https://ntrs.nasa.gov/search.jsp?R=19910018452 2020-03-19T16:46:33+00:00Z

UAH Research Report Number 824

A Diagnostic Prototype of the Potable Water Subsystem of the Space Station Freedom ECLSS

NAS8-36955 D.O. 25

Prepared for
Brandon S. Dewberry

Information and Electronic Systems Lab
Software and Data Management Division
Systems Software Branch
Marshall Space Flight Center

Prepared by
Brenda D. Lukefahr
Daniel M. Rochowiak
Brian L. Benson
John S. Rogers
James W. McKee

The Johnson Research Center
The University of Alabama in Huntsville
Huntsville, Alabama 35899

November 1989

UAH grants to the Government, and others acting on its behalf, a paid-up, nonexclusive, inevocable, worldwide lisensce to reproduce, prepare deriavative works, distribute copies to the public, and perform publicly and display publicly by or on behalf of the Government.

Copyright © 1989 UAH

THIS WORK SPONSORED BY NASA/ MSFC UNDER CONTRACT NAS 8-36955, DELIVERY ORDER 25

TABLE OF CONTENTS

INTRODUCTION

This report is an addition to UAH Report No. 823 "ECLSS Advanced Automatic Preliminary Requirement-Final Report", and is the documentation for the demonstration software constructed for that project All of the acronyms used in this paper are defined in UAH Report No 823.

The purpose of the software is demonstrate how a rule based approach to programming can be used for the purposes of simulation and diagnostics within the general ECLSS environment. The specific domain for the software is the potable water loop, but the general conception could be applied to other ECLSS subsystems.

The demonstration software was constructed from the CLIPS inference engine and the HyperCard™ hypertext software for the Macintosh™ computer.

It should be noted that this demonstration software is a preliminary prototype system. There are several reasons for this. First the domain is in a state of flux. The actual configuration and components of the potable water system have not been fixed. Thus, the system is about the intended design of the potable water system as we understood it to be in the middle of 1989. Second the general project of which this software is part was an evaluation project. Thus, this effort is not intended to be a full scale prototyping effort. Finally, the software is constructed from readily available off the shelf software tools. A full prototype system may use either more powerful tools or be constructed from scratch in Ada. The first option was rejected since it was deemed desirable to use the NASA developed CLIPS inference engine, and the latter was rejected owing to time, money, and expertise constraints.

About the Demonstration Software

The principles employed in construction this demonstration software are rather simple. The HyperCard software was used to build the user interface and interpretation functions for the system. In a rather coarse analogy, this would correspond to the DISPLAY unit of the ECLSS Software Support Package (now ECLSSMGR). As noted in Report No. 823, this is an important site for a knowledge based system. The actual data generation and diagnostics are implemented in the CLIPS system. Using the same rather coarse analogy, this would correspond to either part of the ECLSSERR (at a high level) or CHKSTA (at a low level). With this construction it should be clear that each of the computational agents, the interface and the inference engine, maintain a degree of autonomy. Thus modifications to one of the agents need not demand any modifications of the other agent.

As shown in Figure 1, the two components run in a serial fashion. Information is transferred in ASCII files. Other schema are, of course, possible given the general design. For example the display and the inference engine could be located on physically distinct computers or distinct processors in a single computer environment. Communications in either of these cases could be arranged either through a network (distinct computers) or a bus transfer (distinct processors). In a more well developed system, it is likely that one of these options would be used.

