternating cycles. As one tank drains to the electrolysis module, the other tank fills with water. This is accomplished by means of a 4-way, 3-position directional control valve (DCV). See Figure 3.4-1. The Differential

pressure across the bellows of the water tanks force the water to the electrolysis module membrane. When the draining tank reaches a low level set-point, detected by a photo-cell indicator, the DCV switches to drain the other tank and allows this tank to refill. This cycle continues throughout the normal operational mode of the oxygen generation process.

When the water reaches the electrolysis module, it encounters a permeable membrane that separates the water supply from the water vapor region of the 18 electrolysis cells. Upon reaching this permeable membrane of the electrolysis module, the water diffuses across it and forms water vapor on the other side. The electrolysis module then produces H_2 from the water vapor at the cathode and passes OH^- ion through a KOH matrix. O_2 is then formed on the other side of this matrix at the anode of the module. Both gases then leave the electrolysis unit and pass through the pressure control assembly. See Figure 3.4-2. This pressure control assembly, consisting of pressure control valves on both the O_2 and H_2 outgoing lines, regulates the differential pressure of the two departing gases.

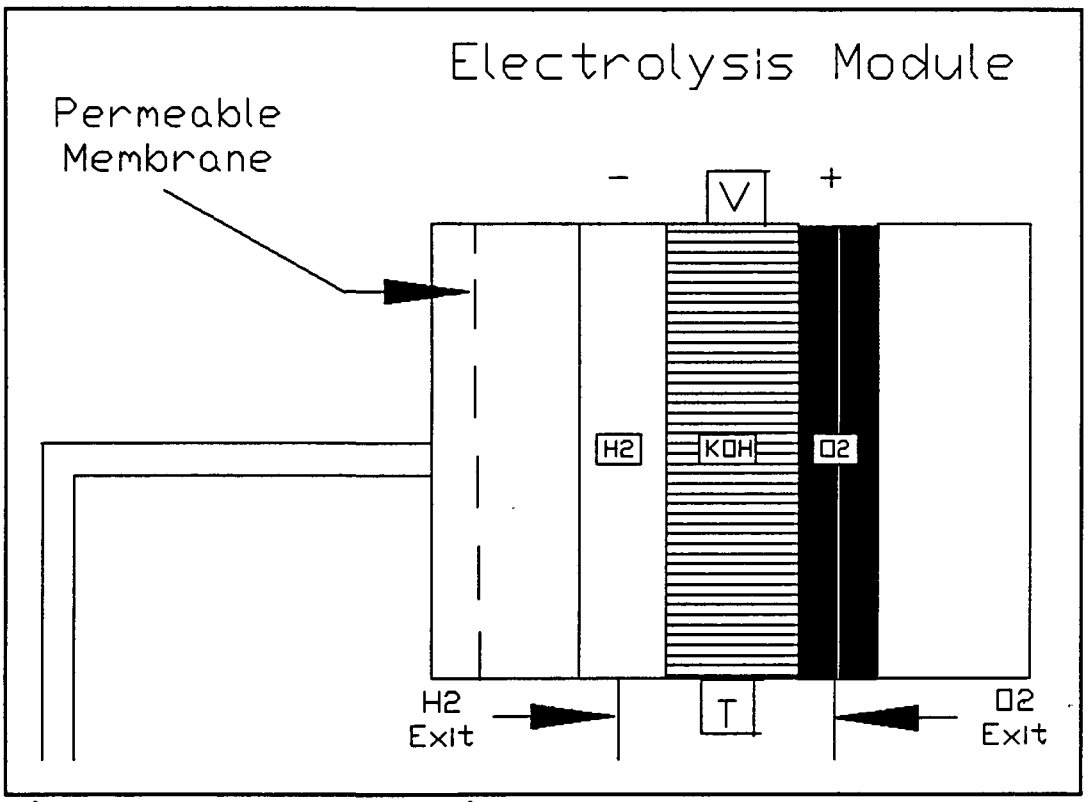

Figure 3.4-2 Electrolysis Module

The reation of interest at the cathode is

$$4H_2O+4e^-=2H_2+4OH^-,$$

which for the anode one obtains

$$4OH^-=O_2+2H_2O+4e^-.$$

MATH MODELING

The modeling of the OGA has been separated into two components: the water storage tanks and the electrolysis module. Each of these have their separate governing equations which are covered in their respective areas.

Water Tank

The mathematical modeling for the water tanks can be done by modeling the water portion, and the nitrogen portion of each tank.

Assumptions

1. Water has constant physical properties.
2. Nitrogen has a constant heat capacity.
3. The water flow into the storage tank can be considered turbulent since the tank fills in a four to five minute period through a small diameter pipe.
4. The water flow out of the storage tank can be considered laminar since it flows out over a three hour period through a similar sized pipe.
5. The nitrogen flow into and out of the storage tank can be considered as turbulent during the fill mode; and is laminar during the drain mode.
6. Nitrogen follows the ideal gas law.
7. All heat transfer by conduction and radiation will be considered negligible because of minimal temperature differences in the system (i.e. no heat loss).

Math Modeling of Water Portion of Tank

When modeling the water portion of the water tank, the dependent variables of interest are the change in the volume and temperature, and the inlet and outlet water flows.

A material balance can be written to relate the change in volume of the tank with respect to time. This material balance can be written as:

Rate of Mass Accumulation = [Rate of Mass]$_{in}$ - [Rate of Mass]$_{out}$.

Symbolically this becomes

$$\frac{\rho_{H_2O}\partial V_{T1}}{\partial t} = W_{H_2O,1} - W_{H_2O,2}, \tag{1}$$

where
V_{T1} = volume of water contained in the storage tank at any given time (m^3),
$W_{H2O,1}$= mass flow rate of water in (Kg/sec),
$W_{H2O,2}$= mass flow rate of water out (Kg/sec),
ρ_{H2O} = density of water (Kg/m^3).

During the tank fill mode, the $W_{H2O,2}$ term in (1) is equal to zero. In the drain mode, the $W_{H2O,1}$ term is equal to zero.

Since the tank takes four to five minutes to fill, it is a good assumption that the water flowing into the tank is turbulent because the Reynolds number (Re) is approximately 40,000. In deriving an equation for the turbulent flow of water through a tube, a good starting point is the Blassius Formula for friction factors. The Blassius Formula is as follows:

$$f = \frac{0.0791}{Re^{1/4}} \tag{2}$$

for, Reynolds numbers in the range $2.1(10)^3 < Re < 10^5$,
where
Re = the Reynolds number given by Re= D <v> ρ_{H2O}/μ_{H2O},
D = the tube diameter (m),
<v> = the average velocity of the water (m/sec),
μ_{H2O} = the viscosity of water (Kg/m sec).

When the definition for the Reynolds number is substituted into the Blassius Formula, Equation (2) becomes

$$f = \frac{0.0791}{D\langle v\rangle \dfrac{\rho_{H_2O}}{\mu_{H_2O}}}. \tag{3}$$

The friction factor can also be written as

$$f = \frac{D(P_{source} - P_{V1})}{2L\rho_{H_2O}\langle v\rangle^2}, \tag{4}$$

where
P_{source} = the pressure of the external water source (N/m2),
P_{V1} = the pressure in the water side of the tank (N/m2),
L = the tube length (m).

Setting Equations (3) and (4) equal to each other gives

$$f=\frac{0.0791}{D\langle v\rangle\frac{\rho_{H_2O}}{\mu_{H_2O}}}=\frac{D(P_{source}-P_{V1})}{2L\rho_{H_2O}\langle v\rangle^2}. \tag{5}$$

Multiplying each side by the tube cross sectional area($A=\pi D^2/4$) and then rearranging gives

$$\rho_{H_2O}\langle v\rangle A=\frac{(D^{5/4}\rho_{H_2O}(P_{source}-P_{V1}))^{4/7}A}{(0.1582L\mu_{H_2O}{}^{1/4})^{4/7}}. \tag{6}$$

Substituting Equation (6) into Equation (1) gives

$$\frac{\rho_{H_2O}\partial V_{T1}}{\partial t}=\frac{(D^{5/4}\rho_{H_2O}(P_{source}-P_{V1}))^{4/7}A}{(0.1582L\mu_{H_2O}{}^{1/4})^{4/7}} \tag{7}$$

since

$$W_{H_2O}=\rho_{H_2O}\langle v\rangle A.$$

With the equation derived for the change in volume of the water tank during the fill mode with respect to time, an equation that models volume change when the water is flowing out of the tank is needed. The water in the tank flows out over a period of three hours, so laminar flow is assumed. The laminar flow of a fluid through a tube is given by the Hagen-Poiseuille law as

$$Q=\frac{\pi(P_{V1}-P_{elec})r^4}{8\mu_{H_2O}L}, \tag{8}$$

where
P_{elec} = the pressure in the electrolysis module.

Altering Equation (8) can be changed into mass flow rate by multiplying by ρ_{H2O} to obtain

$$W_{H_2O,2}=\frac{\pi(P_{V1}-P_{elec})r^4\rho_{H_2O}}{8\mu_{H_2O}L} \tag{9}$$

Substituting Eq. (9) into Eq. (7), gives the change in volume of the water portion of the water tank during the drain mode as

$$\frac{\rho_{H_2O}\partial V_{T1}}{\partial t} = -\frac{\pi (P_{V1}-P_{elec})\, r^4 \rho_{H_2O}}{8\mu_{H_2O}L}. \qquad (10)$$

With these equations that model the change in volume of the tank during the fill and drain modes, an equation needs to be derived to relate the change in temperature of the water with respect to time. This can be done by writing an energy balance on the system. The energy balance for the water portion of the water tank is given by:

[Rate of Energy] = [Rate of Energy]$_{in}$ - [Rate of Energy]$_{out}$ -[Net Rate of Heat Loss to Surroundings].

Symbolically, this becomes:

$$\rho_{H_2O} C_{p_{H_2O}} \frac{\partial (V_{T1}(T_{H_2O}-T_{ref}))}{\partial t} = W_{H_2O,1} C_{p_{H_2O}} (T_{H_2O,1}-T_{ref}) - W_{H_2O,2} C_{p_{H_2O}} (T_{H_2O,2}-T_{ref}) - 0, \qquad (11)$$

where
C_{pH2O} = heat capacity of Water (KJ/Kg K),
$T_{H2O,1}$ = temperature of water into the compartment (K),
$T_{H2O,2}$ = temperature of water out of the compartment (K).

Note that the net rate of heat loss to the surroundings equals zero. Assumptions were made that the system was well insulated and any heat loss to the surroundings is negligible.

Equations for the mass flow rate of water in and out of the tank have already been derived. By manipulating the left hand side of Eq. (11) a relationship can be drawn between the rate of change in temperature and volume given by

$$C_{p_{H_2O}} \rho_{H_2O} (T_{H_2O}-T_{ref}) \frac{\partial V_{T1}}{\partial t} + C_{p_{H_2O}} \rho_{H_2O} V_{T1} \frac{\partial T}{\partial t} \qquad (12)$$

When solved for $\delta T/\delta t$, Eq. (12) relates the change in temperature of the water in the tank with time. This term should be negligible if the temperature of the water and the temperature of the surroundings are approximately the same. Assuming this, the only variables of interest in the system become mass flow rate of the water and the rate of change of volume in the water tank.

Nitrogen Portion of the Water Tank

When modeling the Nitrogen portion of the water tank, the

variables of interest are as follows:

$W_{N2,1}$ = mass flow rate of Nitrogen in (Kg/sec),
$W_{N2,2}$ = mass flow rate of Nitrogen out (Kg/sec),
$T_{N2,1}$ = temperature of Nitrogen in (K),
$T_{N2,2}$ = temperature of Nitrogen out (K),
$P_{N2,1}$ = pressure of the Nitrogen in the Water tank (N/m^2),
$P_{N2,2}$ = pressure of the Nitrogen in the supply tank (N/m^2),
V_{T2} = volume of the Nitrogen side of the tank (m^3).

These variables can be related by differentiating the Ideal Gas Law with respect to time to give

$$\rho_{N_2}\frac{\partial V_{T2}}{\partial t}+V_{T2}\frac{\partial \rho_{N_2}}{\partial t}=RT_{N_2}\frac{\partial n_{N_2}}{\partial t}+n_{N_2}R\frac{\partial T_{N_2}}{\partial t}, \quad (13)$$

where
$\partial V_{T2}/\partial t$ equals $-\partial V_{T1}/\partial t$ since

$$V_{tank} = V_{T1} + V_{T2} \quad (14)$$

or:

$$\frac{\partial V_{tank}}{\partial t}=0=\frac{\partial V_{T1}}{\partial t}+\frac{\partial V_{T2}}{\partial t}. \quad (15)$$

The $\delta n_{N2}/\delta_t$ term can be determined by making a mole balance of the Nitrogen compartment given by

[Accumulation of Moles] = [Rate of Moles in]-[Rate of Moles out]

Symbolically this can be written as

$$\frac{\partial n_{N2}}{\partial t}=(W_{N_2,1}-W_{N_2,2})\,(1/M), \quad (16)$$

where
n_{N2} = moles of Nitrogen,
M = molecular weight of Nitrogen.

When the water side of the tank is filling, the Nitrogen side of the tank is draining. This process occurs over a four to five minute period. Assuming turbulent flow for the nitrogen in this case is a safe assumption. By analogy, Eq. (6) applies to the flow rate of nitrogen leaving the tank given by

$$W_{N_2,2}=\frac{(D^{5/4}\rho_{N_2}(P_{N2,1}-P_{N2,2}))^{4/7}A}{(0.1582L\mu_{N_2}{}^{1/4})^{4/7}}, \quad (17)$$

where

ρ_{N2} = density of the nitrogen at the given pressure (Kg/m^3),
μ_{N2} = viscosity of the nitrogen (Kg/m sec).

The flow of nitrogen into the tank can be considered laminar due to the extended period of time of the nitrogen fill. Similar to the case above, a simple manipulation of the equivalent water side produces the equation for nitrogen flow into the tank of

$$W_{N_2,1}=\frac{\pi\,(P_{N_2}2-P_{N_2}1)\,r^4\rho_{N_2}}{8\mu_{N_2}L}, \tag{18}$$

Knowing these two flow rates, and assuming that the heat loss to the surroundings is negligible, an equation similar to Eq. (11) can be derived from the energy balance of the nitrogen portion of the tank to give

$$\rho_{N_2}C_{p_{N_2}}\frac{\partial(V_{T2}(T_{N_2}-T_{ref}))}{\partial t}=W_{N_2,1}C_{p_{N_2}}(T_{N_2,1}-T_{ref})-W_{N_2,2}C_{p_{N_2}}(T_{N_2,2}-T_{ref}), \tag{19}$$

where
c_{pN2} = Heat capacity of Nitrogen (KJ/Kg K)

Expanding the left-hand side of Eq. (19) gives

$$C_{p_{N_2}}\rho_{N_2}(T_{N_2}-T_{ref})\frac{\partial V_{T2}}{\partial t}+C_{p_{N_2}}V_{T2}\rho_{N_2}\frac{\partial T_{N_2}}{\partial t}. \tag{20}$$

However, knowing that

$$\rho_{N_2}\frac{\partial V_{T2}}{\partial t}=\frac{\partial n_{N_2}}{\partial t}, \tag{21}$$

the left side of Eq. (19) reduces to

$$W_{N_2,1}-W_{N_2,2}+C_{p_{N2}}V_{T2}\rho_{N_2}\frac{\partial T_{N_2}}{\partial t}. \tag{22}$$

Solving for $\partial T_{N2}/\partial t$ give

$$\frac{\partial T_{N_2}}{\partial t}=\frac{W_{N_2,1}(C_{p_{N2}}(T_{N_2,1}-T_{ref})-1)+W_{N_2,2}(1-C_{p_{N2}}(T_{N_2,2}-T_{ref}))}{C_{p_{N2}}V_{T2}\rho_{N_2}}. \tag{23}$$

With equations for $\partial V_{T2}/\partial t$, $\partial n_{N2}/\partial t$, and $\partial T_{N2}/\partial t$, an equation for $\partial P_{N2}/\partial t$ can be derived by substitution into Eq. (13). This equation will yield the temperature, pressure, moles of nitrogen and the volume of the tank at any time. The temperature term is constant because of the negligible difference in the nitrogen and ambient temperatures.

Electrolysis Module

The assumptions for this model are as follows:

1. Water has constant physical properties.
2. Water is an incompressible fluid.
3. The heat capacities of the gases are constant.
4. There is no accumulation of gases in the electrolysis module.
5. "The non-electrolytic 'parasitic' energy used by the electrolyses unit per kilogram of water converted to gas" is negligible.
6. The amount of water vapor leaving in the product gases is negligible.

Initially, the electrolysis module will be modeled as a single cell. Modeling could be accomplished by dividing the module into a feedwater compartment and the actual electrolysis compartment. Variables of interest in modeling the water compartment are:

$W_{H2O,1}$ = mass flow rate of water into module (Kg/sec),
$W_{H2O,2}$ = mass flow rate of water out (Kg/sec),
ρ_{H2O} = density of water (Kg/m3),
V_{E1} = volume of water contained in the water compartment of the electrolysis module (m^3).

A material balance on the feedwater module produces the following equation:

$$\frac{\rho_{H_2O}\partial V_{E1}}{\partial t} = W_{H_2O,1} - W_{H_2O,2}. \qquad (24)$$

The volume of the water compartment in the electrolysis module remains constant at steady state, therefore the above equation can be reduced to

$$W_{H2O,1} = W_{H2O,2}. \qquad (25)$$

The flow out of the water tank is considered laminar, so it follows that the flow into the water compartment of the electrolysis module is laminar as well. This allows Eq. (9) to be used for the flow rate of water into the feedwater compartment which takes the form

$$W_{H_2O,1}=\frac{\pi\,(P_{tank}-P_{elec})\,r^4\rho_{H_2O}}{8\mu_{H_2O}L}.$$

At steady state, the vapor rate ($W_{H2O,2}$) is equal to the rate of water out of the feedwater module. The flow rate out could also be estimated by deriving an expression for the diffusion of a vapor through a porous membrane. However, very little information exists about the membrane and the above equations provide a better representation.

Writing an energy balance for the water compartment produces and equation similar to those used for the water tank. However, the equation for the water compartment contains a term for the vaporization of water. The energy balance is written as follows:

$$\rho_{H_2O}C_{p_{H_2O}}V_{E1}\frac{\partial T_{H_2O}}{\partial t}=W_{H_2O,1}C_{p_{H_2O}}(T_{H_2O,1}-T_{ref})-W_{H_2O,2}C_{p_{H_2O}}(T_{H_2O,2}-T_{ref})$$

$$-Q_{cond,E1}-Q_{rad,E1}+(-h)\,W_{H_2O,2}, \qquad (26)$$

where
$T_{H2O,1}$ = temperature of water into the compartment(K),
$T_{H2O,2}$ = temperature of water out of the compartment(K),
$Q_{cond,E1}$ = heat loss by conduction(KJ/sec.),
$Q_{rad,E1}$ = heat loss by radiation(KJ/sec.),
-h = heat of vaporization of water (KJ/Kg).

The equations for flow rate in and out of the water compartment can be replaced by Eq. (27). The heat loss by conduction through the tank walls can be expressed as

$$Q_{cond,E1}=A_{E1}\left[\frac{1}{r_{Eo}h_{H_2O}}+\frac{1nr_{E1}/r_{Eo}}{K_T}+\frac{1}{r_{E1}h_{surr}}\right]^{-1}(T_{H_2O}-T_{surr}), \qquad (27)$$

where
r_{Eo} = inside radius of the tank (m),
r_{E1} = outside radius of the tank (m),
A_{E1} = surface area of the water compartment, excluding side with membrane (m^2).

Heat loss by radiation from the tank can be written as follows:

$$Q_{rad,E1}=A_{E1}\epsilon_{E1}\sigma\,(T^4_{H_2O}-T^4_{surr}), \qquad (28)$$

where
ϵ_{E1} = emissivity of the outer compartment walls.

The temperature in the feed water compartment is known at any particular time. Also, the flow rate into and out of this compartment can be calculated.

When looking at the electrolysis compartment of the module, there are three variables that need to be considered, i.e., the production of hydrogen, oxygen, and the change in temperature in the module. It becomes necessary at this point to work on a mole basis. Writing a mole balance for hydrogen and oxygen yields the form

[moles accumulated]=[moles in] - [moles out] + [moles produced].

By dividing Eq. (26) by the molecular weight of water, the number of moles entering the electrolysis compartment are obtained. This number of moles equals the number of moles of H_2 and half the number of moles of O_2 leaving the system. The equations for these two variables can be stated as

$$q_{H_2O,1}=\frac{\pi\,(P_{tank}-P_{elec})\,r^4\rho_{H_2O}}{8\,(18)\,\mu_{H_2O}L}, \qquad (29)$$

where
$q_{H2O,1}$ = The molar flow-rate of water into the electrolysis module (moles/sec).