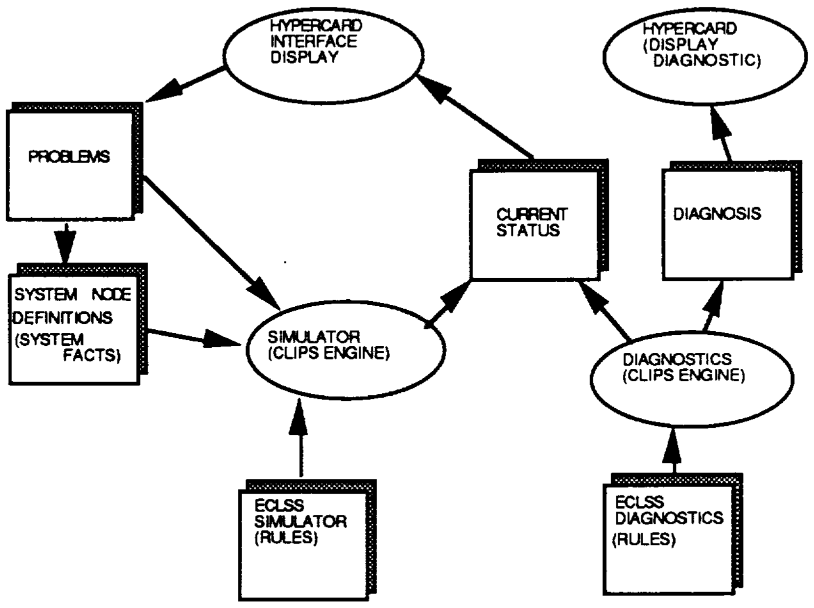

HYPERCARD INTERFACE DISPLAY
HYPERCARD (DISPLAY DIAGNOSTIC)
PROBLEMS
CURRENT STATUS
DIAGNOSIS
SYSTEM NODE DEFINITIONS (SYSTEM FACTS)
SIMULATOR (CLIPS ENGINE)
DIAGNOSTICS (CLIPS ENGINE)
ECLSS SIMULATOR (RULES)
ECLSS DIAGNOSTICS (RULES)

FIGURE 1 DATA FLOW DIAGRAM

Since we did not have direct access to a test platform, a simulation is used to generate data for the demonstration. This requires that the user enter faults and the simulation produce the state of the system that would result from such faults. However, it should be noted that in a more highly developed system the simulation might be used as a prediction generator. If the diagnostics were running in real time the faults isolated by the system could be used as input to the simulation. This would allow the system to predict what would happen if the fault were really there. Additionally, any changes brought about by the diagnostics could also be entered into the simulation. If both of these avenues of opportunity are taken, a new rule system could be used to compare and the projected results to the actual results. This would allow for a greater amount of validation of actions and greater control of the physical process that is the domain for the software.

Further, it should be noted that the information used by each of the agents is symbolic in form. In this sense the inputs for the system make no special assumptions about how the information is generated. Thus, the information could come directly from sensors, or from a database, or from a suitably constructed neural net. Operating at a symbol level in this way allows for maximum flexibility and growth.

In brief, the demonstration software attempts to demonstrate how the recommendations in the UAH Report No. 823 could be implemented in a specific domain.

INSTALLATION

This section explains how to install the simulator program and diagnostic program on a Macintosh. Create a folder and place all of the software on the distribution disk into this folder. The folder should contain CLIPS, and CLIPS files named:

Simulator	[Rules for simulator]
Diagnostics	[Rules for diagnostics]
Current Status	[State description from simulator]
Problems	[Problem definitions from syseclss]
Diagnosis	[Results of diagnostics]

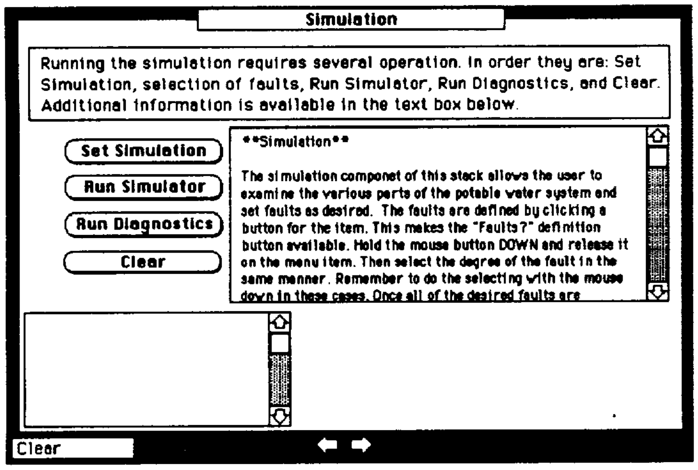
The other file in the folder should be the HyperCard stack named "syseclss."

The HyperCard software can be in any folder.

OPERATION

To operate the software, open the folder in which the software is stored, and double click on the "syseclss" stack icon.

Clicking anywhere on the opening screen will take the user to the selector card.



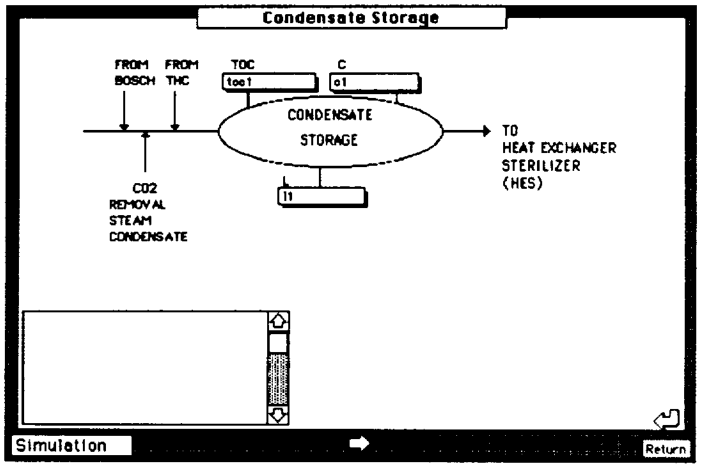
Running the simulation requires several operation. In order they are: Set Simulation, selection of faults, Run Simulator, Run Diagnostics, and Clear. Additional information is available in the text box below.

Simulation

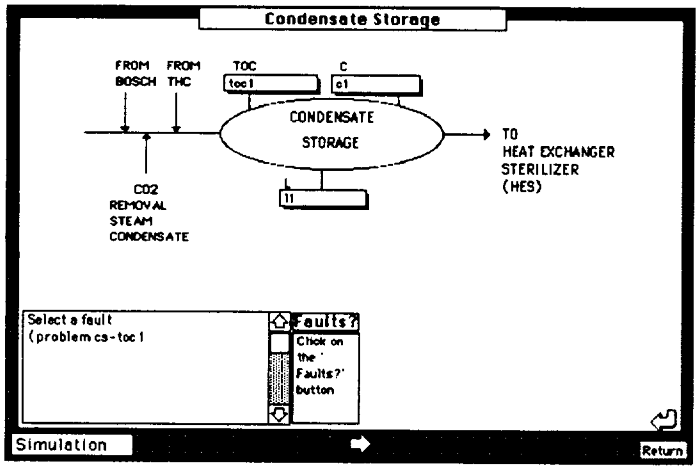
The simulation componet of this stack allows the user to examine the various parts of the potable water system and set faults as desired. The faults are defined by clicking a button for the item. This makes the "Faults?" definition button available. Hold the mouse button DOWN and release it on the menu item. Then select the degree of the fault in the same manner. Remember to do the selecting with the mouse down in these cases. Once all of the desired faults are