Knowing that the moles of hydrogen and oxygen produced are dependent upon the electrolysis rate given by

$$q=I/F*z, \qquad (30)$$

where
q = moles of gas produced (mols/sec),
I = current supplied to the electrodes (col/sec),
F = Faraday's constant (9.6487×10^4),
z = equivalents/mole(H2=1,O2=2),

the moles of hydrogen produced can be written as

$$q_{H_2,2}=\frac{I(col/\sec)}{(9.6487\times10^4 col/eqv.)\,(2 eqv.H_2/mole)}. \qquad (31)$$

Here hydrogen has 2 equivalents per mole. The stoichiometric relationship of the dissociation of water into hydrogen and oxygen is

$$H_2O \Rightarrow H_2 + \tfrac{1}{2}O_2. \qquad (32)$$

Therefore the number of moles of oxygen produced equals one-half that of the amount of hydrogen produced. One can say that the rate of production of moles of oxygen equals one-half that of hydrogen which is written as

$$q_{O2,out} = \tfrac{1}{2} q_{H2,out}. \qquad (33)$$

With this relationship, the flow rate of water into the electrolysis module for a given current can be calculated, assuming the accumulation of moles in the module equals zero. By knowing the flow rate of the water, a pressure drop from the water tank to the electrolysis can be also found.

An energy balance will produce the temperature changes in the electrolysis module. This energy balance is similar to the energy balances calculated earlier. However, it also accounts for the energy supplied to the system through the electrodes which operate at approximately 137 °F. The energy balance can be expressed as

$$\rho_{avg} C_{p_{avg}} V_{E1} \frac{\partial T_{H_2O}}{\partial t} = W_{H_2O,1} C_{p_{H_2O}} (T_{H_2O,1} - T_{ref}) - q_{H_2,2}(2)\, C_{p_{H_2}} (T_{H_2} - T_{ref})$$

$$- q_{O_2 2}(32)\, C_{p_{O_2}} (T_{O_2} - T_{ref}) - Q_{cond,E1} - Q_{rad,E1} + E_{elec} W_{H_2O,1} + (H_{rxn}), \qquad (34)$$

where:

$T_{H2O,1}$ = temperature of water into the compartment(K),
$T_{H2,2}$ = temperature of hydrogen out of the module(K),
$T_{O2,2}$ = temperature of oxygen out of the module(K),
ρ_{avg} = average density of the three gases (Kg/m^3),
C_{pavg} = average heat capacity of the gas (KJ/Kg K),
C_{pO2} = average heat capacity of the oxygen (KJ/Kg K),
C_{pH2} = average heat capacity of the H_2 (KJ/Kg K),
$Q_{cond,E2}$ = heat loss by conduction(KJ/sec),
$Q_{rad,E2}$ = heat loss by radiation(KJ/sec),
E_{elec} = electrical energy added to the system by the electrodes (KJ/Kg),
H_{rxn} = the chemical energy added to the system due to the heat of the reaction.

The heat loss through conduction can be expressed as follows:

$$Q_{cond,E2} = A_{E2} \left[\frac{1}{r_{E2} h_{H_2}} + \frac{1 n r_{E3}/r_{E2}}{K_T} + \frac{1}{r_{E3} h_{surr}} \right]^{-1} (T_{H_2} - T_{surr}), \qquad (35)$$

where

r_{E2} = inside radius of the electrolysis module (m),
r_{E3} = outside radius of the electrolysis module (m),
A_{E2} = surface are of electrolysis module, excluding side with the membrane (m^2).

Heat loss by radiation from the tank can be written as

$$Q_{rad,E2}=A_{E2}\epsilon_{E2}\sigma\,(T_{H_2}^4-T_{surr}^4)\,, \tag{36}$$

where
ϵ_{E2} = emissivity of the electrolysis module.

The energy added to the system by the electrodes can be represented by the following equation:

$$E_{elec}=\frac{((2973.\frac{V}{I})+E_p)\,M_w}{(M_W+M_V)\,2.7878\times10^{-4}}\,, \tag{37}$$

where
E_p = "the non-electrolytic 'parasitic' energy used by the electrolyses unit per kilogram of water converted to gas,
V = cell Voltage,
I = electrolyses unit current efficiency,
M_w = mass of water electrolyzed to gas (Kg),
M_v = mass of water leaving the system (Kg).

The value of E_p is less than 0.05% of the total energy, and is therefore negligible. The M_v term is a function of temperature and pressure and is essentially constant at the temperature and pressure of interest. This allows a safe assumption that a negligible amount of water vapor leaves the system with the product gases. The previous equation can then be simplified to

$$E_{elec} = 10702\ V/I. \tag{38}$$

The chemical energy removed from the system due to the heat of reaction can be written as

$$H_{rxn}=M\sum\,(H^o-H_o^o+\Delta H_{fo}^o)_{products}-M\sum\,(H^o-H_o^o+\Delta H_{fo}^o)_{reactants}, \tag{39}$$

where
$H^o - H^o_o$ = the enthalpy of the element at 298K and the system temperature, respectively (KJ/mole),
$\Delta H^o_{f\,o}$ = the heat of formation of the compound from its elements.

Oxygen Generation Assembly

The rate of production of O_2 in the OGA is directly related to the electrolysis rate, which in turn is directly dependent upon the amount of current supplied to the electrolysis module. This

assumption is the basis for the modeling of the entire OGA.

The rate of electrolysis depends upon the following relationship:

$$q=\frac{I}{Fz}, \tag{40}$$

where
q = rate of gas produced (moles/sec),
I = current supply to the electrodes (col/sec),
F = Faraday's Constant (9.6487×10^4 (col/eqv)),
z = equivalents/mole (H_2 = 1, O_2 = 2).

For example, the moles of hydrogen produced can be written as

$$q_{H2,out}=\frac{I(col/\text{sec})}{(9.6487x10^4(col/eqv))\left(\frac{2eqv_{H2}}{mole}\right)}. \tag{41}$$

According to the stoichiometric relationship between H_2 and O_2 is

$$H_2O=>H_2+\frac{1}{2}O_2. \tag{42}$$

Therefore the moles of O_2 produced equals half the moles of H_2 produced.

MODELING TECHNIQUES

The above assumptions that the rate of O_2 and H_2 productions do not take into account any temperature or pressure effects on the electrolysis rates. After studying the complexities of the pressure and temperature effects and seeing how they related to the production results, it was determined that for simplicity, they would be ignored. In essence, this assumption creates a simple black box with known inputs and outputs for the electrolysis module. It also provides a simple linear relationship between the current and the production rate of O_2. The model eliminates any time based differential equations that would otherwise govern the output of the system.

Based on the above assumptions and knowing the requirements for O_2 production, sizing of the water storage and supply side of the OGA can be done. During normal (4-man) operational mode, the OGA needs to produce 9.081b of O_2/day. Knowing this O_2 is generated from 10.22 lb of H_2O/day, a rough tank size was determined to hold 1 lb of water. During the emergency (8-man) operational mode, the system needs to generate 15.6 lb of O_2/day [3:1-11]. Since the water tank fills in only 3 min. and drains in over 70 min., this tank size meets water capacity needs and space requirements.

4.0 EXPERT SYSTEM

4.1 INTRODUCTION

Monitoring and maintaining the Environmental Control and Life Support System (ECLSS) on the Space Station Freedom (SSF) requires the effort and expertise of many supervisors, engineers, and technicians. If the station personnel is to provide this expertise, then they will be less able to provide the expertise and labor needed to perform their mission. However, if ECLSS can intelligently-monitor itself and perform self-diagnosis, then the crew's limited resources can be better applied to the mission. This year, the design team investigated the application of expert systems to provide for intelligent automated control and monitoring of ECLSS. The prototype expert system the teams developed is discussed in the following sections.

Besides the ultimate effect of providing automated life support expertise for the SSF crew, an expert system controller provides special benefits to the system designer. The use of an expert system shell allows developers to program the system's knowledge base using a high-level specification, object oriented, or rule based style language. The format of the knowledge base can be customized to suit the knowledge particular to a given application. Furthermore, expert systems can be developed with incomplete and inexact information. This feature permits expert systems to function in poorly characterized applications such as non-linear modeling and natural language interpretation. Because an expert system can operate on incomplete information, it is easier to produce and improve working prototypes. Improving the prototype is a matter of adding additional and more specific knowledge.

The final goal of the expert system design teams is to provide an intellect controller for the whole of ECLSS. This year, the team developed a prototype expert system for controlling the Atmosphere Revitalization called ARES. As an expert system, ARES has two major components, a knowledge base and an inference engine. The three design teams developed knowledge bases for controlling and monitoring the CO_2 Reduction, CO_2 Removal, and Oxygen Generation subassemblies. Additionally, one of these teams developed the inference engine to drive the expert system. A more detailed description of ARES follows.

4.2 ATMOSPHERE REVITALIZATION EXPERT SYSTEM (ARES)

The function of ARES is to intelligently monitor and control the Atmosphere Revitalization (AR) assembly. Because this is a hardware control application, ARES must receive sensor signals from the assembly, process these input signals, and provide control signals to the assembly. Because ARES is an expert system, the sensor signals are processed after the manner of an expert. That is, as an expert, ARES translates the sensor values into terms compatible with its knowledge base, applies its knowledge to develop responses and diagnostics as an expert would, and implements its responses by providing the proper control signals to the assembly. An explanation facility enables ARES to justify its conclusions to the crew.

As an expert system, ARES has two parts, an inference engine shell and a knowledge base. The inference engine of an expert system is a shell that separates the general knowledge and support software from the application specific knowledge. It is the inference engine that does the "thinking" for the expert system. The knowledge base of an expert system contains the knowledge that represents the expertise for the given application.

ARES's inference engine supports and interprets a near-english, rule-based knowledge base. ARES "thinks" by deductively chaining conditional statements and thereby connects general sensor values to specific conclusions. ARES's general line of reasoning is as follows: "What sensors should I read? What numerical values do these sensors currently offer? What linguistic values may I use to describe these values, how are these linguistic values defined, and how accurately can they be applied? What conclusions can I draw from the linguistic descriptions I just developed? Based on these conclusions, what instructions should I send to the assembly. How should I resolve possible conflicts in the instructions?" These questions are generally answered by facts and rules found in the knowledge base. However, numeric sensor values are read from data files. The general "meta-knowledge" that is needed to interpret and chain rules, evaluate the "fuzzy" logic embedded in the rules, and resolve conflicts between instructions is encode into the inference engine shell.

ARES's knowledge base contains the knowledge needed to control the Atmosphere Revitalization assembly. Rather than expressing the control function as sets of feedback equations as is done for conventional controls, the control function is express as a set of conditional rules. Composed in a near-english format these rules are written for ARES as one would write instructions to a human performing the same job. The general format for a conditional rule is

(<classifier>: if <statement> then <conclusion> <weight>),

where <statement> and <conclusion> are generally of the form

<noun> is/is_not <modifier>.

Note <statement> and <conclusion> can really be of any form but it is preferred that they convey meaning to both developers and users.

The inference engine interprets conditional rules by searching through the knowledge base for a fact that matches the <statement>. The <statement> has a weight value (0-1) associated with it called the confidence value. The confidence value represents the confidence an expert would have in the <statement> for the given circumstances. ARES is constructed so that the <statement> can always be found (eventually) although its confidence value may be zero. Having found the <statement>, the inference engine asserts the <conclusion> into the knowledge base and gives it a confidence value. The confidence value for the conclusion is calculated by multiplying the confidence value found for the <statement> by the <weight> value associated with the conditional rule. A future version will support multiple layers of conclusions and drop the structural restrictions imposed by the <classifier>: convention.

Once asserted by the inference engine, the <conclusion> can be a valid <statement> for another rule. In this manner, conditional rules can be chained. But how are initial statements provide to start the chains? By membership rules that translate numeric sensor values into descriptive linguistic values. The general form of a membership rule is

(if <sensor> is X then <noun> is <modifier>
<min> <low> <high> <max> <units>),

where <sensor> is the name of a sensor listed in the knowledge base. The "<noun> and <modifier>" is the conclusion that the inference engine asserts. The numeric values <min> <low> <high> and <max> define a triangular "fuzzy" membership function used to calculate the confidence value for membership rule's conclusion. If the value read from the <sensor> is between <low> and <high>, then the confidence in the conclusion is absolute (its confidence value is 1). If the value read is less than <min> or greater than <max>, then the confidence in the conclusion is null (its confidence value is 0). All other possible values for the <sensor> are linearly interpolated to produce a continuous function. A <units> is just a term used for the reference of the developer and user.

ARES supports statements of the form

<statement A> <operator> <statement B>,

where <operator> is either OR/AND thereby permitting fuzzy logic. The confidence of the statement is resolved according to fuzzy algebra.

ARES also support statements of the form

<noun> is_not <modifier>.

The confidence value is the fuzzy logic's complement of the statement "<noun> is <modifier>."

It is the nature of an expert's decision process to decide between alternative or conflicting options. If the options represent points values along a roughly continuous scale, the conflict can be resolved by taking the centroid of the points as a compromise value (use confidence values as weights). But, in ARES, the subassemblies only accept a few discrete non-scaler values: ON, OFF, STANDBY, ROLLOVER, PLAYDEAD, ... ARES resolve conflicts by choosing the most confident of the conflicting values.

IMPLEMENTATION:

In APPENDIX 4D is the listing for SAVANT.3. SAVANT.3 is a program written for NASA's CLIPS expert system development shell. The first section of SAVANT.3's code is the particular knowledge base used to verify the system. All the following sections of SAVANT.3 comprise the inference engine that would be transparent to the user.

Expanded knowledge bases are presented in APPENDIXES 4A-4C. These knowledge bases were developed by three teams for the CO_2 Removal Assembly, CO_2 Reduction Assembly, and Oxygen Generation Assembly. The assembly knowledge bases together contain rules used to identify current operation conditions and diagnose problems in the AR. They are based on expertise of how the AR sub-assemblies should be controlled and how ARES should respond to expected conditions.

The following sections discussed the rules of each assembly knowledge base in depth.

4.2.1 CO_2 REMOVAL ASSEMBLY

The CO_2 Removal Assembly is responsible for removing excessive carbon dioxide from the cabin air. This removal is accomplished by a four-bed molecular sieve. The sieve contains two beds for removing moisture from the incoming air and two beds for removing the carbon dioxide. In addition to the four beds, the system contains a blower for moving the air, a precooler, a CO_2 pump and accumulator, and a variety of flow control valves. A system diagram of the assembly is shown in Fig. 4.2-1.

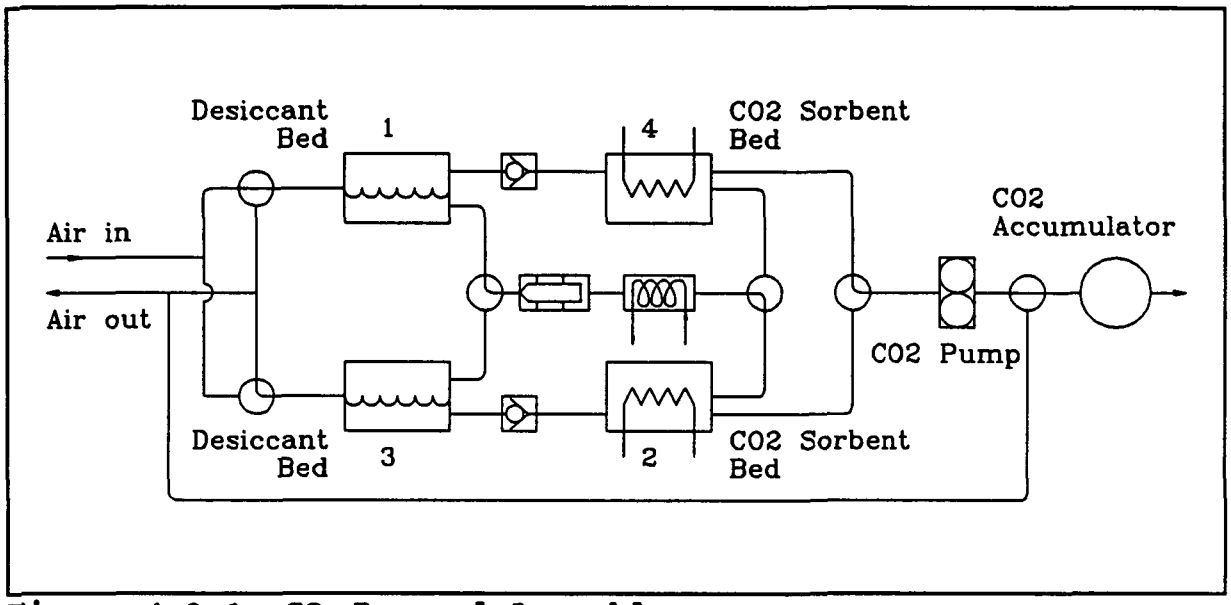

Figure 4.2-1 CO_2 Removal Assembly

The purpose of this section is to indicate the work that has been done on designing an expert system to monitor and control the assembly. The rules were written to be compatible with the CLIPS expert system package and are located in Appendix 4B.

The main function of the designed expert system was to monitor the components of the CO_2 Assembly. Different sensor values are read and compared to certain parameters and the expert system makes a decision based on those sensor values. The focus of the rules was to detect component failures. Currently, the rules do not include any valves or the four beds.

Rule # 1

The expert system monitors the temperature sensor at the output of the precooler. If the value does not fall within accepted parameters, the expert system makes a decision if failure has

occurred or not. If a failure decision is reached, the expert system instructs the Removal Assembly to shut down.

Rule # 2

The expert system monitors the two pressure sensors located on either side of the blower and checks for the differential pressure. If the pressure is unacceptable, the Removal Assembly is instructed to shut down.

Rule # 3

The expert system monitors the power sensor on the CO_2 pump. If the value is unacceptable, then the Removal Assembly is instructed to shut down.

Rule # 4

The expert system monitors the pressure sensor located on the CO_2 pump. If the value is not within the accepted parameters, the Removal Assembly is instructed to shut down.

Rule # 5

The expert system monitors the gas flow at the CO_2 accumulator through two gas flow sensors located on either side of the accumulator. If a failure decision is reached, both the Removal and the CO_2 Reduction subassemblies are instructed to shut down.

4.2.2 CO_2 REDUCTION ASSEMBLY

The CO_2 Reduction Assembly is comprised of a Sabatier Methanian Reactor (SMR) which converts CO_2 and H_2 to methane gas (CH_4) and water vapor (H_2O), a Condenser/Separator (C/S) to remove product water, and a Carbon Formation Reactor (CFR) to break down methane into carbon and hydrogen. A block diagram of the CO_2 Reduction Assembly is shown in Figure 4.2-2.

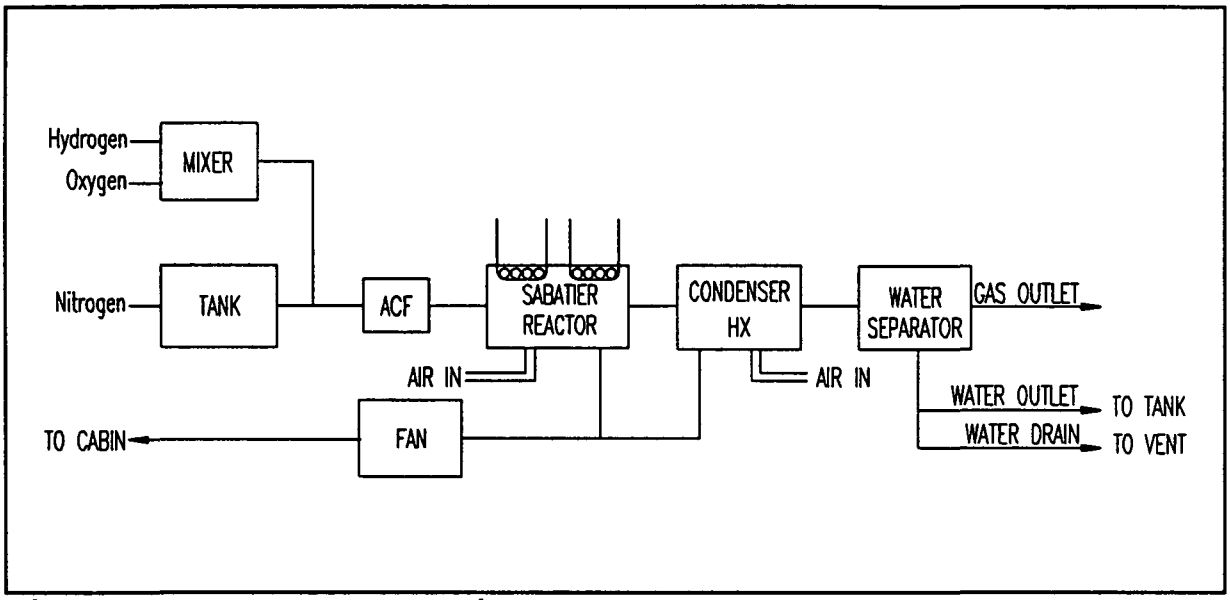

Figure 4.2-2 CO_2 Reduction Assembly

The main focus of this section will be on the steps that were taken to construct an expert system for the SMR. The rules that were written in CLIPS will be explained in the order that the reactants flow through the SMR. See Appendix 4C for the rules as found in CLIPS.

The main purpose of the SMR is to catalytically react CO_2 with H_2 to form methane gas (CH_4) and water vapor (H_2O). The SMR receives CO_2 from the CO_2 Removal Assembly and the H_2 is recycled from the carbon formation reactor and is also received from the CO_2 Removal and Oxygen Generation Subassemblies. The chemical reaction that takes place in the SMR is written as

$$CO_2 + 4H_2 = CH_4 + 2H_2O + HEAT.$$

A 4.0 molar ratio, which is equivalent to a molecular weight of 10.4 g/gmol of H_2 to CO_2, must be maintained. To do this a molecular weight sensor was already included in the prototype SMR, so this was not included in the expert system sensors. The actual volume of CO_2 was needed to accurately control the

molecular weight sensor which was not readily available and difficult to approximate.

Rule # 9

The chemical reaction to convert CO_2 and H_2 to methane and water is exothermic. Therefore the heaters used in the SMR are primarily for reactor start up. Two redundant heaters are used to heat up the catalyst bed to initiate the reaction [Forsythe, 1984]. A sensor to indicate a non-functional heater or heaters was included in the expert system. When the reaction is below 400^0K which would normally be during reactor start-up, there is no conversion taking place. This would indicate to the system that a heater or heaters has broken down and the system is turned off so maintenance procedures can take place.

Rules # 7 and 8

The chemical reaction is also self-limiting via the reverse endothermic reaction, to temperatures below 873^0K [8:]. This causes a loss in the normally expected 98% conversion of carbon dioxide. Thus the efficiency of the reactor decreases. To monitor this, two rules were added to the expert system. The efficiency level is low below the temperature of 850^0K. When this occurs the expert system instructs the SMR to shut off since low efficiency could be an indication of another problem. A normal efficiency level was also included in the rules when the temperature of the reaction was between 850^0K and 873^0K. The SMR would continue to run as normal if this rule was executed.

Rules # 1, 2, 3, and 4

The next step in the SMR is to cool the chemical reaction temperature down to the point that water can condense after it has left the reactor. To do this a cooling jacket consisting of a number of stainless steel serrated fins covered with a shell of stainless steel is wrapped around the reactor bed and cooling air is forced through a mid and rear cooling jacket inlets [Forsythe,1984]. During the simulations of zero gravity on the SMR it was found that a coolant flow of approximately 0.424 cfm would be required for each jacket to compensate for the loss of external convection [1:]. Since the only temperature modeled by the CO_2 Reduction Assembly modelers was the reaction temperature, it was difficult to assume a temperature for the two cooling jacket zones. The coolant flow did not seem to be an adjustable variable. It was either on or off. Using these assumptions the rules were based on having a coolant flow of 0.424 cfm. If either of the jacket flows was below 0.2 cfm the heat would stay in the reaction and the exit temperature would be too high for the condenser/separator. This indicated a problem with one or both of the coolant flows. Therefore the system would be shut

down for maintenance. There is also a normal flow mode between 0.4 and 0.7 cfm when the SMR continues running.

Rules # 5 and 6

Although the condenser/separator has not been modeled, an attempt to include it in the expert system was made. Since the condenser requires the water to be in a liquefied state it was assumed that the exit temperature of the SMR would have to be approximately 373^0K. Therefore, if the exit temperature was above this value then the condenser temperature would be too high and the SMR was turned off. A high exit temperature could be a result of the coolant flows not working. If the exit temperature was below 373^0K then the SMR was considered running normally.

The Carbon Formation Reactor was not included in the CO_2 Reduction Assembly expert system. There was limited information on the CFR and there had also been no attempts to model it.

4.2.3 OXYGEN GENERATION ASSEMBLY

The Oxygen Generation Assembly, OGA, is comprised of a electrolysis module and water tanks to convert H_2O to H_2 and O_2 by the use of a catalysis, KOH. A block diagram of the OGA is shown in figure 4.2-3.

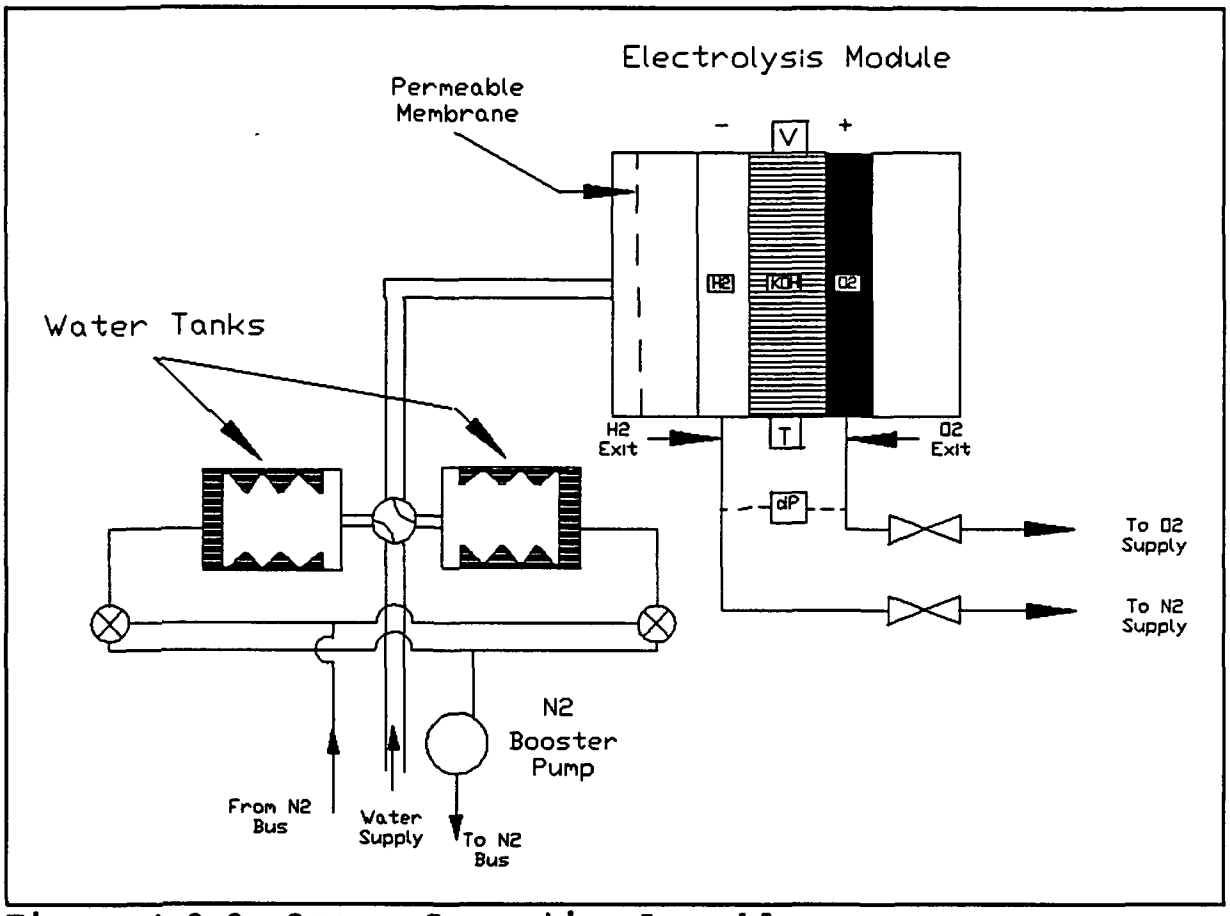

Figure 4.2-3 Oxygen Generation Assembly

The main focus of this section will be on the steps that were taken to construct an expert system for the OGA. The rules that were written in CLIPS will be explained in the order that the reactants flow through the OGA. See Appendix 4A for the rules as found in CLIPS.

The main purpose of the OGA is to produce O_2 by the electrolysis module. The OGA receives water from the CO_2 Reduction Assembly and hygiene water. For the purpose of writing an expert system to control the Atmosphere Revitalization Assembly, It was assumed that all water supply would come from the CO_2 Reduction Assembly.

Rules # 1, 2, and 3

The water in the storage tanks must be moved to the electrolysis module by the use of nitrogen pressure on the bellow of the water tanks. This pressure is the H_2O feed line pressure. From Boeing document 683-10011, the H_2O feed line pressure needed was given as 25 ±5 psia. Rule 2 defines the normal pressure to the electrolysis module between 20 and 30 psia. Rules 1 and 3 are for high and low feed line pressure. If these rules are activated, the OGA will be instructed to shutdown for maintenance.

Rules # 4 and 5

To measure the amount of water in the H_2O supply tanks, an emitter detector infrared sensor was used. If the bellow was obstructing the infrared sensor a binary output of one was the output of the sensor. If the water supply in the tank was low, the bellow would not obstruct the infrared sensor and the sensor outputs a binary output of 0. In this manner the supply of water for the electrolysis module was measured. An output of one indicates water supply is sufficient. Where an output of zero indicates the water supply is insufficient.

Rules # 6, 7, and 8

After the H_2O passes through the electrolysis unit its components are H_2 and O_2. The differential pressure between the H_2 and O_2 output lines is an indication of the electrolysis efficiency. Boeing document 683-10011 specifies the H_2-O_2 differential pressure to be 2.83-3.35 psia for normal operation. If the differential pressure is lower or higher than this specified value it is an indication that one of the cell electrodes has failed and the OGA is instructed to shutdown.

Rules # 9, 10, and 11

The final constraint placed on the OGA is the temperature of O_2 returned to the cabin. Although the temperature of O_2 gas is not an output from the models designed by our group, it has been included in the expert system. The acceptable output temperature listed in Boeing document 683-10011 is 55°-105.8° F. If the temperature of O_2 gas to the cabin is outside this range the OGA is instructed to shutdown.

Rules # 12 and 13

The electrolysis unit uses a series of voltage cells to control and initiate the breakdown of H_2O. If all cell voltages are high meaning greater than 1.7 volts, as listed in NASA document 2-H8RG-BMS-194/90, this is an indication of that the water supply

to the electrolysis module has been lost. Therefore, if all cell voltages are high the OGA is instructed to shutdown.

An important section of ARES is the Explanation Facility. This feature is not necessary for operation of ARES, but very important. ARES is only as accurate as the experts of each assembly. In other words, it is no smarter than its programmers and therefore is suspect to incorrect conclusions. The Explanation Facility allow the user to see the steps taken by ARES in making a conclusion for each assembly. It does this by retracing the paths used to chose responses. Then ARES generates a dialogue explaining the responses, allowing the user to refute the decision.

4.3 VERIFICATION AND TESTING OF EXPERT SYSTEM

ARES is designed to help monitor and control the Environmental Control and Life Support System for the Space Station Freedom (SSF). Currently, the system only has control over limited portions of the Air Revitalization System. After rules that govern system's behavior were written and placed into the knowledge base, the expert system was tested to discover any programming, knowledge, or conceptual errors that may have been present.

The final stage of the expert system's development is verification - applying a series of inputs and insuring the decisions made were correct. Though the current system contains only four components that may be varied, verification is a difficult task. To verify all possible condition combinations would require around 300 individual tests. The complete expert system to govern the entire ECLSS will be nearly impossible to verify completely. For this report, 12 different sets of conditions were tested and checked. These twelve conditions sets fall into nine cases as described below. For each case, the conditions are stated first, then a brief description of the system response is given. The four conditions which can be varied are:

Cabin_O_2 - Indicates the concentration of O_2 in the cabin. Helps control Oxygen Generation Assembly (OGA) and CO_2 Removal Assembly

Inlet_CO_2 - Indicates concentration of CO_2 present in the cabin atmosphere. Helps control CO_2 Removal Assembly.

CO_2_Accumulator - Indicates level of CO_2 storage tank located between CO_2 removal and CO_2 reduction assemblies. Helps control these same two assemblies.

H2O_Accumulator - Indicates level of water in storage tank located between CO_2 reduction and OGA assemblies. Helps control these same two assemblies.

CASE 1:

Condition: All values in normal operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Removal - Commanded to move to an efficiency mode. Since no supplies are low and cabin CO_2 is normal, CO_2 Removal Assembly can reduce power requirements by decreasing production slightly.

CASE 2:

Condition: Both accumulators are empty, all other values in normal operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode. This system will not, however, be able to operate if no water is available. This system should perhaps turn off for a while until water is available. This is the correct response as programmed although not really the correct way to handle the given situation.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode. This is to refill the empty H2O accumulator.

CO_2 Removal - Commanded to move to high production mode. This will help to replenish the CO_2 accumulator and provide resources needed by the CO_2 Reduction assembly to refill H2O Accumulator.

CASE 3:

Condition: Both Accumulators at a low level, all other values in normal operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode. This system will not, however, be able to operate if no water is available. This system should perhaps turn off for a while until water is available. This is the correct response as programmed although not really the correct way to handle the given situation.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode. This is to refill the empty H2O accumulator.

CO_2 Removal - Commanded to move to high production mode. This will help to replenish the CO_2 accumulator and provide resources needed by the CO_2 Reduction assembly to refill H2O Accumulator.

CASE 4:

Condition: Both Accumulators at a high level, all other values in normal operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Removal - Commanded to move to an efficiency mode. Since no supplies are low and cabin CO_2 is normal, CO_2 Removal Assembly can reduce power requirements by decreasing production slightly.

CASE 5:

Condition: Inlet_CO_2 at a high level, all other values in normal operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Removal - Commanded to move to a high removal mode. This is an attempt to remove the excess CO_2 from the cabin and restore a proper CO_2 - O2 balance for the astronauts.

CASE 6:

Condition: Inlet CO_2 at a low level, all other values in normal operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Removal - Commanded to turn off. This is done to avoid overfilling the CO_2 accumulator before the need to remove CO_2 becomes important. This also conserves power at times when the system is not needed.

CASE 7:

Condition: Cabin_O2 level is low, all other values in normal operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode. Since there is no high production mode, the expert system can only make sure the OGA is producing oxygen as the environment requires.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Removal - Commanded to move to an efficiency mode. Since no supplies are low and Cabin CO_2 is normal, CO_2 Removal Assembly can reduce power requirements by decreasing production slightly. This will not affect the rate of oxygen production which is most important in this test case.

CASE 8:

Condition: Cabin O2 at a critical level, all other values in normal operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode. Again, we would desire a high production mode, but none is available. We can only make sure the OGA is on and operating.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Removal - Commanded to move to a high removal mode. This is done to help restore CO_2 - O2 balance in cabin and to insure that adequate resources are available for the OGA to continue operating normally.

CASE 9:

Condition: All values at a high level in the operating range

Response:

OGA - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Reduction - commanded to turn on (if currently off) or to continue in normal operating mode.

CO_2 Removal - Commanded to move to a high removal mode. This is done to compensate for excess CO_2 in the cabin.

SPECIAL NOTE: For this section, one of the input values, the CO_2 accumulator, did not get read into the system. The results may have been different if this value was available.

Limitations

The biggest limitation at present is the number of rules which the system can operate on. With only four pieces of data to make decisions with, those decisions will be limited in scope and accuracy. Additionally, some things have not been taken into account. For example, the system will continue to remove CO_2 from the cabin even if the CO_2 accumulator is full. Next, the system produces confidence factors although these factors currently have no meaning. Finally, the expert system does not always produce the best decision. As noted in cases 2 and 3 above, the OGA is commanded to turn on even though adequate resources for operation may not be available. It is evident that further options need to be considered and tested before a final decision is made as to the operation of the system.

4.4 CONCLUSIONS

Because of the complex nature of the conditions which may affect operation of the ECLSS, standard control schemes do not provide convenient or adequate means for system development and maintenance. The expert system, on the other hand, can handle systems with multiple inputs and outputs and non-linear behavior. Expert systems can be tolerant of inexact or incomplete information permitting rapid prototyping. With an expert system in charge ECLSS will not require constant supervision (a premium commodity on a space flight). For these reasons, an expert system solution makes a better choice for ECLSS control.

The expert system constructed this semester used NASA's CLIPS expert system development shell. The CLIPS shell program does not readily support real-time modeling or color graphics user interfacing. Future effort should include a search for development tools that can provide this support.

The rules defined in the previous sections obviously do not represent a set which will completely control the Air Revitalization assembly. Rather, these rules are a beginning set which define the overall or general operation of the assembly's main components. Many other, more detailed rules will be needed to complete the knowledge bases for the expert system and to provide an adequate control system for the ECLSS. Further information about the operation and interaction of the various sub-assemblies will be required before such rules can be defined.

The Savant.3 decision mechanism is the most complete portion of the current expert system design. The inference engine, which interprets knowledge base data, and the response selector, which chooses an action based on the number of conclusions and their associated confidence factors, will not be affected by the addition of new rules. In fact, these sections could be used with minor alteration in an expert system for control of the entire ECLSS.

System testing validated the programming of the SAVANT.3 inference engine and its general application of deductive reasoning. This testing also validated the technique of representing the knowledge base. While the condensed version of the control scheme generally worked, the expert system faithfully demonstrates certain fallacies in the scheme. But this real objective of the ECLSS simulation -- to help verify the conceptual control schemes.

5.0 APPENDICES

1.0 REFERENCES

2.2 WATER QUALITY MONITOR

1. Baer-Peckham, David L. Water Recovery and Management System Major Components List. Seattle, Washington: Boeing Company, 16 April 1990.

2. Boeing Company. Process Control Water Quality Monitor Schematic. Seattle, Washington: Corporate Offices, 18 April 1990.

3. Boeing Company. Water Quality Monitor Schematic. Seattle, Washington: Corporate Offices, 8 April 1990.

4. Conklin, Keith J. Process Control Water Quality Monitor (PCWQM) Control Description. Memo #2-H8RG-KJC 226/90 Seattle, Washington: Boeing Company, 9 April 1990.

5. Conklin, Keith J. Process Control Water Quality Monitor (PCWQM) Description. Seattle, Washington: Boeing Company, 13 April 1990.

6. Conklin, Keith J. Water Quality Monitor (WQM) Control Description. Memo #2-H8RG-KJC 228/90 Seattle, Washington: Boeing Company, 9 April 1990.

7. Conklin, Keith J. Water Quality Monitor (WQM) Description. Seattle, Washington: Boeing Company, 6 March 1990.

8. Whitley, Ken. Water Quality Monitor. Houston, Texas: Marshall Space Flight Center--Environmental Control and Life Support Branch, 27 June 1990.

2.3 URINE PROCESSOR ASSEMBLY

1. McGriff, Cindy. Urine Processing. Environmental Control and Life Support Branch. National Aeronautics and Space Administration (NASA). 27, June 1990.

2. Miernik, J. H. TIMES Urine Processor Assembly (UPA) Description. Memo #2-H8RG-JHM-223/90. Boeing Company. 19, June 1990.

3. Miernik, J. H. TIMES Urine Processor (UPA) Control Description. Memo #2-H8RG-JHM-225/90. Boeing Company. 19, April 1990.

4. Miernik, J. TIMES Urine Processor Schematic. Dwg # SK683-20510. Boeing Company. 30, April 1990.

5. Miernik, J. H. TIMES Urine Processor Resource Requirements. Memo #2-H8RG-JHM-249/90. Boeing Company. 19, April 1990.

6. Miernik, J. H. TIMES Urine Processor Assembly (UPA) Instrumentation List. Memo #2-H8RG-JHM-224/90. Boeing Company. 19, April 1990.

7. Miernik, J. H. TIMES Urine Processor Electrical Schematic. Dwg # SK683-20511. Boeing Company. 20, April 1990.

8. Miernik, J. TIMES Urine Processor Rack Schematic. Dwg #SK683-20500. Boeing Company. 2 Sheets. 7, April 1990.

9. Miernik, J. Urine Processor Assembly (UPA) ORU List. Memo #2-H8RG-JHM-248/90. Boeing Company. 19, April 1990.

2.4 HYGIENE WATER PROCESSOR

1. Holder, Don. Hygiene Water Recovery. NASA Document (6-27-90).

2. Shaw, Glenn. Hygiene Water Processor Resource Requirements. NASA Document 2-H8RG-RGS-253/90 (1990): 1.

3. Shaw, Glenn. Hygiene Water Processor Control Description. NASA Document 2-H8RG-RGS-222/90 (1990): 1-6.

4. Shaw, Glenn. Hygiene Water Processor Instrumentation List. NASA Document 2-H8RG-RGS-221/90 (1990): 1-12.

5. Shaw, Glenn. Hygiene Water Processor Description. NASA Document 2-H8RG-RGS-258/90(1990) : 1-4.

2.5 POTABLE WATER PROCESSOR

1. Bagdigian, R.M. and P.L. Mortazavi. Status of the Space Station Water Reclamation and Management Subsystem Design Concept. NASA. George C. Marshall Space Flight Center. Science and Engineering Directorate. Environmental Control and Life Support Branch. Document # 371510. Society of Automotive Engineers: 1987.

2. Carter, Layne. Potable Water Recovery. NASA. George C. Marshall Space Fight Center. Science and Engineering Directorate. Environmental Control and Life Support Branch. June 27, 1990.