Clear

The selector card (at left) is the main card for the software. None of the normal HyperCard functions are disabled so the user may change the software as desired. However, it is recommended that at least one use of the software should be completed before any alterations are made. The card contains some initial comments and the top of the card, and help text in the box on the right. Standard navigation arrows appear at the bottom of the card. At the far bottom left corner is the status box. This box will display information about what the system is doing. The main operational elements are the four buttons in the middle right of the card. The first button to be used is the "Set Simulation" button. This button initializes the cards used to enter faults and takes the user to the first card of the sequence. There is some delay during this process. When the first card in the series is presented to the user the system is again ready for input. The second button is the "Run Simulator" button. This button initiates several events. The faults input by the user are collected from the various cards and are placed in a file that can be used by CLIPS. Once this file is created, CLIPS is launched. The code for the simulator rules will appear in the buffer window. Click on this window to make it the active window. From the Buffer menu select the Load Buffer item. This will compile the rules into CLIPS. Next select the Reset Item from the Execution menu. Finally, select the Run item from the Execution menu. When all of the rules have fired, select Quit from the file menu. This will return control to HyperCard™. The current status of the system will the be interpreted and appropriate values will be placed in the elements of the cards that depict the potable water system. The Run Diagnostics button launches CLIPS with the current state of the system. Repeat the steps with CLIPS as above. When control is returned to HyperCard™, the output of the rules will be examined and suitable texts and graphics will be generated. The final button is the Clear button. This button Clears all of the data items and prepares the software for another use of Set Simulation.



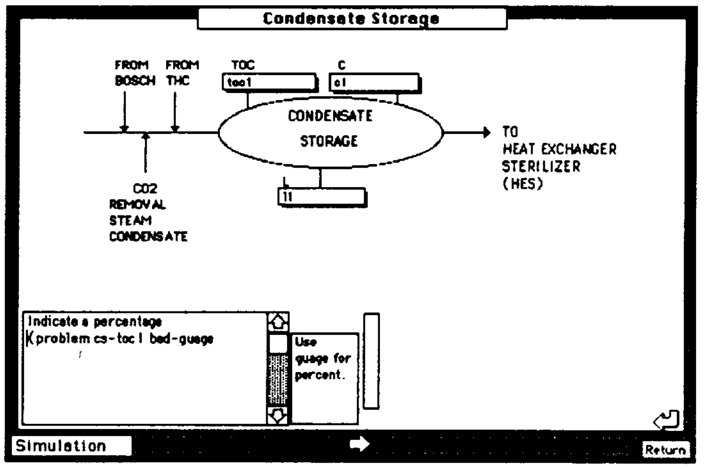
After the Set Simulation script has finished the user is placed at the first card that illustrates the components of the potable water system, Condensate storage. The card is composed of several elements that can be used to enter a fault. Clicking on an element, any of the boxes in this card will do, will begin the fault definition process. It should be noted that the user can navigate through the cards using the navigation buttons at the button of the stack. Navigation is restricted to those cards available for fault definition. To return to the selector card, the user should click the return button at the far right of the card.



Select a fault
(problem cs-toc1

Click on the 'Faults?' button

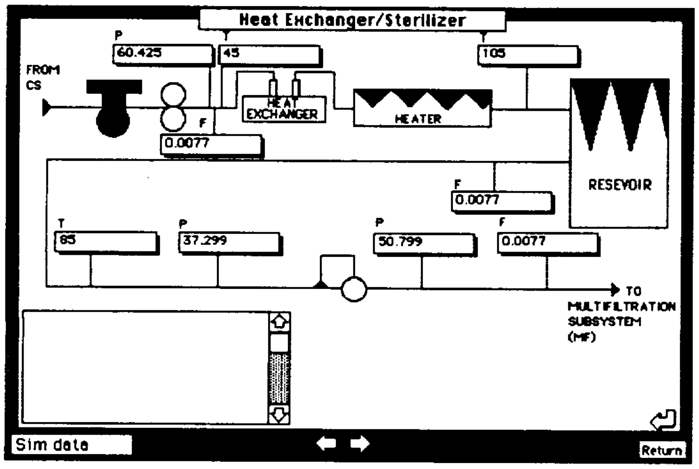
To define a fault, click on one of the boxes. In the example at the left the 'toc1' box has been clicked on and the initial part of the fault has been defined. As the fault is being built, it is displayed in the text box at the lower left. The Faults? button is now exposed. Click on this button and keep the mouse button down. This will reveal a pop-up menu of faults relevant to the selection. When the cursor has highlighted the desired fault, release the mouse button. Note that starting the definition of a fault before the previous fault definition is complete can produce errors. To clean up the errors simply use ordinary text editing procedures in the box where the fault is being defined.