3. Hayase, John K. Potable Water Processor Control Description. Boeing Inc. Document # 2-H8RG-JKH 213/90. April 18, 1990.

3.2 CO_2 REMOVAL

1. College of Engineering, Kansas State University, Automation of Closed Environments in Space for Human Comfort and Safety, Academic Year 1989-90, pp. 8.

2. Dr. B. Jones, Professor of Mechanical Engineering, Kansas State University. (Consultant)

3. Thermodynamics, An Engineering Approach, Yunus A. Cengel and Michael A. Boles, New York: McGraw-Hill Publishing Co., 1989.

3.3 CO_2 REDUCTION

1. SAE Technical Paper Series, #840936, A Study of Sabatier Reactor Operation in Zero 'G'. Robert K. Forsythe, Charles E. Verostko, Robert J. Cusick and Robert L. Blakely. 1984 Society of Automotive Engineers, Inc.

2. Elements of Chemical Reaction Engineering. Scott Fogler.

3. Chemical Process and Thermodynamics. B. G. Kyle.

3.4 OXYGEN GENERATION ASSEMBLY

1. Automation of Closed Environments in Space for Human Comfort and Safety: Report of Academic Year 1989-90, Kansas State University College of Engineering; pages 93-107.

2. Document 2-H8RG-BMS-194/90, Oxygen Generation Description Control Description, pages 1-7.

3. Oxygen Generation Subassembly, NASA, pages 1-11.

4.0 Expert System

1. SAE Technical Paper Series, #840936, A Study of Sabatier Reactor Operation in Zero 'G'. Robert K. Forsythe, Charles E. Verostko, Robert J. Cusick and Robert L. Blakely. 1984 Society of Automotive Engineers, Inc.

2. Self, K. Designing with Fuzzy Logic. IEEE Spectrum. Vol. 27 (11), pp. 105. Nov 1190.

3. Kraus, H. The Development of a Fuzzy Controller and the Comparison of Its Performance Against a Conventional PD Controller. April 1991.

4. Kraus, H. GEDANKEN: A Language Interpreting Artificial Reasoner. (software), March 1991. K-State.

5. Kraus, H. SAVANT.3: A Language Interpreting Artificial Reasoner. (software), March 1991. K-State.

6. Giarratano, J. and Riley, G. Expert Systems: Principles and Programming, PWS-KENT Publishing Co., pp. 64, 1989.

APPENDIX 2A
CO_2 REMOVAL ASSEMBLY PROGRAM

```
/*************************************************************
 *                                                           *
 *  SOURCE FILE:   REMOVE2.C    AT&T 3B1 Version             *
 *                                                           *
 *  DESCRIPTION:   This program models the CO2 removal system. *
 *                 It simulates one adsorption and desorbtion  *
 *                 cycle.  The desorbing desiccant bed has not *
 *                 been included in this model.  This program  *
 *                 can be used as a design tool to test the    *
 *                 CO2 removal system equations.               *
 *                                                           *
 *  USES:          main()                                     *
 *                 simple_plot()     (S. Dyer)                *
 *                                                           *
 *  AUTHORS:       Robert Young, Terry Hon, Joel Sommer       *
 *                                                           *
 *  LAST REVISION: 06 March 1991                              *
 *                                                           *
 *  LIMITATIONS:  The desorbing desiccant bed has not been    *
 *                included in this model because the CLIPS    *
 *                Expert System is not concerned with the     *
 *                rehumidification of cabin air.  Only the    *
 *                levels of CO2 into the Removal System, the  *
 *                CO2 released to the cabin, the CO2 level of *
 *                the accumulator and the CO2 demand of the   *
 *                Reduction System are currently necessary for *
 *                CLIPS to make an inference about mode       *
 *                transitions.                                *
 *                                                           *
 *************************************************************/

#include <stdio.h>
#include <math.h>
#include "p_plot.h"
#include "simple_plot.h"
#define NUM_POINTS 100

main()
{
int dummy, pen;
long int i;
char rite;
double Time[100],     /** time array **/
       xTime,
       Mg[100],       /** mass of gas in void space **/
       xMg,
       Md[100],       /** mass of gas in sorbent material **/
       xMd,
       Tb[100],       /** temperature of bed and exiting air **/
       xTb,
       Ua[100],       /** internal energy of accumulator **/
       xUa,
       Ma[100],       /** mass of CO2 in accumulator **/
       xMa,
       Pp[100],       /** output pressure of pump **/
       xPp,
       Tp[100],       /** output temperature of pump **/
```

```
        xTp,
        Ta[100],        /** temperature of accumulator **/
        xTa,
        Conc_w[100],    /** relative humidity of outgoing air **/
        xConc_w,
        Ccs_w,          /** conc. of H2O on sorbent (kgH2O/kgair) **/
        M_flow,         /** mass flow rate of air (kg/sec) **/
        Ti,             /** incoming air temperature (K) **/
        T_des[100],     /** temp. of air leaving desiccant bed (K) **/
        xT_des,
        xT_fan,         /** temperature of air after fan (K) **/
        xT_hx,          /** air temp. after heat exchanger (K) **/
        Cci_w,          /** conc. of H2O entering bed (kgH2O/kgair) **/
        Cco_w,          /** conc. of H2O leaving bed (kgH2O/kgair) **/
        Md_w[100],      /** mass of H2O on sorbent (kg) **/
        xMd_w,
        Ccs,            /** conc. CO2 in sorbent (kgCO2/kg air) **/
        Cci,            /** conc. CO2 entering bed (kg CO2/kg air) **/
        Cco,            /** conc. CO2 leaving bed (kg CO2/ kg air) **/
        Conc[100],      /** conc. CO2 leaving bed (ppm) **/
        xConc,
        T_sorb[100],    /** temp. of air leaving sorbent bed (K) **/
        xT_sorb;


double Pg,          /** CO2 pressure in void space **/
       Pb,          /** pump exit pressure **/
       m_o,         /** mass flow rate out of bed   dMg/dt **/
       m_out,       /** assembly exiting flow rate (bang-bang) **/
       w,           /** angular velocity of pump (rad/sec)  **/
       Q_loss,      /** accumulator heat loss (bang-bang) **/
       Tref,        /** reference temp. of desorbent bed  **/
       k_w,         /** ratio of Cco_w/Ccs_w **/
       k_s;         /** ratio of Cco/Ccs **/

double Power=1000.0,
       Mb=20.0,          /** mass of sorbent material (kg) **/
       Cpa=1003.0,       /** heat capacity of air (J/kgK) **/
       Cvs=844.0,        /** heat capacity of sorbent (J/kgK) **/
       Sc_w=2700.0E3,    /**     heat of sorption of H2O
                                 into sorbent (J/kgK)        **/
       Cvb=657.0,
       Cp = 846.0,
       R=0.1889,
       To=0.0,       /** reference temperature for internal energy  **/
       Vg=1.0,
       Va=2.0,
       k1=0.7,
       k2=.0006,
       k3=.0023,
       k4= 2.0,      /** accumulator heat transfer coef. (Watts/K) **/
       k = 1.29,
       Sc=951E3;     /** CO2 heat of sorption into sorbent (J/kgK) **/

double dt,
       dTb,
       dT_des,
       dT_sorb,
       dMd_w,
       dMg,
       dMd,
       dMa,
```

```
      dUa;

                /** Inputting of parameters by user **/

printf("\nInput mass flow rate of air, M_flow (kg/s) : ");
scanf("%lf",&M_flow);
printf("\nInput H2O concentration entering assembly,");
printf(" Cci_w (kgH2O/kg air) : ");
scanf("%lf",&Cci_w);
printf("\nInput concentration of CO2 entering assembly,");
printf(" Cci (kgCO2/kg air): ");
scanf("%lf",&Cci);
printf("\nInput temperature of incoming air, Ti (K) : ");
scanf("%lf",&Ti);
printf ("\nInput angular velocity of pump (radians/second) : ");
scanf ("%lf",&w);
printf ("\nInput desired bed reference temperature, Tref (K) : ");
scanf ("%lf",&Tref);

Time[0] = 0.0;          /** absolute start time in minutes **/
xTime = 0.0;
                        /** initial conditions for desiccant bed **/

   Md_w[0] = xMd_w = 0;
   T_des[0] = xT_des = Ti;
   Ccs_w = Md_w[0] / Mb;
   k_w = 0.1;
   Cco_w = k_w * Ccs_w;
   Conc_w[0] = xConc_w = Cco_w * 10000.0;
                        /** initial conditions for sorbent bed **/

   Md[0] = xMd = 0.0,
   T_sorb[0] = xT_sorb = Ti;
   Ccs = Md[0] / Mb;
   k_s = 0.1;
   Cco = k_s * Ccs;
   Conc[0] = xConc = Cco * 1.0E6;
   dt = 1.0;
printf("\n\nWorking desiccant and sorbent bed forward cycle...\n\n");

             /************************************************
              *                                              *
              *   Desiccant bed and Sorbent bed calculations *
              *                                              *
              ************************************************/
  i = 1;
  while (i<100) {
    for (dummy=1;dummy<=33;dummy++){

                        /** desiccant bed equations **/

      dT_des = dt * ((M_flow * Cpa * (Ti - xT_des)) +
                     (M_flow * Sc_w * (Cci_w - Cco_w))) / (Mb * Cvs);
      dMd_w = M_flow * (Cci_w - Cco_w) * dt;
      xT_des = xT_des + dT_des;
      xMd_w = xMd_w + dMd_w;
      Ccs_w = xMd_w / Mb;
      Cco_w = k_w * Ccs_w;
      xConc_w = Cco_w * 10000.0;

                /** fan and heat exchanger "equations" **/
```

```
    xT_fan = xT_des + 6.81;
    xT_hx = xT_des - 50.0;      /** this is a kludge **/

                /** sorbent bed calculations **/

    dT_sorb = ((M_flow * Cpa * (xT_des - xT_sorb)) +
              (M_flow * Sc * (Cci - Cco))) * dt / (Mb * Cvs);
    dMd = M_flow * (Cci - Cco) * dt;
    xT_sorb = xT_sorb + dT_sorb;
    xMd = xMd + dMd;
    Ccs = xMd / Mb;
    Cco = k_s * Ccs;
    xConc = Cco * 1.0E6;

    xTime = xTime + dt/60.0;
  }

  Time[i] = xTime;
  T_des[i] = xT_des;
  Md_w[i] = xMd_w;
  Conc_w[i] = xConc_w;
  T_sorb[i] = xT_sorb;
  Md[i] = xMd;
  Conc[i] = xConc;
  i = i + 1;
}

       /*******************************************************
        *                                                     *
        *  Plotting routine for Desiccant and Sorbent beds    *
        *                                                     *
        *******************************************************/

 printf("\n\nDone...");
 printf("\n\nPlease choose which device you wish to use");
 printf(" for plotting purposes.\n");
 printf("\nDISPLAY = 1");
 printf("\nPLOTTER = 2");
 printf("\n\n");
 scanf("%d",&pen);
 if (pen == 1)
    rite=DISPLAY;
 else rite=PLOTTER;

            /** Simple plot calls for desiccant bed data **/

 simple_plot(rite,i,Time,T_des,"Time","minutes","Temperature",
             "degrees Kelvin",
             "Temperature of Desiccant Bed (T_des[i])",1,CURVE);

 scanf("%d",&dummy);
 simple_plot(rite,i,Time,Md_w,"Time","minutes","Mass H2O","kg",
             "Mass H2O in Sorbent Bed Material (Md_w[i])",1,CURVE);

 scanf("%d",&dummy);
 simple_plot(rite,i,Time,Conc_w,"Time","minutes",
            "Relative Humidity","%",
            "Relative Humidity of Air Leaving Desiccant Bed",
             1,CURVE);

 scanf("%d",&dummy);
```

```
            /** Simple plot calls for sorbent bed data **/

 simple_plot(rite,i,Time,T_sorb,"Time","minutes","Temperature",
             "degrees Kelvin","Temp. of Air Leaving Sorbent Bed",
             1,CURVE);

 scanf("%d",&dummy);
 simple_plot(rite,i,Time,Md,"Time","Minutes","Mass CO2","kg",
             "Mass CO2 in Sorbent Bed Material (Md[i])",1,CURVE);

 scanf("%d",&dummy);
 simple_plot(rite,i,Time,Conc,"Time","minutes","Concentration",
             "ppm","Conc. CO2 Leaving Sorbent Bed",1,CURVE);

 scanf("%d",&dummy);

 system("clear");              /** clear the screen **/
 printf("\n\nWorking on desorbing cycle...");

            /** initial conditions for desorbing cycle **/

 Time[0] = xTime = 62;  /** reset array for this cycle **/
 Md[0] = xMd = Md[99];  /** reset array for this cycle **/
 Tb[0] = xTb = T_sorb[99]; /** reset array for this cycle **/
 Mg[0] = k1 * Vg * Md[0] * (Tb[0] - Tref) / (Mb * R * Tb[0]);
    xMg = Mg[0];
 Ua[0] = xUa = 407.34;
 Ma[0] = xMa = 0.002;
 Ta[0] = xTa = 300.0;
 Pp[0] = xPp = 0.0567;
 Tp[0] = xTp = 300.0;

               /*********************************************
                *                                           *
                *      Calculations for Desorbing cycle     *
                *                                           *
                *********************************************/

i = 1;
while(i<100){
  for(dummy=1;dummy<=28;dummy++){

    if (xTb > 330.0)  /** bang-bang bed temp. controller **/
       Power=0.0;
    else Power = 1000.0;

    Pg = xMg * R * xTb / Vg;
    Pb = k1 * xMd / Mb * (xTb - Tref);
    xPp = xMa * R * xTa / Va;
    xTp = xTb*pow((xPp/Pg),(1.0-1.0/k));
    xTa = xUa / (xMa * Cp) + To;
    m_o = w*k3*Pg/R/xTb;

    if (xMa < Ma[0])    /** bang-bang flow out control **/
    m_out = 0.0;
    else
    m_out = 0.75 * m_o;

    if (xTa > 600.0)    /** bang-bang accumulator cooler **/
       Q_loss=k4*(xTa - 300);
    else Q_loss=0.0;
```

```
      dMd = (Pg - Pb) * k2 * dt;
      dMg = ((Pb - Pg) * k2 - (w * Pg * k3)/R/xTb) * dt;
      dTb = (dMd * Sc / dt + Power)/Mb/Cvb * dt;
      dMa = dt * (m_o - m_out);
      dUa = dt * (m_o*(Tp[i-1]-To)*Cp-m_out*
                  (Ta[i-1]-To)*Cp - Q_loss);

      xMg = xMg + dMg;
      xMd = xMd + dMd;
      xTb = xTb + dTb;
      xMa = xMa + dMa;
      xUa = xUa + dUa;
      xTime = dt/60.0 + xTime;
    }

    Time[i] = xTime;
    Mg[i] = xMg;
    Md[i] = xMd;
    Tb[i] = xTb;
    Ma[i] = xMa;
    Ua[i] = xUa;
    Ta[i] = xTa;
    Tp[i] = xTp;
    Pp[i] = xPp;

    i = i + 1;
  }

   printf("\nPress number and return to begin plotting...");
   scanf("%d",&dummy);

         /** SIMPLE PLOT CALLS FOR DESORBENT CYCLE DATA **/

   simple_plot(rite,i,Time,Md,"Time","minutes",
                "Mass CO2","kg","Mass CO2 in Sorbent Bed Material (Md[i])",
               ,1,CURVE);

   scanf("%d",&dummy);
   simple_plot(rite,i,Time,Ma,"Time","minutes",
                "Mass CO2","kg","Mass CO2 in Accumulator (Ma[i])"
               ,1,CURVE);

   scanf("%d",&dummy);
   simple_plot(rite,i,Time,Ua,"Time","minutes",
                "Internal Energy","J",
              "Internal Energy in Accumulator (Ua[i])"
              ,1,CURVE);

   scanf("%d",&dummy);
   simple_plot(rite,i,Time,Tp,"Time","minutes", "Temperature",
              "K","Temperature of Pump Outlet (Tp[i])", 1,CURVE);

   scanf("%d",&dummy);

   simple_plot(rite,i,Time,Ta,"Time","minutes","Temperature",
              "K","Temperature of Accumulator (Ta[i])",1,CURVE);
   scanf("%d",&dummy);
   simple_plot(rite,i,Time,Pp,"Time","minutes","Pressure",
              "kPa","Outlet Pressure of Pump (Pp[i])",1,CURVE);
   scanf("%d",dummy);
   printf("\nDone sending plots...");
}
```

```
/*************************************************************
 * PROGRAM:   CLPMODEL.H        VERSION:  1.0                *
 *                                                           *
 * DESCRIPTION: Global variables for clpmodel.c              *
 *                                                           *
 * USES:      none                                           *
 *                                                           *
 * PROGRAMMER: Robert W. Young                               *
 *                                                           *
 * LAST REVISION: 03 March 1991                              *
 *                                                           *
 *************************************************************/

#define NORMAL 0   /** function main should return a value to DOS **/

/** GLOBAL VARIABLES **/
double time, tstop;
int pair;
int dummy;
long int i;

double xTime,          /** time in cycle                                **/
       xMg,            /** mass of gas in void space                    **/
       xMd,            /** mass of gas in sorbent material              **/
       xTb,            /** temperature of bed and gas desorbed          **/
       xUa,            /** internal energy of accumulator               **/
       xMa,            /** mass of CO2 in accumulator                   **/
       xPp,            /** output pressure of pump                      **/
       xTp,            /** output temperature of pump                   **/
       xTa,            /** temperature of accumulator                   **/
       xT_fan,         /** temperature of air after fan (K)             **/
       xT_hx,          /** air temp. after heat exchanger (K)           **/
       xConc_w,        /** relative humidity of outgoing air            **/
       Ccs_w,          /** conc. of H2O on sorbent (kgH2O/kgair)        **/
       M_flow,         /** mass flow rate of air (kg/sec)               **/
       Ti,             /** incoming air temperature (K)                 **/
       xT_des,         /** temp. of air leaving desiccant bed (K)       **/
       Cci_w,          /** conc. of H2O entering bed (kgH2O/kgair)      **/
       Cco_w,          /** conc. of H2O leaving bed (kgH2O/kgair)       **/
       xMd_w,          /** mass of H2O on sorbent (kg)                  **/
       Ccs,            /** conc. CO2 in sorbent (kgCO2/kg air)          **/
       Cci,            /** conc. CO2 entering bed (kg CO2/kg air)       **/
       Cco,            /** conc. CO2 leaving bed (kg CO2/ kg air)       **/
       xConc,          /** conc. CO2 leaving bed (ppm)                  **/
       xT_sorb;        /** temp. or air leaving sorbent bed (K)         **/

double Pg,             /** CO2 pressure in void space                   **/
       Pb,             /** pump exit pressure                           **/
       m_o,            /** mass flow rate out of bed   dMg/dt           **/
       m_out,          /** assembly exiting flow rate (bang-bang)       **/
       w,              /** angular velocity of pump (rad/sec)           **/
       Q_loss,         /** accumulator heat loss (bang-bang)            **/
       Tref,           /** reference temp. of desorbent bed             **/
       k_w,            /** ratio of Cco_w/Ccs_w                         **/
       k_s;            /** ratio of Cco/Ccs                             **/

double Power,
       Mb=20.0,        /** mass of sorbent material (kg)                **/
       Cpa=1003.0,     /** heat capacity of air (J/kgK)                 **/
       Cvs=844.0,      /** heat capacity of sorbent (J/kgK)             **/
       Sc_w=2700.0E3, /**     heat of sorption of H2O
                              into sorbent (J/kgK)                     **/
```

```
        Cvb=657.0,
        Cp = 846.0,
        R=0.1889,
        To=0.0,         /** reference temperature for internal energy **/
        Vg=1.0,
        Va=2.0,
        k1=0.7,
        k2=.0006,
        k3=.0023,
        k4= 2.0,        /** accumulator heat transfer coef. (Watts/K) **/
        k = 1.29,
        Sc=951E3;       /** CO2 heat of sorption into sorbent (J/kgK) **/

double dt,
        dTb,
        dT_des,
        dT_sorb,
        dMd_w,
        dMg,
        dMd,
        dMa,
        dUa;
FILE *clips,            /** file pointer for storage of sensor data    **/
         *model;        /** file pointer for storage of model data     **/
char    *mode;          /** string to hold CLIPS mode instructions     **/

/*****************************************************************
 * PROGRAM:   CLPMODEL.C        VERSION:  1.1                    *
 *                                                               *
 * DESCRIPTION: Simulate CO2 removal sub-assembly using math     *
 *              modeling equations derived in Fall 1990 report   *
 *              of "Automation of Closed Environments in Space   *
 *              for Human Comfort and Safety".                   *
 *              Program will provide five minutes of sensor      *
 *              data to an ASCII file to be read by CLIPS and    *
 *              acted upon.  CLIPS will generate an ASCII com-   *
 *              mand file which will direct operation of the     *
 *              model.                                           *
 *                                                               *
 * USES:       stdio.h                                           *
 *             math.h                                            *
 *             "clpmodel.h"                                      *
 *                                                               *
 * PROGRAMMER: Robert W. Young                                   *
 *                                                               *
 * LAST REVISION: 04 March 1991                                  *
 *                                                               *
 * LIMITATIONS:  The desorbing desiccant bed has not been        *
 *               included in this model because the CLIPS        *
 *               Expert System is not concerned with the         *
 *               rehumidification of cabin air.  Only the        *
 *               levels of CO2 into the Removal System, the      *
 *               CO2 released to the cabin, the CO2 level of     *
 *               the accumulator and the CO2 demand of the       *
 *               Reduction System are currently necessary for    *
 *               CLIPS to make an inference about mode           *
 *               transitions.                                    *
 *****************************************************************/
#include <stdio.h>
#include <math.h>
#include "model.h"
```

```
/*************************************************************
 * FUNCTION: read_model                                      *
 * PURPOSE:  Write data to CO2REMOV.DAT for re-starting model *
 * INPUTS:   none.                                           *
 * OUTPUTS:  time -- current time in half cycle              *
 *           xMg  -- mass of gas in void space               *
 *           xMd  -- mass of gas in sorbent material         *
 *           xTb  -- temperature of bed and gas desorbed     *
 *           xUa  -- internal energy in CO2 accumulator      *
 *           xMa  -- mass of CO2 in accumulator              *
 *           xT_sorb -- temperature of sorbent bed           *
 *           xT_des -- temperature of desiccant bed          *
 *           xMd_w -- mass of H2O in desiccant bed           *
 *           xTa -- temperature of accumulator               *
 *           xTp -- temperature of pump outlet               *
 *           xPp -- pressure of pump outlet                  *
 *           w -- pump speed                                 *
 *           power -- power applied to sorbent bed heaters   *
 *           pair -- which pair of beds is operating         *
 *           Cci -- incoming concentration of CO2            *
 *           Cci_w -- incoming concentration of H2O          *
 *************************************************************/
void read_model()
{
    model = fopen("co2remov.dat","r");
    fscanf(model,"%d %f %f %f %f %f",pair,time,xMg,xMd,xMd_w,xMa);
    fscanf(model,"%f %f %f %f %f %f",xTb,xT_sorb,xT_des,xUa,w,Power);
    fscanf(model,"%f %f %f %f %f",xTa,xTp,xPp,Cci,Cci_w);
    fclose(model);
}
/*************************************************************
 * FUNCTION: read_commands                                   *
 * PURPOSE:  Read CLIPS CO2REMOV.CMD file for model operation *
 *           instructions.  CLIPS commands are currently     *
 *           ignored.                                        *
 * INPUTS:   none.                                           *
 * OUTPUTS:  mode -- current mode of operation as determined *
 *                   by CLIPS                                *
 *************************************************************/
void read_commands()
{
    clips = fopen("co2remov.cmd","r");
    fscanf(clips,"%s",mode);
    fclose(clips);
}
/*************************************************************
 * FUNCTION: decode_commands                                 *
 * PURPOSE:  Decode the command issued by CLIPS into model   *
 *           parameters.  Since the CLIPS commands are ignored *
 *           no decoding is done.                            *
 * INPUTS:   mode -- current mode of operation as determined *
 *                   by CLIPS                                *
 * OUTPUTS:  tstop -- end of half cycle time                 *
 *           power -- power applied to sorbent bed heaters   *
 *           m_out_scale -- fraction of mass flow into CO2   *
 *                          accumulator released to CO2      *
 *                          reduction assembly               *
 *           w -- CO2 pump speed                             *
 *           blow -- blower speed                            *
 *************************************************************/
void decode_commands()
{
```

```
    tstop = 55.0;      /** model runs on 55 minute half cycles **/
}
/*****************************************************************
 * FUNCTION: adsorb_1                                            *
 * PURPOSE:  Simulate the operation of Desiccant bed #1, the     *
 *           blower, heat exchanger and sorbent bed #2.          *
 *           This provides for later simulation of component     *
 *           failures.                                           *
 *****************************************************************/
void adsorb_1()
{
    Ccs_w = xMd_w / Mb;
   k_w = 0.1;
   Cco_w = k_w * Ccs_w;
    xConc_w = Cco_w * 10000.0;
    Ccs = xMd / Mb;
   k_s = 0.1;
   Cco = k_s * Ccs;
    xConc = Cco * 1.0E6;
    dt = 0.1;
    for (dummy=1;dummy<=5*60/dt;dummy++){
          /** desiccant bed equations **/
    dT_des = dt * ((M_flow * Cpa * (Ti - xT_des)) +
      (M_flow * Sc_w * (Cci_w - Cco_w))) / (Mb * Cvs);
        dMd_w = M_flow * (Cci_w - Cco_w) * dt;
        xT_des = xT_des + dT_des;
        xMd_w = xMd_w + dMd_w;
        Ccs_w = xMd_w / Mb;
        Cco_w = k_w * Ccs_w;
        xConc_w = Cco_w * 10000.0;
     /** fan and heat exchanger "equations" **/
        xT_fan = xT_des + 6.81;
        xT_hx = xT_des - 50.0;      /** this is a kludge **/
                                 /** sorbent bed calculations **/
        dT_sorb = ((M_flow * Cpa * (xT_hx - xT_sorb)) +
                  (M_flow * Sc * (Cci - Cco))) * dt / (Mb * Cvs);
        dMd = M_flow * (Cci - Cco) * dt;
        xT_sorb = xT_sorb + dT_sorb;
        xMd = xMd + dMd;
        Ccs = xMd / Mb;
        Cco = k_s * Ccs;
        xConc = Cco * 1.0E6;
        xTime = xTime + dt/60.0;
    }
}

/*****************************************************************
 * FUNCTION: desorb_1                                            *
 * PURPOSE:  Simulate the operation of sorbent bed #4, CO2       *
 *           pump and CO2 accumulator.                           *
 *           This provides for later simulation of component     *
 *           failures.                                           *
 *****************************************************************/
void desorb_1()
{
    for(dummy=1;dummy<=5*60/dt;dummy++){
        if (xTb > 330.0)  /** bang-bang bed temp. controller **/
            Power=0.0;
        else
            Power = 1000.0;
        Pg = xMg * R * xTb / Vg;
        Pb = k1 * xMd / Mb * (xTb - Tref);
```

```
        xPp = xMa * R * xTa / Va;
        xTp = xTb*pow((xPp/Pg),(1.0-1.0/k));
        xTa = xUa / (xMa * Cp) + To;
        m_o = w*k3*Pg/R/xTb;
        if (xMa < 0.002)     /** bang-bang flow out control **/
            m_out = 0.0;
        else
            m_out = 0.75 * m_o;
        if (xTa > 600.0)     /** bang-bang accumulator cooler **/
            Q_loss=k4*(xTa - 300);
        else
            Q_loss=0.0;
        dMd = (Pg - Pb) * k2 * dt;
        dMg = ((Pb - Pg) * k2 - (w * Pg * k3)/R/xTb) * dt;
        dTb = (dMd * Sc / dt + Power)/Mb/Cvb * dt;
        dMa = dt * (m_o - m_out);
        dUa = dt * (m_o*(xTp-To)*Cp-m_out*(xTa-To)*Cp-Q_loss);
        xMg = xMg + dMg;
        xMd = xMd + dMd;
        xTb = xTb + dTb;
        xMa = xMa + dMa;
        xUa = xUa + dUa;
        xTime = dt/60.0 + xTime;
    }
}

/*****************************************************************
 * FUNCTION: adsorb_2                                            *
 * PURPOSE:  Simulate the operation of Desiccant bed #3, the     *
 *           blower, heat exchanger and sorbent bed #4.          *
 *           This provides for later simulation of component     *
 *           failures.                                           *
 *****************************************************************/
void adsorb_2()
{
    Ccs_w = xMd_w / Mb;
   k_w = 0.1;
   Cco_w = k_w * Ccs_w;
    xConc_w = Cco_w * 10000.0;
    Ccs = xMd / Mb;
   k_s = 0.1;
   Cco = k_s * Ccs;
    xConc = Cco * 1.0E6;
    dt = 0.1;
    for (dummy=1;dummy<=5*60/dt;dummy++){
                    /** desiccant bed equations **/
        dT_des = dt * ((M_flow * Cpa * (Ti - xT_des)) +
                 (M_flow * Sc_w * (Cci_w - Cco_w))) / (Mb * Cvs);
        dMd_w = M_flow * (Cci_w - Cco_w) * dt;
        xT_des = xT_des + dT_des;
        xMd_w = xMd_w + dMd_w;
        Ccs_w = xMd_w / Mb;
        Cco_w = k_w * Ccs_w;
        xConc_w = Cco_w * 10000.0;
            /** fan and heat exchanger "equations" **/
        xT_fan = xT_des + 6.81;
        xT_hx = xT_des - 50.0;       /** this is a kludge **/
                    /** sorbent bed calculations **/
        dT_sorb = ((M_flow * Cpa * (xT_hx - xT_sorb)) +
                  (M_flow * Sc * (Cci - Cco))) * dt / (Mb * Cvs);
        dMd = M_flow * (Cci - Cco) * dt;
        xT_sorb = xT_sorb + dT_sorb;
```

```
        xMd = xMd + dMd;
        Ccs = xMd / Mb;
        Cco = k_s * Ccs;
        xConc = Cco * 1.0E6;
        xTime = xTime + dt/60.0;
    }
}

/*****************************************************************
 * FUNCTION: desorb_2                                            *
 * PURPOSE:  Simulate the operation of sorbent bed #2, CO2       *
 *           pump and CO2 accumulator.                           *
 *           This provides for later simulation of component     *
 *           failures.                                           *
 *****************************************************************/
void desorb_2()
{
    for(dummy=1;dummy<=5*60/dt;dummy++){
        if (xTb > 330.0)  /** bang-bang bed temp. controller **/
            Power=0.0;
        else Power = 1000.0;
        Pg = xMg * R * xTb / Vg;
        Pb = k1 * xMd / Mb * (xTb - Tref);
        xPp = xMa * R * xTa / Va;
        xTp = xTb*pow((xPp/Pg),(1.0-1.0/k));
        xTa = xUa / (xMa * Cp) + To;
        m_o = w*k3*Pg/R/xTb;
        if (xMa < 0.002)    /** bang-bang flow out control **/
            m_out = 0.0;
        else
            m_out = 0.75 * m_o;
        if (xTa > 600.0)    /** bang-bang accumulator cooler **/
            Q_loss=k4*(xTa - 300);
        else
            Q_loss=0.0;
        dMd = (Pg - Pb) * k2 * dt;
        dMg = ((Pb - Pg) * k2 - (w * Pg * k3)/R/xTb) * dt;
        dTb = (dMd * Sc / dt + Power)/Mb/Cvb * dt;
        dMa = dt * (m_o - m_out);
        dUa = dt * (m_o*(xTp-To)*Cp-m_out*(xTa-To)*Cp-Q_loss);
        xMg = xMg + dMg;
        xMd = xMd + dMd;
        xTb = xTb + dTb;
        xMa = xMa + dMa;
        xUa = xUa + dUa;
        xTime = dt/60.0 + xTime;
    }
}
/*****************************************************************
 * FUNCTION: write_clips                                         *
 * PURPOSE:  Write data to CO2REMOV.SEN for CLIPS to analyze.    *
 * INPUTS:   time -- current time in half cycle                  *
 *           Cci  -- concentration of CO2 at assembly inlet      *
 *           CCo  -- concentration of CO2 at assembly outlet     *
 *           xMa  -- mass of CO2 in accumulator                  *
 *           m_out -- mass CO2 released to reduction assembly    *
 * OUTPUTS:  none.                                               *
 *****************************************************************/
void write_clips()
{
    clips = fopen("co2remov.sen","w");
    fprintf(clips,"(sensor CO2_REMOVAL INLET_CO2 %f kg/kg %f)\n"
```

```
                ,Cci,time);
    fprintf(clips,"(sensor CO2_REMOVAL OUTLET_CO2 %f kg/kg %f)\n"
                ,Cco,time);
    fprintf(clips,"(sensor CO2_REMOVAL CO2_ACCUMULATOR %f kg %f)\n"
                ,xMa,time);
    fprintf(clips,"(sensor CO2_REMOVAL CO2_DEMAND %f kg %f)\n"
                ,m_out,time);
    fclose(clips);
}
/*************************************************************
 * FUNCTION: write_model                                     *
 * PURPOSE:  Write data to CO2REMOV.DAT for re-starting model *
 * INPUTS:   time -- current time in half cycle              *
 *           xMg  -- mass of gas in void space               *
 *           xMd  -- mass of gas in sorbent material         *
 *           xTb  -- temperature of bed and gas desorbed     *
 *           xUa  -- internal energy in CO2 accumulator      *
 *           xMa  -- mass of CO2 in accumulator              *
 *           xT_sorb -- temperature of sorbent bed           *
 *           xT_des -- temperature of desiccant bed          *
 *           xMd_w -- mass of H2O in desiccant bed           *
 *           xTa -- temperature of accumulator               *
 *           xTp -- temperature of pump outlet               *
 *           xPp -- pressure of pump outlet                  *
 *           w -- pump speed                                 *
 *           power -- power applied to sorbent bed heaters   *
 *           pair -- which pair of beds is operating         *
 *           Cci -- incoming concentration of CO2            *
 *           Cci_w -- incoming concentration of CO2          *
 * OUTPUTS:  none.                                           *
 *************************************************************/
void write_model()
{
    model = fopen("co2remov.dat","w");
    fprintf(model,"%d %f %f %f %f %f\n",pair,time,xMg,xMd,xMd_w,xMa);
    fprintf(model,"%f %f %f %f %f %f\n",xTb,xT_sorb,xT_des,xUa,w,Power);
    fprintf(model,"%f %f %f %f %f",xTa,xTp,xPp,Cci,Cci_w);
    fclose(model);
}

/*************************************************************
 * The two halves of the assembly could be modeled using only *
 * two functions, but by using four functions, the model can  *
 * by used to simulate component failures and valve change    *
 * sequencing.                                               *
 *************************************************************/
main()
{
     read_model();          /** get info to start model **/
     read_commands();       /** read CLIPS command file **/
     printf("\nOperating in %s mode.",mode);
     decode_commands();     /** decode CLIPS command into model
                                parameters **/

     /*********************************************************
     *   Entry point for restarting model.                    *
     *                                                        *
     *   First check to see if mode change should be made, and *
     *   make if necessary.  Then using model data, run model *
     *   for five minutes, writing sensor data and model data *
     *   and quit to batch file.                              *
     *********************************************************/
```

```
    if (time >= tstop) {        /** Determine which pair of functions **/
        time = 0.0;             /** will be operating.  If the model  **/
        if (pair == 1)          /** is about to change half cycles,   **/
            pair = 2;           /** make the necessary arrangements.  **/
        else
            pair = 1;
    }
    printf("\nPair %d is operating...",pair);

    if (pair == 1) {            /** Run the model for five minutes and **/
        adsorb_1();             /** output data to the necessary files.**/
        desorb_1();
        write_clips();
        write_model();
      }
    else {                      /** Run the model for five minutes and **/
        adsorb_2();             /** output data to the necessary files.**/
        desorb_2();
        write_clips();
        write_model();
    }
    return NORMAL;              /** main() should return a value to DOS **/
}
```

APPENDIX 2B
CO_2 REDUCTION ASSEMBLY PROGRAM

```
/*********************************************************
*                                                        *
*    Program Name:     Sabatier Methanation Reactor      *
*    Author:                 Jeff Newell                 *
*    Date:                    2/15/91                    *
*  Description:      This program will determine and     *
*                   set a reactor volume depending on    *
*                   the initial carbon dioxide flow      *
*                   rate entered by the user.  It will   *
*                   then let the user enter different    *
*                   flow rates to see how conversion     *
*                   is affected.  It will also give      *
*                   outputs of CO2 and H2 not reacted    *
*                   as well as H2O and CH4 produced.     *
*********************************************************/

/*----- INCLUDE THE NECESSARY HEADER FILES -----*/

#include <stdio.h>
#include <math.h>

/*----- DEFINE CONSTANTS USED -----*/

#define A  1.112393E6     /* catalyst activity constant */
#define E  8553           /* catalyst activity constant = Ea/R */

/*----- START MAIN FUNCTION -----*/

main()
{
   /*----- INITIALIZE VARIABLES -----*/
   int     doanother=1,  /* variable for doing another run */
           index;              /* index for arrays */

   double  actconv,            /* actual conversion rate */
           actflow,            /* actual flow rate of CO2 */
           ch4prod,            /* CH4 produced */
           co2notreac,       /* CO2 not reacted */
           co2rate,          /* incoming flow of CO2 */
           conversion,       /* conversion rate */
           h2notreac,          /* H2 not reacted */
           h2oprod,            /* H2O produced */
           integral,         /* mass balance integral */
           keq,                /* equilibrium constant */
           part1keq,         /* part of keq */
           part2keq,         /* part of keq */
           part1reac,        /* part of reaction rate */
           part2reac,        /* part of reaction rate */
           part3reac,        /* part of reaction rate */
           sqrtemp,          /* square of temperature */
           value,            /* used for integration */
           volcent1,         /* volume of sabvol in cubic cent. */
           volcent2,         /* volume of sabatier in cubic cnt */
           volgal;           /* volume of Sabatier in gallons */

   double  reac[550],          /* rate of reaction */
           pco2[550],        /* partial pressure of CO2 */
           pch4[550],        /* partial pressure of CH4 */
```

```
            ph2o[550],        /* partial pressure of H2O */
            ph2[550],            /* partial pressure of H2 */
            sum[550],         /* integral */
            temp[550];        /* temperature of reaction */


   /*------------------ ENTER INPUT VALUES ---------------------*/

   /* This section will let the user input a flow rate of CO2 */

   printf("Enter initial flow rate of carbon dioxide (in mol/sec):       \n");
   scanf("%lf",&co2rate);

   /*  This section calculates the volume required to achieve
       maximum conversion in cubic cent. and gallons */

   volcent1 = 1.652E7 * co2rate;
   volgal = volcent1 * 2.642E-4;

   /* This section lets the user know the volume necessary
       to achieve maximum conversion for his flow rate */

   printf("In order to achieve a maximum conversion rate of 0.515,       \n");
   printf("the volume of the reactor must be at least %5.2lf
\n",volcent1);
   printf("cubic centimeters, or in other words, %2.2lf gallons.
\n",volgal);
   printf("\n\n\n");

   /* This section sets the volume of the reactor to the value
          above */

   printf("The volume of the reactor is now set at %2.2lf gallons.
\n",volgal);
   printf("\n\n");


   while(doanother==1)
   {

      /* This section lets the user input his own volume */

      printf("You may now enter a test flow rate of carbon \n");
      printf("dioxide to see the exact conversion rate for \n");
      printf("the volume set above. \n");
      printf("\n");
      printf("Enter test carbon dioxide flow rate (in mol/sec):
\n");
      scanf("%lf",&actflow);

      /* This section determines what the value of the mass balance
          integral must be for the user's volume and flow rate */

      integral = volcent1 / actflow;

      /* This section compares the value of the mass balance
integral to the value it should be for maximum conversion. If the integral is
larger than 1.652E7, then the volume of the reactor is large enough to achieve
maximum conversion for the user's flow rate.  If this is the case, this
section sets the actual conversion rate to 51.5% and determines the temperature
of reaction. */
```

```
if(integral>=1.652E7)
{
   actconv = .515;
   index = 515;
   temp[index] = 1088 * actconv + 297.59;
}

/* This section also compares the value of the mass balance integral to the
maximum value.  In this case, if the integral is smaller than 1.652E7, then the
reactor is not large enough to achieve maximum conversion.  If this is the case,
this section evaluates the mass balance integral to determine the conversion rate
for the inputted flow rate of CO2. */

if(integral<1.652E7)
{

   /* Initialize values */

   conversion = 0.0;
   index = 0;
   actconv = 0.0;
   sqrtemp = 0.0;
   value = 0.0;

   for(conversion=.001;conversion<=.515;conversion+=.001)
   {
      index = conversion * 1000;
      temp[index] = 1088 * conversion + 297.59;
      /* part1&2keq used to calculate keq */
      sqrtemp = temp[index]*temp[index];
      part1keq = 5600/sqrtemp + 34633/temp[index];
      part2keq = -16.4*(log(temp[index])) + 0.00557*temp[index];
      keq = exp((1.0/1.987)*(part1keq + part2keq) + 33.165);
      /* partial pressures of reactants and products */
      pco2[index] = 0.2*((1-conversion)/(1-.4*conversion));
      pch4[index] = 0.2*(conversion/(1-0.4*conversion));
      ph2o[index] = 0.2*((2*conversion)/(1-0.4*conversion));
      ph2[index] = 0.2*((4-4*conversion)/(1-0.4*conversion));
      /* part1,2&3reac used to calculate reaction rate */
      part1reac = pow(pco2[index],0.25)*ph2[index];
      part2reac = pow(pch4[index],0.25)*pow(ph2o[index],0.5)/keq;
      part3reac = A/temp[index];
      reac[index] = part3reac*(part1reac-part2reac)*exp(-E/temp[index]);

      /* This line multiplies the integrand by the time step      */
         value = (1/reac[index])*.001;
     /* This line adds the values above to determine conversion */
      sum[index] = sum[index-1] + value;
     /* determines if integral has been computed */
     if(sum[index]>=integral)
      {
         actconv=conversion;
         conversion=.515;
      }
   }
}
/* This section prints output values */
printf("Actual rate of conversion = %4.4lf \n\n",actconv);
printf("Final temperature of reaction = %6.2lf K
\n\n",temp[index]);
co2notreac = 44.0*actflow*(1-actconv);
printf("Amount of CO2 not reacted = %4.4lf grams/sec.
```

```
 \n",co2notreac);

      h2notreac = 2.0*4.0*actflow*(1-actconv);
      printf("Amount of H2 not reacted = %4.4lf grams/sec.
  \n",h2notreac);
      h2oprod = 18.0*2.0*actflow*actconv;
      printf("Amount of H2O produced = %4.4lf grams/sec. \n"
 ,h2oprod);
      ch4prod = 16.0*actflow*actconv;
      printf("Amount of CH4 produced = %4.4lf grams/sec.
    \n",ch4prod);
      printf("If you wish to do another analysis, enter 1, else 0.
     \n");
      scanf("%d",&doanother);
  }
}
/*--------------- END OF FILE -------------------*/
```