Indicate a percentage
(problem cs-toc1 bad-guage

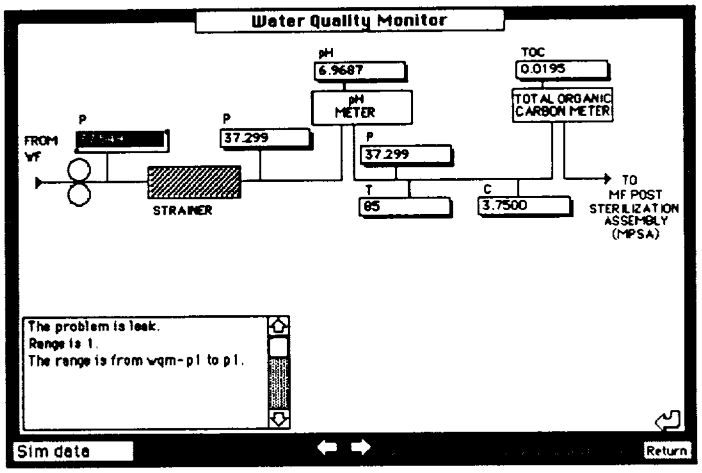
Use guage for percent.

Once the button is released the fault definition will be expanded and a vertical gauge will appear to the right of where the Faults? button had been. This gauge operates in percentages and the range of the gauge is determined by previous selections. To use the gauge, click in the area and hold the mouse button down. Moving the mouse up and down will set gauge values. When these are as the user desires, release the mouse button. At this point the fault definition is complete. The user may now add additional faults to this gauge, to the elements of this card, or any other card. When the process of entering faults is complete, use the return button to go to the selector card.



Once the fault definition process is completed use the Run Simulator button to generate results. The instructions for the use of the button are presented above. When the simulator has completed its work the values generated will be inserted into the appropriate places in the cards. It should be noted that the values for levels and flow totalizers will not be presented. The reason for this is that these are dynamic elements and the simulation systems only addresses static features of the potable water loop.

ORIGINAL PAGE IS
OF POOR QUALITY



The problem is leak.
Range is 1.
The range is from wqm-p1 to p1.

The final representations are generated through the Run Diagnostics button. The instructions for this button are presented above. This button completes its operation by generating text that explains the fault and produces a highlight area for the place at which the fault occurs or begins. When all of the diagnosed faults have been examined the Clear button can be used to reset the system.

Since the screen dumps were constructed, a new feature has been added to the demonstation interface. To the left of the Return button an Information button has been added. This navigation button takes the user to an information card. The card displays information about how to use the software, the ranges and units for the sensors, and additional notes. The Return button on this card is special. It returns the user to the card from which the Information button was activated. This sort of addition is the sort that the use of multiple agents with some degree of autonomy makes possible.

About the HyperCard Interface

The Hypercard interface was constructed in an object oriented manner in the HyperTalk language. The major functions of the scripts are to initialize the system, call CLIPS, interpret CLIPS results, and construct the actual displays. A complete listing of the HyperTalk script code can be found in Appendix A.

The initialization of the system focuses on the display of active buttons and their links to data fields. This allows new data elements to be added freely. What is initially displayed to the user is the name of the data field into which data will be placed, and the button that activates the fault definition process overlays the field. Thus the name of the unit is constant, is displayed to the user, and creates an economy of representation.