APPENDIX 2C
OXYGEN GENREATION ASSEMBLY PROGRAM

```
/*****************************************************************
 * SOURCE FILE:   OGA.C
 *
 * DESCRIPTION:  This program models the OGA electrolysis unit,
 *               H20 tanks, and the H2 and O2 output from the
 *           electrolysis unit.
 *
 * USES:         main()
 *
 * AUTHORS:      Original in fortran by Cark Perkins, David Kley,
 *                 and Frank Lunkwitz.
 *
 *           Rewrite in C by Kevin Forssberg and William Biehl
 *
 * DATE:          March 1, 1991
 *****************************************************************/

/*************
 * NOTE TO ROBERT YOUNG AND HAL KRAUS:
 *      -- KEVIN AND BILL COULD NOT FIGURE OUT HOW TO STOP
 *         AND RESTART THE PROGRAM EVERY 5 MIN.  THEREFORE,
 *         THIS PROGRAM DOES NOT HAVE THIS CAPABILITY.
 ************/

#include  <stdio.h>
#include  <math.h>

#define    F      96487
#define    FULL       35 /* A FULL WATER TANK IS PRESET AT 35 MOLES */
#define    EMPTY   3  /* H20 AND A EMPTY TANK IS AT 3 MOLES H20  */
#define    X       18 /* X IS THE NUMBER OF CELLS IN THE ELECTROL-
                       YSIS UNIT                                  */

FILE       *outfile;    /* DECLARAITION OF OUTPUT FILE             */

main()
{
   int         I=0,     /* ELECTROLYSIS UNIT CURRENT                */
           t=0,     /* INCREMENTING VARIABLE IN FOR LOOPS       */
           TIME=0, /* TIME INITIAL SET POINT                    */
           LEVEL=0;/* THIS IS THE DISCRETE INITIAL SETTING OF
                      THE WATER TANK WHERE 1 INDICATES FULL
                      AND 0 INDICATES EMPTY                     */


   double    H2OUT,       /* MOLES OF H2 OUT OF ELECTROLYSIS UNIT*/
          O2OUT,       /* MOLES OF O2 OUT OF ELECTROLYSIS UNIT*/
          H2TANK=0.0, /* INITIAL SETTING OF H2 TANK            */
          O2TANK=0.0, /* INITIAL SETTING OF O2 TANK            */
          H2O,         /* AMOUNT OF WATER IN SYSTEM             */
          TEMP,        /* OPERATING TEMP OF ELECTROLYSIS UNIT */
          O2_OUTTAKE_TEMP, /* LINEAR CONVERTION FOR TEMP OF O2
                              OUT TO THE CABIN                  */
   /*****************************************************/
   /*PRAMETERS NOT CALCULATED BY MODEL BUT NEEDED FOR EXPERT     */
   /*SYSTEM.  -- TO VARY EXPERT SYSTEM OUTPUTS CHANGE THESE      */
   /*          VARIABLES AT THIS POINT ONLY.                    */
```

```
    /*************************************************************/
        H2O_FEEDLINE_PRESSURE=25.0,  /* H2O PRESSURE TO ELECTROL-
                                YSIS UNIT                  */
        H_O_DEFERENTIAL_PRESSURE=3.0,/* DERERENTIAL PRESSURE OF
                                O2 AND H2 OUTPUT LINES  */
        CELL_VOLTAGE=1.67;           /* INDIVIDUAL CELL VOLTAGE
                                IN ELECTROLYSIS UNIT    */


    outfile=fopen("OGA.DAT","w"); /* OPEN DATAFILE "OGA.DAT"     */

    H2O = FULL;
    for(I=15; I<=28; I++)
        {
        TIME++;
        H2OUT= (double) I/F*X*60.0;
        O2OUT=H2OUT/2.0;
        H2TANK=H2TANK+H2OUT;
        O2TANK=O2TANK+O2OUT;
        H2O=H2O-H2OUT-O2OUT;
        TEMP=((-0.0391*I*I) + (4.38*I) + 67);
        O2_OUTTAKE_TEMP=TEMP-68;
        if (H2O <= EMPTY)
            {
            H2O=FULL;
            LEVEL=0;
            }
        else
            LEVEL=1;
fprintf(outfile,"(SENSOR OGA H2O_FEEDLINE_PRESSURE %4.2f PSIG %2.0d)\n",
    H2O_FEEDLINE_PRESSURE,TIME);
fprintf(outfile,"(SENSOR OGA H2O_PHOTOCELL %d discrete %2.0d)\n",
    LEVEL,TIME);
fprintf(outfile,"(SENSOR OGA H/O_DEFERENTIAL_PRESSURE %4.2f PSIG %2.0d)\n",
    H_O_DEFERENTIAL_PRESSURE,TIME);
fprintf(outfile,"(SENSOR OGA O2_OUTTAKE_TEMP %5.3f F %2.0d)\n",
    O2_OUTTAKE_TEMP,TIME);
fprintf(outfile,"(SENSOR OGA CELL_VOLTAGE %4.2f volts %2.0d)\n\n\n",
    CELL_VOLTAGE,TIME);
        }
    for(t=1; t<200; t++)
        {
        TIME++;
        H2OUT=H2OUT;
        O2OUT=O2OUT;
        H2TANK=H2TANK+H2OUT;
        O2TANK=O2TANK+O2OUT;
        H2O=H2O-H2OUT-O2OUT;
        if(H2O <= EMPTY)
            {
            H2O = FULL;
            LEVEL = 0;
            }
        else
            LEVEL = 1;
fprintf(outfile,"(SENSOR OGA H2O_FEEDLINE_PRESSURE %4.2f PSIG %2.0d)\n",
    H2O_FEEDLINE_PRESSURE,TIME);
fprintf(outfile,"(SENSOR OGA H2O_PHOTOCELL %d discrete %2.0d)\n",
    LEVEL,TIME);
fprintf(outfile,"(SENSOR OGA H/O_DEFERENTIAL_PRESSURE %4.2f PSIG %2.0d)\n",
    H_O_DEFERENTIAL_PRESSURE,TIME);
fprintf(outfile,"(SENSOR OGA O2_OUTTAKE_TEMP %5.3f F %2.0d)\n",
```

```
    O2_OUTTAKE_TEMP,TIME);
fprintf(outfile,"(SENSOR OGA CELL_VOLTAGE %4.2f volts %2.0d)\n\n\n",
    CELL_VOLTAGE,TIME);
        }
    fclose(outfile);  /* CLOSE OUTPUT FILE "OGA.DAT" */
}
```

APPENDIX 2D
AIR REVITALIZATION SYSTEM PROGRAM

Contained within this appendix is information about the compliling of the computer modeling groups into one. Information is given for the benefit of giving next semester students background on the computer program and why it operates the way it does. Also,included in this appendix is logon procedures for operating the program on the AT&t lab computers on second floor of Durland.

Logon on Procedures

To log onto the 3bl's, AT&T lab -- second floor Durland:

login: **<student>** ----type student @ login command and return
----use mouse to select **Full screen unix**
To run program: ----**insert floppy w/ program**
----program name is **AR_model.c**
To look at program --Type: **mread -t ar_model.c ar_model.c**
To compile program --Type: **cc -G -I/usr/local/include ar_model.c -lksu -levdi -lm -o run**
To run program ----Type: **run**

3bl's packet, available from Ruth Dyer, gives editor commands, logon procedures, etc.

Assumptions and Conditions of Program

Assumptions of each individual portion of this modeling program can be examined in detail from previous sections. Each model has its' input and output parameters, which are discussed in their individual sections.

This program takes all three modeling programs and puts them into one working program, each dependent on the other. A few things have been changed and added to complete this task.

1. The CO2 is not delivered from the Removal to the Reduction until the sorbent bed is reversed and CO2 is being removed from the bed.

2. Certain values are inputs to the system and are asked for when running the program.

3. Between plots of output, any key has to be hit to continue to see next plot.

4. A tank is inserted between the CO2 Reduction and the OGA systems. This tank is initially set and has two inputs and one output: in--CO2 Red. and Hygiene Water
out-- OGA system.

5. The OGA system operates a little differently in this program than it does seperately.
a. The system is already at steady state by the time the CO2 Red. receives any CO2 from the CO2 Rem.
b. The time interval is different in this program, it is made to better match the time scale that is seen
from the CO2 Removal.
c. Since the small H2O tanks in the OGA system will not empty in the time period in which this program operates, the water is assumed to come directly from the tank that we created between the CO2 Red.

and the OGA.

Modeling Program

The following is a listing of the file that was created and edited to fit all three working computer models of the AR system.

```
/*****************************************************************
 *                                                               *
 *  SOURCE FILE:    MODEL4.C                                     *
 *                                                               *
 *  DESCRIPTION:  This program models the CO2 reduction system *
 *                during its sorbtion and desorption half      *
 *                cycles.  CO2 conversion and O2 generation    *
 *                by the REDUCTION and OGA subsystems are also *
 *                included; however, the REDUCTION model is    *
 *                not time dependent.  We have arranged it so  *
 *                its outputs appear time dependent by assuming*
 *                steady-state operation at all times.         *
 *                                                             *
 *  USES:         main()                                       *
 *                simple_plot()     (S. Dyer)                  *
 *                                                             *
 *  AUTHORS:      Robert Young, Terry Hon, Joel Sommer,        *
 *                Carl Perkins                                 *
 *                                                             *
 *  LAST REVISION:29 April 1991                                *
 *                                                             *
 *****************************************************************/

#include <stdio.h>
#include <math.h>
#include "p_plot.h"
#include "simple_plot.h"
#define NUM_POINTS 100

void input();
void co2removal();
void co2reduction();
void oga();
void h2otank ();


int dummy;
long int i,q;
double Time[100],     /** time array **/
       xTime,
       Mg[100],       /** mass of gas in void space **/
       xMg,
       Md[100],       /** mass of gas in sorbent material **/
       xMd,
       Tb[100],       /** temperature of bed and gas desorbed **/
       xTb,
       Ua[100],       /** internal energy of accumulator **/
       xUa,
       Ma[100],       /** mass of CO2 in accumulator **/
       xMa,
       Pp[100],       /** output pressure of pump **/
       xPp,
       Tp[100],       /** output temeprature of pump **/
       xTp,
```

```
        Ta[100],        /** temperature of accumulator **/
        xTa,
        Conc_w[100],    /** relative humidity of outgoing air **/
        xConc_w,
        Ccs_w,          /** conc. of H2O on sorbent (kgH2O/kgair) **/
        M_flow,         /** mass flow rate of air (kg/sec) **/
        Ti,             /** incoming air temperature (K) **/
        T_des[100],     /** temp. of air leaving desicant bed (K) **/
        xT_des,
        Cci_w,          /** conc. of H2O entering bed (kgH2O/kgair) **/
        Cco_w,          /** conc. of H2O leaving bed (kgH2O/kgair) **/
        Md_w[100],      /** mass of H2O on sorbent (kg) **/
        xMd_w,
        Ccs,            /** conc. CO2 in sorbent (kgCO2/kg air) **/
        Cci,            /** conc. CO2 entering bed (kg CO2/kg air) **/
        Cco,            /** conc. CO2 leaving bed (kg CO2/ kg air) **/
        Conc[100],      /** conc. CO2 leaving bed (ppm) **/
        xConc,
        T_sorb[100],    /** temp. or air leaveing sorbent bed (K) **/
        xT_sorb,
        Fa[100],        /** CO2 flow exiting the REDUCTION ass. (kmoles/s)**/
        F_water[100];   /** H2O flow exiting REDUCTION ass. (kmoles/s)**/


double  Pg,             /** CO2 pressure in void space **/
        Pb,             /** pump exit pressure **/
        m_o,            /** mass flow rate out of bed   dMg/dt **/
        xm_out,
        m_out[100],     /** assembly exiting flow rate (bang-bang) **/
        w,              /** angular velocity of pump (rad/sec)  **/
        Q_loss,         /** accumulator heat loss (bang-bang) **/
        Tref,           /** reference temp. of desorbent bed  **/
        k_w,            /** ratio of Cco_w/Ccs_w **/
        k_s,            /** ratio of Cco/Ccs **/
        to_red,         /** CO2 flow rate to REDUCTION assembly (kg/s) **/
        conv,           /** conversion factor for REDUCTION **/
        F,              /** Faraday's constant for OGA **/
        X,              /** number of electrolysis cells in OGA **/
        H2OUT[100],     /** OGA hydrogen production rate **/
        O2OUT[100],     /** OGA oxygen production rate **/
        a,              /** OGA current in amps **/
        minimum,        /** minimum level in h2otank **/
        hyg_H2O,        /** H2O flow rate into h2o_tank from HYGIENE **/
        oga_H2O[100],   /** H2O flow rate into h2o_tank from OGA **/
        elapse[100],    /** time array for OGA and h2otank models **/
        level[100];     /** level in h2otank (kmoles)**/

double  Power=1000.0,     /** power input to desorbing CO2 bed **/
        Mb=20.0,          /** mass of sorbent material (kg) **/
        Cpa=1003.0,       /** heat capacity of air (J/kgK) **/
        Cvs=844.0,        /** heat capacity of sorbent (J/kgK) **/
        Sc_w=2700.0E3,    /**     heat of sorption of H2O
                                  into sorbent (J/kgK)        **/

        Cvb=657.0,
        Cp = 846.0,
        R=0.1889,
        To=0.0,  /** reference temperature for internal energy **/
        Vg=1.0,
        Va=2.0,
        k1=0.7,
        k2=.0006,
        k3=.0023,
        k4= 2.0,      /** accumulator heat transfer coef. (Watts/K) **/
        k = 1.29,
        Sc=951E3;     /** CO2 heat of sorption into sorbent (J/kgK) **/
```

```
double dt,
       dTb,
       dT_des,
       dT_sorb,
       dMd_w,
       dMg,
       dMd,
       dMa,
       dUa;
main()
{
input();          /** subroutine which asks for user input **/
co2removal();     /** subroutine modelling the REMOVAL assembly **/
co2reduction();   /** subroutine modelling the REDUCTION assembly **/
oga();            /** subroutine modelling the OGA assembly **/
h2otank();        /** subroutine modelling the H2O tank serving OGA **/
return(0);
}
                 /** Inputting of parameters by user **/
void input()
{
system("clear");
printf("\nInput mass flow rate of air, M_flow (kg/s) : ");
scanf("%lf",&M_flow);
printf("\nInput H2O concentration entering assembly,");
printf(" Cci_w (kgH2O/kg air) : ");
scanf("%lf",&Cci_w);
printf("\nInput concentration of CO2 entering assembly,");
printf(" Cci (kgCO2/kg air): ");
scanf("%lf",&Cci);
printf("\nInput temperature of incoming air, Ti (K) : ");
scanf("%lf",&Ti);
printf ("\nInput angular velocity of pump (radians/second) : ");
scanf ("%lf",&w);
printf ("\nInput desired bed reference temperature, Tref (K) : ");
scanf ("%lf",&Tref);
printf("\nInput desired conversion of CO2 (0.52 max) : ");
scanf("%lf",&conv);
}
void co2removal()
{
Time[0] = 0.0;         /** absolute start time in minutes **/
xTime = 0.0;
                       /** initial conditions for desiccant bed **/
   Md_w[0] = xMd_w = 0;
   T_des[0] = xT_des = Ti;
   Ccs_w = Md_w[0] / Mb;
   k_w = 0.1;
   Cco_w = k_w * Ccs_w;
   Conc_w[0] = xConc_w = Cco_w * 10000.0;
                       /** initial conditions for sorbent bed **/
   Md[0] = xMd = 0.0,
   T_sorb[0] = xT_sorb = Ti;
   Ccs = Md[0] / Mb;
   k_s = 0.1;
   Cco = k_s * Ccs;
   Conc[0] = xConc = Cco * 1.0E6;

   dt = 1.0;
  printf("\n\nWorking desiccant and sorbent bed forward cycle...\n\n");

             /************************************************
              *                                              *
              *   Desiccant bed and Sorbent bed calculations  *
              *                                              *
              ************************************************/
  i = 1;
```

```
while (i<100) {
  for (dummy=1;dummy<=33;dummy++){

                    /** desiccant bed equations **/
       dT_des = dt * ((M_flow * Cpa * (Ti - xT_des)) +
                (M_flow * Sc_w * (Cci_w - Cco_w))) / (Mb * Cvs);
    dMd_w = M_flow * (Cci_w - Cco_w) * dt;
    xT_des = xT_des + dT_des;
    xMd_w = xMd_w + dMd_w;
    Ccs_w = xMd_w / Mb;
    Cco_w = k_w * Ccs_w;
    xConc_w = Cco_w * 10000.0;
                  /** sorbent bed calculations **/
    dT_sorb = ((M_flow * Cpa * (xT_des - xT_sorb)) +
              (M_flow * Sc * (Cci - Cco))) * dt / (Mb * Cvs);
    dMd = M_flow * (Cci - Cco) * dt;
    xT_sorb = xT_sorb + dT_sorb;
    xMd = xMd + dMd;
    Ccs = xMd / Mb;
    Cco = k_s * Ccs;
    xConc = Cco * 1.0E6;
    xTime = xTime + dt/60.0;
  }
  Time[i] = xTime;
  T_des[i] = xT_des;
  Md_w[i] = xMd_w;
  Conc_w[i] = xConc_w;
  T_sorb[i] = xT_sorb;
  Md[i] = xMd;
  Conc[i] = xConc;
  i = i + 1;
}

       /*****************************************************
        *                                                   *
        *  Plotting routine for Desiccant and Sorbent beds  *
        *                                                   *
        *****************************************************/

 printf("\n\nDone...");
 printf("\n\nPress a number and <Return> to plot...");
 scanf("%d",&dummy);
            /** Simple plot calls for desiccant bed data **/
 simple_plot(DISPLAY,i,Time,T_des,"Time","minutes","Temperature",
             "degrees Kelvin",
             "Temperature of Desicant Bed (T_des[i])",1,CURVE);
 scanf("%d",&dummy);
 simple_plot(DISPLAY,i,Time,Md_w,"Time","minutes","Mass H2O","kg",
             "Mass H2O in Sorbent Bed Material (Md_w[i])",1,CURVE);
 scanf("%d",&dummy);
 simple_plot(DISPLAY,i,Time,Conc_w,"Time","minutes",
            "Relative Humidity","%",
            "Relative Humidity of Air Leaving Desicant Bed",
             1,CURVE);
 scanf("%d",&dummy);
            /** Simple plot calls for sorbent bed data **/
 simple_plot(DISPLAY,i,Time,T_sorb,"Time","minutes","Temperature",
             "degrees Kelvin","Temp. of Air Leaving Sorbent Bed",
             1,CURVE);
 scanf("%d",&dummy);
 simple_plot(DISPLAY,i,Time,Md,"Time","Minutes","Mass CO2","kg",
             "Mass CO2 in Sorbent Bed Material (Md[i])",1,CURVE);
 scanf("%d",&dummy);
 simple_plot(DISPLAY,i,Time,Conc,"Time","minutes","Concentration",
             "ppm","Conc. CO2 Leaving Sorbent Bed",1,CURVE);
 scanf("%d",&dummy);
 system("clear");             /** clear the screen **/
```

```
 printf("\n\nWorking on desorbing cycle...");
            /** initial conditions for desorbing cycle **/
 Time[0] = xTime = 62;  /** reset array for this cycle **/
 Md[0] = xMd = Md[99];  /** reset array for this cycle **/
 Tb[0] = xTb = T_sorb[99]; /** reset array for this cycle **/
 Mg[0] = k1 * Vg * Md[0] * (Tb[0] - Tref) / (Mb * R * Tb[0]);
    xMg = Mg[0];
 Ua[0] = xUa = 407.34;
 Ma[0] = xMa = 0.002;
 Ta[0] = xTa = 300.0;
 Pp[0] = xPp = 0.0567;
 Tp[0] = xTp = 300.0;

                /***********************************************
                *                                              *
                *      Calculations for Desorbing cycle        *
                *                                              *
                ***********************************************/

i = 1;
while(i<100){
  for(dummy=1;dummy<=28;dummy++){

    if (xTb > 330.0)  /** bang-bang bed temp. controller **/
       Power=0.0;
    else Power = 1000.0;

    Pg = xMg * R * xTb / Vg;
    Pb = k1 * xMd / Mb * (xTb - Tref);
    xPp = xMa * R * xTa / Va;
    xTp = xTb*pow((xPp/Pg),(1.0-1.0/k));
    xTa = xUa / (xMa * Cp) + To;
    m_o = w*k3*Pg/R/xTb;

    if (xMa < Ma[0])    /** bang-bang flow out control **/
 xm_out = 0.0;
    else
 xm_out = 0.85 * m_o;  /** CO2 flow rate to REDUCTION **/

    if (xTa > 860.0)    /** bang-bang accumulator cooler **/
       Q_loss=k4*(xTa - 300);
    else Q_loss=0.0;

    dMd = (Pg - Pb) * k2 * dt;
    dMg = ((Pb - Pg) * k2 - (w * Pg * k3)/R/xTb) * dt;
    dTb = (dMd * Sc / dt + Power)/Mb/Cvb * dt;
    dMa = dt * (m_o - xm_out);
    dUa = dt * (m_o*(Tp[i-1]-To)*Cp-xm_out*
               (Ta[i-1]-To)*Cp - Q_loss);

    xMg = xMg + dMg;
    xMd = xMd + dMd;
    xTb = xTb + dTb;
    xMa = xMa + dMa;
    xUa = xUa + dUa;
    xTime = dt/60.0 + xTime;
  }

  Time[i] = xTime;
  Mg[i] = xMg;
  Md[i] = xMd;
  Tb[i] = xTb;
  Ma[i] = xMa;
  Ua[i] = xUa;
  Ta[i] = xTa;
  Tp[i] = xTp;
  Pp[i] = xPp;
```

```
    m_out[i] = xm_out;
    i = i + 1;
  }

   printf("\nPress number and return to begin plotting...");
   scanf("%d",&dummy);

         /** SIMPLE PLOT CALLS FOR DESORBENT CYCLE DATA **/

   simple_plot(DISPLAY,i,Time,Ma,"Time","minutes",
               "Mass CO2","kg","Mass CO2 in Accumulator (Ma[i])"
               ,1,CURVE);

   scanf("%d",&dummy);
   simple_plot(DISPLAY,i,Time,Ua,"Time","minutes",
                  "Internal Energy","J",
              "Internal Energy in Accumulator (Ua[i])"
              ,1,CURVE);

   scanf("%d",&dummy);
   simple_plot(DISPLAY,i,Time,Tp,"Time","minutes", "Temperature",
              "K","Temperature of Pump Outlet (Tp[i])", 1,CURVE);

   scanf("%d",&dummy);
   simple_plot(DISPLAY,i,Time,Ta,"Time","minutes","Temperature",
              "K","Temperature of Accumulator (Ta[i])",1,CURVE);

   scanf("%d",&dummy);
   simple_plot(DISPLAY,i,Time,Pp,"Time","minutes","Pressure",
              "kPa","Outlet Pressure of Pump (Pp[i])",1,CURVE);
   scanf("%d",&dummy);
   printf("\nDone sending plots..");
   system ("clear");
}


   void co2reduction()
  {

   for (i=1; i<100;i++)
    {
      Fa[i] = m_out[i] /44.0 * (1-conv);
      F_water[i] =2/44.0 * m_out[i] * conv;
    }


      simple_plot(DISPLAY,i,Time,Fa,"Time","minutes","CO2 Flow Rate",
              "kmoles/s","CO2 Flow Rate Exiting REDUCTION Assembly (Fa[i])",
              1,CURVE);
      scanf("%d",&dummy);
      simple_plot(DISPLAY,i,Time,F_water,"Time","minutes","H2O Flow Rate",
              "kmoles/s",
              "H2O Flow Rate Exiting REDUCTION Assembly (F_water[i])",
              1,CURVE);
      scanf("%d",&dummy);
      printf("\nDone with REDUCTION...\n\n");
      system("clear");

}
     void oga()
     {
       F=96487.0;
       X=18.0;
       a=15.0;         /** initial current in electrolysis module **/
       elapse[0]=0.0; /** time array **/
       H2OUT[0]=0.0;
       O2OUT[0]=0.0;
```

```
  oga_H2O[0]=0.0;
  for (q=1;q<=13;q++)  /** this loop models transient operation **/
   {                   /** when the current is increasing w/time **/
     H2OUT[q]= a*X/F;
     O2OUT[q]= H2OUT[q]/2.0;
     oga_H2O[q]= H2OUT[q];
     a=a+1;
   }
  while (q<=46)      /** this loop models steady-state operation **/
   {                   /** when current is constant **/
     H2OUT[q]=a*X/F;
     O2OUT[q]=H2OUT[q]/2.0;
     oga_H2O[q]=H2OUT[q];
     elapse[q]=q;
     q=q+1;
   }
  simple_plot(DISPLAY,q,elapse,O2OUT,"Time","minutes","flow rate",
              "kmoles/s","Oxygen Production by OGA",1,CURVE);
  scanf("%d",&dummy);
  system ("clear");
  printf("\nDone with OGA...\n\n");
}

void h2otank()
{         /** models the water tank that the OGA draws from **/
  q=1;
  hyg_H2O=0.0;
  level[0]=65.0;
  while (q<=45)
   {
    minimum=50.0;
    if (level[q]>minimum)  /** checks hyg_H2O supply **/
      hyg_H2O=0.0;
    else hyg_H2O=0.1;              /** turns hyg_H2O supply on **/
    level[q]=level[q-1]-(60.0*oga_H2O[q])+
               (60.0*F_water[i]+hyg_H2O);
    q=q+1;
   }
  simple_plot(DISPLAY,q,elapse,level,"Time","minutes","tank level",
              "kmoles","h2otank level",1,CURVE);
  scanf("%d",&dummy);
  system ("clear");
  printf("\n\nDone with AR-Simulation...\n\n");
}
```

APPENDIX 3A
CO_2 REMOVAL ASSEMBLY KNOWLEDGE BASE

```
CO2 REMOVAL PRECOOLER OUTPUT AIR TEMPERATURE SENSOR     (RULE #1)

PARAMETER MODIFIER DEFINITIONS

(level: if COOLER_TEMP is X then TEMP is HI     100   212   500   900  F)
(level: if COOLER_TEMP is X then TEMP is NORMAL   0    32   100   212  F)
(level: if COOLER_TEMP is X then TEMP is LOW   -400  -100     0    32  F)

DEFINE CONDITIONS

(condition: if COOLER_TEMP is HI      then COOLER is FAIL     1)
(condition: if COOLER_TEMP is NORMAL  then COOLER is FAIL     1)
(condition: if COOLER_TEMP is LOW     then COOLER is FAIL     1)

DEFINE RESPONSES

(response: if COOLER is FAIL then CO2_REMOVAL_COMMAND is OFF   0.70)


CO2 REMOVAL BLOWER PRESSURE DIFFERENTIAL SENSORS   (RULE #2)

PARAMETER MODIFIER DEFINITIONS

(level: if BLOWER is X then BLOW_PRES is HI      18    35   100  100 psia)
(level: if BLOWER is X then BLOW_PRES is NORMAL   8    12    18   35 psia)
(level: if BLOWER is X then BLOW_PRES is LOW   -100  -100     8   12 psia)

DEFINE CONDITIONS

(condition: if BLOW_PRES is HI      then BLOWER is FAILED   0.2)
(condition: if BLOW_PRES is NORMAL  then BLOWER is FAILED   0.0)
(condition: if BLOW_PRES is LOW     then BLOWER is FAILED   1.0)

DEFINE RESPONSES

(response: if BLOWER is FAILED then CO2_REMOVAL_COMMAND is OFF   1)


CO2 PUMP POWER SENSOR                              (RULE #3)

PARAMETER MODIFIER DEFINITIONS

(level: if CO2_PUMP is X then PUMP_POWER is HI     400   600   800   1000 watt)
(level: if CO2_PUMP is X then PUMP_POWER is NORMAL  20   100   300    500 watt)
(level: if CO2_PUMP is X then PUMP_POWER is LOW      0     5    20    100 watt)

DEFINE CONDITIONS

(condition: if PUMP_POWER is HI      then CO2_PUMP is FAILED  1.0)
(condition: if PUMP_POWER is NORMAL  then CO2_PUMP is FAILED  0.0)
(condition: if PUMP_POWER is LOW     then CO2_PUMP is FAILED  0.4)

DEFINE RESPONSES

(response: if CO2_PUMP is FAILED then CO2_REMOVAL_COMMAND is OFF   1)


CO2 PUMP PRESSURE SENSOR                           (RULE #4)

PARAMETER MODIFIER DEFINITIONS

(level: if CO2_PUMP_PRES is X then PUMP_PRES is HI     50  100  500  900 psia)
(level: if CO2_PUMP_PRES is X then PUMP_PRES is NORMAL  2   10   30   75 psia)
(level: if CO2_PUMP_PRES is X then PUMP_PRES is LOW     0    0    2    5 psia)

DEFINE CONDITIONS

(condition: if PUMP_PRES is HI      then CO2_PUMP is FAILED  0.8)
(condition: if PUMP_PRES is NORMAL  then CO2_PUMP is FAILED  0.0)
(condition: if PUMP_PRES is LOW     then CO2_PUMP is FAILED  0.1)
```

```
DEFINE RESPONSES

(response: if CO2_PUMP is FAILED then CO2_REMOVAL_COMMAND is OFF   0.9)


CO2 ACCUMULATOR AIR FLOW SENSOR     (RULE #5)

PARAMETER MODIFIER DEFINITIONS

(level: if CO2_ACCUM is X then ACCUM_FLOW is HI     50  100  500  900  lb/hr)
(level: if CO2_ACCUM is X then ACCUM_FLOW is NORMAL  1    5   20   50  lb/hr)
(level: if CO2_ACCUM is X then ACCUM_FLOW is Low  -900 -100    0    1  lb/hr)

DEFINE CONDITIONS

(condition: if ACCUM_FLOW is HI     then CO2_ACCUM is FAILED  0.2)
(condition: if ACCUM_FLOW is NORMAL then CO2_ACCUM is FAILED  0.0)
(condition: if ACCUM_FLOW is LOW    then CO2_ACCUM is FAILED  0.8)

DEFINE RESPONSES

(response: if CO2_ACCUM is FAILED then CO2_REMOVAL_COMMAND   is OFF  0.9)
(response: if CO2_ACCUM is FAILED then CO2_REDUCTION_COMMAND is OFF  1.0)
```

APPENDIX 3B
CO_2 REDUCTION ASSEMBLY KNOWLEDGE BASE

```
/**  This set of statements defines the sensors, their names, and the units they read in **/


(deffacts sensor_list
    "(define_sensor:  ?subassembly   ?sensor           ?units)"
     (define_sensor   CO2_REDUCTION  JACKET_FLOW_1     CFM)
     (define_sensor   CO2_REDUCTION  JACKET_FLOW_2     CFM)
     (define_sensor   CO2_REDUCTION  SYSTEM_EXIT_TEMP  K)
     (define_sensor   CO2_REDUCTION  REACTION_TEMP     K)
     )

/**  This section defines the ranges the sensors will be reading if certain problems are occurring.  These ar the "rules"
     referred to in the report portion of this paper. **/


(deffacts parameter_level_definitions
   "        sensor                  parameter         modifier  minimum  lower  upper  maximum  units"
   (level:if JACKET_FLOW_1 is X then COOLANT_FLOW_1 is LO          0.2     0.2    0.4    0.4     CFM)   (Rule 1)
   (level:if JACKET_FLOW_1 is X then COOLANT_FLOW_1 is NORMAL      0.4     0.4    0.7    0.7     CFM)   (Rule 2)
   (level:if JACKET_FLOW_2 is X then COOLANT_FLOW_2 is LO          0.2     0.2    0.4    0.4     CFM)   (Rule 3)
   (level:if JACKET_FLOW_2 is X then COOLANT_FLOW_2 is NORMAL      0.4     0.4    0.7    0.7     CFM)   (Rule 4)
   (level:if SYSTEM_EXIT_TEMP is X then CONDENSER_TEMP is HIGH     360     373    1000   1000     K)   (Rule 5)
   (level:if SYSTEM_EXIT_TEMP is X then CONDENSER_TEMP is NORMAL   300     310    350     360     K)   (Rule 6)
   (level:if REACTION_TEMP is X then EFFICIENCY_LEVEL is LO        873     900    2000   2000     K)   (Rule 7)
   (level:if REACTION_TEMP is X then EFFICIENCY_LEVEL is NORMAL    400     450    850     873     K)   (Rule 8)
   (level:if REACTION_TEMP is X then EFFICIENCY_LEVEL is NOT_FUNCTIONAL    300  310   380   400   K)   (Rule 9)
     )


/**  This section defines the probable trouble occurring for the sensor indications listed above. **/


(deffacts condition_definitions
   "condition:                                                    strength"
   (condition: if COOLANT_FLOW_1 is LO     then  EXIT_TEMP is HIGH  1)
   (condition: if COOLANT_FLOW_1 is NORMAL then  EXIT_TEMP is OK    1)
   (condition: if COOLANT_FLOW_2 is LO     then  EXIT_TEMP is HIGH  1)
   (condition: if COOLANT_FLOW_2 is NORMAL then  EXIT_TEMP is OK    1)
   (condition: if CONDENSER_TEMP is HIGH   then  CONDENSER is NOT_WORKING 1)
   (condition: if CONDENSER_TEMP is NORMAL then  CONDENSER is WORKING 1)
   (condition: if EFFICIENCY_LEVEL is LO   then  REACTOR   is TOO_HOT 1)
   (condition: if EFFICIENCY_LEVEL is NORMAL then REACTOR  is AT_RIGHT_TEMP     1)
   (condition: if EFFICIENCY_LEVEL is NOT_FUNCTIONAL then REACTOR is TOO_COLD 1)
     )


/**  This section defines the expert system's suggested response to problems occurring in the assembly. **/


(deffacts response_define
          "?sub_assembly ?mode     ?condition ?strength"
   (response: if EXIT_TEMP is HIGH  then CO2_REDUCTION_COMMAND is  OFF  1)
   (response: if EXIT_TEMP is OK    then CO2_REDUCTION_COMMAND is NORMAL 1)
   (response: if CONDENSER is NOT_WORKING then CO2_REDUCTION_COMMAND is OFF   1)
   (response: if CONDENSER is WORKING then CO2_REDUCTION_COMMAND is NORMAL 1)
   (response: if EFFICIENCY_LEVEL is LO then CO2_REDUCTION_COMMAND is OFF 1)
   (response: if EFFICIENCY_LEVEL is NORMAL then CO2_REDUCTION_COMMAND is NORMAL 1)
   (response: if EFFICIENCY_LEVEL is NOT_FUNCTIONAL then CO2_REDUCTION_COMMAND is  OFF  1)
     )


/**  This section sends a set of command signals to the system based on the decisions made above. **/


(deffacts implementation_commands
               "?sub_assembly ?mode          ?command"
     (implement: if CO2_REDUCTION_COMMAND is OFF  then  COMMAND CO2_REDUCTION to OFF)
     (implement: if CO2_REDUCTION_COMMAND is NORMAL then COMMAND CO2_REDUCTION to NORMAL)
```

APPENDIX 3C
OXYGEN GENERATION ASSEMBLY KNOWLEDGE BASE

```
/**  This set of statements defines the sensors, their names, and the units they read in **/

(deffacts sensor_list
    "(sensor: ?subassembly ?senor                    ?units)"
     (sensor:  OGA         H2O_FEEDLINE_PRESSURE     PSI   )
     (sensor:  OGA         H2O_PHOTOCELL             logical)
     (sensor:  OGA         H/O_DEFERENTIAL_PRESSURE  PSI   )
     (sensor:  OGA         O2_OUTTAKE_TEMP           F     )
     (sensor:  OGA         CELL_VOLTAGE              volts )
     )


/**  This section defines the ranges the sensors will be reading if certain problems are occuring.  These
     are the "rules" referred to in the report portion of this paper. **/


(deffacts parameter_level_definitions
     "          sensor                                parameter          modifier minimum lower upper maximum units"
     (level: if H2O_FEEDLINE_PRESSURE is X then MODULE_H2O_PRESSURE is HI         30      31    40    40 PSI)  (rule 1)
     (level: if H2O_FEEDLINE_PRESSURE is X then MODULE_H2O_PRESSURE is NOMINAL    19      20    30    31 PSI)  (rule 2)
     (level: if H2O_FEEDLINE_PRESSURE is X then MODULE_H2O_PRESSURE is LOW        0       0     19    20   PSI)  (rule 3)
     (level: if H2O_PHOTOCELL is X then H2O_SUPPLY_LEVEL is HI                    1       1     1     1   BIN)  (rule 4)
     (level: if H2O_PHOTOCELL is X then H2O_SUPPLY_LEVEL is LOW                   0       0     0     0   BIN)  (rule 5)
     (level: if H/O_DEFERENTIAL_PRES is X then CROSS_CATALYIST_PRESSURE is HI     3.1     3.35  4     4   PSI)  (rule 6)
     (level: if H/O_DEFERENTIAL_PRES is X then CROSS_CATALYIST_PRESSURE is NOMINAL 2.83    2.9   3.1   3.35PSI)  (rule 7)
     (level: if H/O_DEFERENTIAL_PRES is X then CROSS_CATALYIST_PRESSURE is LOW    0       0     2.83  2.9PSI)  (rule 8)
     (level: if O2_OUTTAKE_TEMP is X then OUTTAKE_O2 is HOT                       105.8   110   125   125   F)   (rule 9)
     (level: if O2_OUTTAKE_TEMP is X then OUTTAKE_O2 is NOMINAL                   50      55.4  105.8 110   F)   (rule 10)
     (level: if O2_OUTTAKE_TEMP is X then OUTTAKE_O2 is COLD                      32      32    50    50  F)   (rule 11)
     (level: if CELL_VOLTAGE is X then CELL_VOLTAGE is HI                         1.68    1.7   2     2  V)   (rule 12)
     (level: if CELL_VOLTAGE is X then CELL_VOLTAGE is NOMINAL                    1.6     1.64  1.68  1.7V)   (rule 13)
     )

/** This section defines the probable trouble occuring for the sensor indications listed above. **/


(deffacts condition_define:
     "condition:                                                                                    strength "
     (condition:  if MODULE_H2O_PRESSURE is HI       then H2O_DELIVERY_PRESSURE is HI       1)
     (condition:  if MODULE_H2O_PRESSURE is NOMINAL  then H2O_DELIVERY_PRESSURE is NOMINAL  1)
     (condition:  if MODULE_H2O_PRESSURE is LOW      then H2O_DELIVERY_PRESSURE is LOW      1)
     (condition:  if H2O_SUPPLY_LEVEL is HI          then WATER_TANK is OK                  1)
     (condition:  if H2O_SUPPLY_LEVEL is LOW         then WATER_TANK is EMPTY               1)
     (condition:  if CROSS_CATALYIST_PRESSURE is HI  then CELL_ELECTRODE is BAD              1)
     (condition:  if CROSS_CATALYIST_PRESSURE is NOMINAL  then CELL_ELECTRODE is GOOD        1)
     (condition:  if CROSS_CATALYIST_PRESSURE is LOW then CELL_ELECTRODE is BAD              1)
     (condition:  if OUTTAKE_O2 is HOT               then THROTTLE is TOO_TIGHT             1)
     (condition:  if OUTTAKE_O2 is NOMINAL           then THROTTLE is FINE                  1)
     (condition:  if OUTTAKE_O2 is COLD              then THROTTLE is TOO_OPEN              1)
     (condition:  if CELL_VOLTAGE is HI              then INTAKE_H2O is LOST                1)
     (condition:  if CELL_VOLTAGE is NOMINAL         then INTAKE_H2O IS FINE                1)
     )


 /** This section defines the expert systems suggested response to problems occuring in the assembley **/


(deffacts response_define
              "?sub_assembly ?mode     ?condition ?strength"
     (response: if H2O_DELIVERY_PRESSURE is HI      then OGA_COMMAND is SHUTDOWN   1)
     (response: if H2O_DELIVERY_PRESSURE is NOMINAL then OGA_COMMAND is NORMAL     1)
     (response: if H2O_DELIVERY_PRESSURE is LOW     then OGA_COMMAND is SHUTDOWN   1)
     (response: if WATER_TANK is OK        then OGA_COMMAND is NORMAL    1)
     (response: if WATER_TANK is EMPTY     then OGA_COMMAND is SWITCH    1)
     (response: if CELL_ELECTRODE is BAD   then OGA_COMMAND is SHUTDOWN  1)
     (response: if CELL_ELECTRODE is GOOD  then OGA_COMMAND is NORMAL    1)
     (response: if THROTTLE is TOO_TIGHT   then OGA_COMMAND is SHUTDOWN  1)
     (response: if THROTTLE is FINE        then OGA_COMMAND is NORMAL    1)
     (response: if THROTTLE is TOO_OPEN    then OGA_COMMAND is SHUTDOWN  1)
     (response: if INTAKE_H2O is LOST      then OGA_COMMAND is SHUTDOWN  1)
     (response: if INTAKE_H2O IS FINE      then OGA_COMMAND is NORMAL    1)
     )
```

```
/**  This section sends a set of command signals to the system based on the decisions made above. **/


(deffacts implementation_commands
                "       response?sub_assembly ?mode          ?command"
     (implement: if OGA_COMMAND is NORMAL    then  COMMAND: OGA to NORMAL    )
     (implement: if OGA_COMMAND is SHUTDOWN  then  COMMAND: OGA to SHUTDOWN  )
     (implement: if OGA_COMMAND is UNPOWERED then  COMMAND: OGA to UNPOWERED )
     (implement: if OGA_COMMAND is STANDBY   then  COMMAND: OGA to STANDBY   )
     (implement: if OGA_COMMAND is CURRENT   then  COMMAND: OGA to CURRENT   )
     (implement: if OGA_COMMAND IS SWITCH    then  COMMAND: OGA to SWITCH    )
     )
```

APPENDIX 3D
SAVANT.3 EXPERT SYSTEM CODE

```
;-----------------------------------------------------------------
; CO2_REMOVAL - KNOWLEDGE BASE

(deffacts sensor_list
;      "(sensor: ?sub-assembly ?sensor          ?units )"
        (sensor: CO2_REMOVAL   INLET_CO2          ppm )
        (sensor: CO2_REMOVAL   CO2_ACCUMULATOR    g   )
     (sensor: OGA           H2O_ACCUMULATOR    g   )
     (sensor: OGA           CABIN_O2           ppm )
     )

(deffacts parameter_modifier_definitions  "    define triangle membership functions"
;    (level: if ?sensor   is ?X then ?parameter is ?modifier     ?minimum ?lower ?upper ?maximum ?units)
     (level: if INLET_CO2 is  X then CO2_LEVEL  is HI              900     1100  1000000 1000000  ppm)
     (level: if INLET_CO2 is  X then CO2_LEVEL  is NORMAL          275      325     900   1100    ppm)
     (level: if INLET_CO2 is  X then CO2_LEVEL  is LOW               0        0     275    325    ppm)

     (level: if CO2_ACCUMULATOR is  X then CO2_SUPPLY is HI     700      900    1000   1000  g  )
     (level: if CO2_ACCUMULATOR is  X then CO2_SUPPLY is OK     100      300     700    900  g  )
     (level: if CO2_ACCUMULATOR is  X then CO2_SUPPLY is LOW     50       75     100    300  g  )
     (level: if CO2_ACCUMULATOR is  X then CO2_SUPPLY is CRITICAL 0        0      50     75  g  )
     (level: if CO2_ACCUMULATOR is  X then CO2_SUPPLY is EMPTY    0        0       5     10  g  )

     (level: if H2O_ACCUMULATOR is  X then H2O_SUPPLY is HI    1400     1800    2000   2000  g  )
     (level: if H2O_ACCUMULATOR is  X then H2O_SUPPLY is OK     200      600    1400   1800  g  )
     (level: if H2O_ACCUMULATOR is  X then H2O_SUPPLY is LOW    100      300     200    600  g  )
     (level: if H2O_ACCUMULATOR is  X then H2O_SUPPLY is CRITICAL 0        0     100    300  g  )
     (level: if H2O_ACCUMULATOR is  X then H2O_SUPPLY is EMPTY    0        0       5     10  g  )

     (level: if CABIN_O2 is  X then O2_LEVEL is HI    230000   250000   1000000   1000000  ppm )
     (level: if CABIN_O2 is  X then O2_LEVEL is OK     15000    18000     23000     25000  ppm  )
     (level: if CABIN_O2 is  X then O2_LEVEL is LOW    12000    15000     15000     18000  ppm  )
     (level: if CABIN_O2 is  X then O2_LEVEL is CRITICAL 0        0      12000     15000  ppm  )

     )

(deffacts define_conditions
;    (condition: if ?parameter is ?modifier                          then ?condition          is ?modifier ?strength)
     (condition: if CO2_LEVEL is HI                                  then CO2_REMOVAL_DEMAND is HI            1    )
     (condition: if O2_LEVEL  is CRITICAL                            then CO2_REMOVAL_DEMAND is HI            1    )
     (condition: if CO2_SUPPLY is LOW   or H2O_SUPPLY is CRITICAL    then CO2_REMOVAL_DEMAND is HI            1    )
     (condition: if CO2_LEVEL is NORMAL                              then CO2_REMOVAL_DEMAND is NORMAL        .8   )
     (condition: if CO2_LEVEL is LOW                                 then CO2_REMOVAL_DEMAND is LOW           .7   )
     (condition: if CO2_LEVEL is LOW    and CO2_SUPPLY is LOW        then CO2_REMOVAL_DEMAND is NORMAL        .9   )

     (condition: if O2_LEVEL  is HI                                  then O2_GENERATION_DEMAND is LOW         1    )
     (condition: if O2_LEVEL  is_not HI                              then O2_GENERATION_DEMAND is HI          1    )

     (condition: if H2O_SUPPLY is HI                                 then CO2_REDUCTION_DEMAND is LOW         1    )
     (condition: if H2O_SUPPLY is_not HI                             then CO2_REDUCTION_DEMAND is HI          1    )

     (condition: if H2O_SUPPLY is EMPTY                              then OGA_SUB-ASSEMBLY is OUT_OF_WATER    1    )
     (condition: if H2O_SUPPLY is CRITICAL                           then OGA_SUB-ASSEMBLY is OUT_OF_WATER   .2    )
     (condition: if CO2_SUPPLY is EMPTY                              then CO2_REDUCTION_SUB-ASSEMBLY is OUT_OF_CO2     1    )
     (condition: if CO2_SUPPLY is CRITICAL                           then CO2_REDUCTION_SUB-ASSEMBLY is OUT_OF_CO2    .2    )

     )

(deffacts define_responses
;    (response: if ?condition          is ?modifier then ?sub-assembly          is ?commanded      ?strength)
     (response: if CO2_REMOVAL_DEMAND   is HI       then CO2_REMOVAL   is GO_TO_HIGH_REMOVAL_MODE       1    )
     (response: if CO2_REMOVAL_DEMAND   is NORMAL   then CO2_REMOVAL   is GO_TO_POWER_EFFICIENCY_MODE   1    )
     (response: if CO2_REMOVAL_DEMAND   is LOW      then CO2_REMOVAL   is TURN_OFF                      1    )

     (response: if O2_GENERATION_DEMAND is LOW      then OGA           is TURN_OFF                      1    )
     (response: if O2_GENERATION_DEMAND is HI       then OGA           is TURN_ON                       1    )
     (response: if CO2_REDUCTION_DEMAND is LOW      then CO2_REDUCTION is TURN_OFF                      1    )
     (response: if CO2_REDUCTION_DEMAND is HI       then CO2_REDUCTION is TURN_ON                       1    )

     (response: if OGA_SUB-ASSEMBLY is OUT_OF_WATER then OGA           is TURN_OFF                      1    )
     (response: if CO2_REDUCTION_SUB-ASSEMBLY is OUT_OF_CO2 then CO2_REDUCTION is TURN_OFF              1    )
```

```
    )

(deffacts command_implementation
;    (implement: if ?sub-assembly          is ?commanded        then WRITE: ?sub-assembly to ?string      )
     (implement: if CO2_REMOVAL   is GO_TO_HIGH_REMOVAL_MODE   then WRITE: CO2_REMOVAL   to HIGH_POWER   )
     (implement: if CO2_REMOVAL   is GO_TO_POWER_EFFICIENCY_MODE  then WRITE: CO2_REMOVAL   to ECONOMY      )
     (implement: if CO2_REMOVAL   is TURN_OFF                  then WRITE: CO2_REMOVAL   to OFF          )
     (implement: if OGA           is TURN_OFF                  then WRITE: OGA           to OFF          )
     (implement: if OGA           is TURN_ON                   then WRITE: OGA           to ON           )
     (implement: if CO2_REDUCTION is TURN_OFF                  then WRITE: CO2_REDUCTION to OFF          )
     (implement: if CO2_REDUCTION is TURN_ON                   then WRITE: CO2_REDUCTION to ON           )

     )

; SAVANT.3
;---------------------------------------------------------------------
; "INFERENCE" ENGINE  (shell above CLIPS)

(defrule evaluate_parameters "(direct reading of parameters)"
        (sensor: ?sub-assembly1 ?sensor  ?units )
     (sub-assembly: ?sub-assembly2 ?file_id )
     (time ?time ?time ?iterations)
     (sensor ?sub-assembly1 ?sensor ?value ?units ?time_stamp)
     (test (= ?time ?time_stamp))
=>
     (assert (?sensor is ?value ?units =(gensym) ))
     )

(defrule determine_parameter_modifiers  "evaluate membership function to convert value to weighted level"
        (level: if ?sensor is ?X then ?parameter is ?modifier ?minimum ?lower ?upper ?maximum ?units)
     (?sensor is ?value ?units ?tag )
;    (test (and (>= ?value ?minimum) (< ?value ?maximum)))
=>
     (bind ?weight 0)
     (if (and (>  ?value ?minimum) (<  ?value ?lower  ))
        then   (bind ?weight (/ (- ?value ?minimum ) (- ?lower ?minimum ))))
     (if (and (>= ?value ?lower  ) (<= ?value ?upper))
        then   (bind ?weight 1))
     (if (and (>  ?value ?upper  ) (<  ?value ?maximum))
        then   (bind ?weight (/ (- ?value ?maximum ) (- ?upper ?maximum ))))
     (bind ?confidence ?weight)
     (assert (?parameter is ?modifier ?confidence =(gensym) ))
     )

(defrule determine_conditions
     (condition: if $?parameter_statement then $?asserted_condition_statement ?strength)
     ($?parameter_statement ?confidence ?tag )
=>
     (bind ?confidence (* ?confidence ?strength))
     (assert ($?asserted_condition_statement ?confidence =(gensym) ))
     )

(defrule determine_response
     (response: if $?condition_statement then $?asserted_response_statement ?strength)
     ($?condition_statement ?confidence ?tag )
=>
     (bind ?confidence (* ?confidence ?strength))
     (assert ($?asserted_response_statement ?confidence =(gensym) ))
     )

;mutual renforcement?

; SAVANT.3
;---------------------------------------------------------------------------
; RESOLVE LOGICAL OPERATIONS

(defrule logical_and            "VVV"
     (declare (salience -5))
 ?rm <-   ($?left $?left_statement and $?right_statement $?right)
     ($?left_statement ?lconfidence ?ltag )
     (test (numberp ?lconfidence ))
     ($?right_statement ?rconfidence ?rtag )
     (test (numberp ?rconfidence ))
=>
;                          V
     (bind ?confidence (* ?lconfidence ?rconfidence ))
     (retract ?rm)
        (assert ($?left $?left_statement AND $?right_statement $?right))
     (assert ($?left_statement AND $?right_statement ?confidence =(gensym) ))
```

```
    )

(defrule logical_or              "VV"
    (declare (salience -5))
?rm <-    ($?left $?left_statement or $?right_statement $?right)
    ($?left_statement ?lconfidence ?ltag )
    (test (numberp ?lconfidence ))
    ($?right_statement ?rconfidence ?rtag )
    (test (numberp ?rconfidence ))
=>
;                          V
    (bind ?confidence (min (+ ?lconfidence ?rconfidence ) 1))
    (retract ?rm)
       (assert ($?left $?left_statement OR $?right_statement $?right))
    (assert ($?left_statement OR $?right_statement ?confidence =(gensym) ))
    )

(defrule logical_not
    (declare (salience -5))
?rm <-    ($?left ?noun is_not ?modifier $?right)
    (?noun is ?modifier ?confidence ?tag )
    (test (numberp ?confidence ))
=>
;                                V
    (bind ?not_confidence (- 1 ?confidence ))
    (retract ?rm)
    (assert ($?left ?noun IS_NOT ?modifier $?right))
    (assert (?noun IS_NOT ?modifier ?not_confidence =(gensym) ))
    )

(defrule mutual_renforcement "'OR' confidence values of idential facts
                    [those with identical nouns and modifiers].
                    Form a single fact from the two."
?rm1 <-   (?noun1 is ?modifier1 ?confidence1 ?tag1 )
?rm2 <-   (?noun2 is ?modifier2 ?confidence2 ?tag2 )
       (test (and (eq ?noun1 ?noun2)
            (eq ?modifier1 ?modifier2 )
                 (neq ?tag1 ?tag2 )))
=>
    (bind ?confidence (min (+ ?confidence1 ?confidence2) 1))
    (retract ?rm1 ?rm2)
    (assert (?noun1 is ?modifier1 ?confidence =(gensym) ))
    )

; SAVANT.3
;-------------------------------------------------------------------------
; RESPONSE SELECTION AND IMPLEMENTATION

(defrule select_response1
    (declare (salience -10))
    (response: if $?condition_statement then ?sub-assembly is ?command ?strength)
?rm1 <-   (?sub-assembly is ?command1 ?confidence1 ?tag1 )
?rm2 <-   (?sub-assembly is ?command2 ?confidence2 ?tag2 )
       (test (neq ?command1 ?command2 ))
       (test (neq ?tag1 ?tag2))
=>
    (if (< ?confidence1 ?confidence2)
       then   (retract ?rm1)
       else (retract ?rm2))
    )

;threshold?

(defrule implement_response
    (declare (salience -100))
    (implement: if ?sub-assembly1 is ?mode then WRITE: ?sub-assembly to ?string )
       (?sub-assembly1 is ?mode ?confidence ?tag )
    (sub-assembly: ?sub-assembly2 ?file_id)
    (time ?time ?time ?iterations)
=>
       (bind ?file_id.ext (str_cat ?file_id ".CMD"))
    (bind $?command_fact (mv-append command ?sub-assembly1 to ?string ?time))
    (printout ?file_id.ext "(" $?command_fact ")" crlf )
    (printout t            "(" $?command_fact ")" crlf )
    )

; SAVANT.3
;-------------------------------------------------------------------
; EXPLANATION FACILITY
```

```
(defrule echo_parameters
       (sensor: ?sub-assembly1 ?sensor  ?units )
     (?sensor is ?value ?units ?tag1 )
       (level: if ?sensor is ?X then ?parameter is ?modifier $? ?units)
     (?parameter ?link ?modifier ?confidence ?tag2 )
     (test (> ?confidence 0 ))
=>
     (printout t ?sensor " is " ?value " " ?units "    => " ?parameter " " ?link " " ?modifier "  " ?confidence crlf)
     )

(defrule evaluate_parameters "(direct reading of parameters)"
       (sensor: ?sub-assembly1 ?sensor  ?units )
     (sub-assembly: ?sub-assembly2 ?file_id )
     (time ?time ?time ?iterations)
     (sensor ?sub-assembly1 ?sensor ?value ?units ?time_stamp)
     (test (= ?time ?time_stamp))
=>
     (assert (?sensor is ?value ?units =(gensym) ))
     )

(defrule determine_parameter_modifiers  "evaluate membership function to convert value to weighted level"
       (level: if ?sensor is ?X then ?parameter is ?modifier ?minimum ?lower ?upper ?maximum ?units)
     (?sensor is ?value ?units ?tag )
;    (test (and (>= ?value ?minimum) (< ?value ?maximum)))
=>
     (bind ?weight 0)
     (if (and (>= ?value ?minimum) (< ?value ?lower  ))
        then   (bind ?weight (/ (- ?lower   ?value) (- ?lower ?minimum))))
     (if (and (>= ?value ?lower  ) (< ?value ?upper))
        then   (bind ?weight 1))
     (if (and (>= ?value ?upper  ) (< ?value ?maximum))
        then   (bind ?weight (/ (- ?maximum ?value) (- ?maximum ?upper))))
     (bind ?confidence ?weight)
     (assert (?parameter is ?modifier ?confidence =(gensym) ))
     )


(defrule start_explanation_of_response
     (declare (salience -1000))
 ?rm <- (?response is ?mode ?confidence ?tag )
     (response: if $?condition_statement then ?response is ?mode ?strength)
=>
     (retract ?rm)
     (assert (?response is ?mode ?confidence ))
     (assert (explain_response: ?response ?mode))
     (printout t crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf "EXPLANATION:" crlf)
     (printout t "SAVANT commands the " ?response " sub-assembly to " ?mode "." crlf crlf)
     (printout t "This is because SAVANT has determined that the following conditions are " crlf)
     (printout t "likely to exist onboard, and SAVANT has been told that the proper response " crlf)
     (printout t "to these conditions is to command the " ?response " sub-assembly " crlf)
     (printout t "to " ?mode "." crlf)
        )

(defrule explain_conditions
     (declare (salience -10))
     (explain_response: ?response ?mode)
     (response: if ?condition is ?modifier then ?response is ?mode ?strength)
     ( ?condition is ?modifier ?confidence ?tag )
     (test (> ?confidence 0))
=>
     (assert (explain_condition: ?condition ?modifier ))
     (printout t crlf "CONDITION: " ?condition " is likely to be " ?modifier crlf)
     (printout t      "           This condition is likely because: " crlf)
        )

(defrule list_evidence
     (explain_response: ?response ?mode)
     (response: if ?condition is ?cmodifier then ?response is ?mode ?rstrength)
     (explain_condition:  ?condition ?cmodifier )
     (condition: if $?parameter_statement then ?condition is ?cmodifier ?cstrength)
     ($?parameter_statement ?pconfidence ?tag )
     (test (> ?pconfidence 0))
=>
     (printout t      "                > " $?parameter_statement "    ( " ?pconfidence " ) " crlf)
        )

(defrule complete_explanation_of_condition
     (declare (salience -10))
     (explain_response: ?response ?mode)
```

```
       (?response is ?mode ?confidence )
=>
     (printout t crlf "The cumulative confidence of this conclusion is " ?confidence "." crlf)
     (printout t "Continue (Y/N)? ")
     (bind ?continue (read))
;    (if (or (eq ?continue n) (eq ?continue no) (eq ?continue N) (eq ?continue NO) )
;     then (halt))
     )

; SAVANT.3
;---------------------------------------------------------------------
; FILE I/O OVERHEAD

(defrule open_command_file
     (declare (salience  100))
     (sub-assembly: ?sub-assembly_name ?file_id)
=>
        (bind ?file_id.ext (str_cat ?file_id ".CMD"))
     (open ?file_id.ext ?file_id.ext "w")
     )

(defrule close_command_file
     (declare (salience -100))
     (sub-assembly: ?sub-assembly_name ?file_id)
=>
        (bind ?file_id.ext (str_cat ?file_id ".CMD"))
     (close ?file_id.ext)
     )

; SAVANT.3
;-----------------------------------------------------------------------------
;

(deffacts sub-assembly_I/O_file_definitions
;    (sub-assembly: ?sub-assembly  ?filename  )
     (sub-assembly: CO2_REMOVAL    "CO2REMOV" )
     )

(deffacts configure_program
        "            sub-assembly   file_id  "
     (^O)
;    (load ".CMD")
     (load ".SEN")
     (extract_time 0)
     )

(defrule load_facts_file
     (declare (salience 100))
     (sub-assembly: ?sub-assembly_name ?file_id)
     (load ?ext)
=>
        (bind ?file_id.ext (str_cat ?file_id ?ext))
     (load-facts ?file_id.ext)
     )

; SAVANT.3
;-------------------------------------------------------------------------
; TIME STAMP EXTRACTION AND INSERTION

(defrule extract_time  "what time is it?"
     (declare (salience 100))
 ?rm <-   (extract_time ?time)
     (sensor CO2_REMOVAL CLOCK ?iterations min ?time_stamp )
     (test (< ?time ?time_stamp ))
=>
     (if (!= ?time_stamp ?time)
        then   (bind ?time ?time_stamp )
          (retract ?rm)
          (assert (extract_time ?time))
     ))

(defrule time_extracted  "time is:"
     (declare (salience -100))
 ?rm <-   (extract_time ?time)
     (sensor CO2_REMOVAL CLOCK ?iterations min ?time)
=>
     (retract ?rm)
     (assert (time ?time ?time ?iterations))
     )
```

```
; SAVANT.3
;---------------------------------------------------------------------
; EXIT

(defrule exit
     (declare (salience -1000))
     (time ?time ?time ?iterations)
=>
     (save-facts "facts")
     (if (>= ?time 60 )
     then (halt)
;         block out following line with a semicolon to keep system from looping
;    else (batch "model.bat")
     ))
```