All of the scripts used in fault definition are similar and all use the underlying getData script. Once the button for fault definition is clicked, the name of the element is placed in a global variable and a message is sent to getData. This handler interprets the message and constructs the first part of the fault definition. The handler then makes the Faults? button available to the user. The script for this button constructs the menu of relevant faults and constructs the second part of the query. Finally the bar button for input is made available and the final part of the fault definition is constructed. As the fault definition is constructed it is displayed to the user.

Once the faults have been defined, the Run Simulation button's script access the defined faults, check for some, though not all, errors, constructs the file to be passed to CLIPS, and launches CLIPS. On returning from CLIPS, the a stack level script checks for the mode of the system and takes action. In this case, it puts the current status file into the state field, interprets it, and displays the results.

The Run Diagnostics button activates a similar collection of scripts and objects. The major difference is that upon return from CLIPS the system detects that it is in a different mode and uses the evaluate handler to interpret the data.

The Clear button's script simply cleans up the data in the system and prepares it for another usage.

About CLIPS Simulator

The Simulator has been designed to create parameter readings that would be representitive of the operational sensor gauge readings that would be found under various degrees of system failure. A static approach was taken that produces a time slice of these operational parameters. This limits the ability of the Diagnostic system to problems that demonstrate a definite parameter anomaly pattern and elimanates some of the more powerful diagnostic techniques such as trend analysis. Therefore, several of the dynamic parameters (such as tank level gauge readings, flow totalizers, etc...) are not modeled in this version of the Simulator. If the evaluating engineer desires a more complete simulation of fault impact or system fault tolerance, a real time dynamic simulator should be developed. With this thought in mind the simulation rules that were developed for this static parameter generator would be reasonably easy to modify and expand to take real time system changes into account.

The Simulator uses rules defined in CLIPS and an ASCII file containing set of system faults which were define in the HyperCard user interface. As mentioned earlier, system control is maintained in the HyperCard user interface. The clips environment is loaded and executed from HyperCard and control automatically returned to HyperCard once the system has completed firing all of its rules. Therefore, the user will not necessarily be required know anythig about CLIPS or the rule set itself to operate the system.

The rules, a complete listing of which can be found in Appendix B, have been designed to enable the system component defintions to be transparent . These definitions are contained in a separate CLIPS file and define the system totally. In theory this should allow the developer of the system the flexiblity to revise the system's design without being concerned about rule modifications. A simple change in the system definitions file should suffice. However, it should be noted that due to time limitations this rule abstraction has not been tested and the introduction of totally new functional members to the system will require modification.

SIMULATOR CONSIDERATIONS

There are two types of physical and chemical system definitions: those which can be represented in at least a pseudo-linear form and those which are totally non-linear. Therefore, the Simulator takes a hybrid approach to process of developing the system defintions. For the linear components, an additive superposition principal seems to work well and thus allows independent treatment of these parameters. For the non-linear terms, a multiple pass arrangement is used which allows anomaly effects to propagate back and forth through the system until the system becomes stable. Basically a original nominal system is generated first and then the anomalies are allowed to perturb that system. Since a firing sequence of the rules is required to allow a non-perturbed original system, a salience was given to each of the rules to control the order of the rule firings

Several simplifing assumptions were made which also allowed for a faster and more stream-lined simulation prototype. Most of these assumptions appear to provide results which will at least be good enough for this prototypes needs. However, for a detailed study each of these assumptions would need to be carefully reviewed to determine exactly how much of an impact it will cause in the simulator's overall deviation from reality. The assumptions are listed below:

1. First, after a careful evaluation of traditional methods of calculating the pressure drop through a pipe, it was determined that the flow rate was such that no appreciable pressure drop would be seen from point to point in the system due to skin friction.

2. Pressure was assumed to be a function of flowrate with non-uniform increases found across check valves and pumps.

3. Again because of the time required for the fluid to travel from component to component and the assumption of large volumes of ventilation air blowing across the systems pipework, the temperature of the fluid was assumed to return to ambient almost immedialely after leaving a series of heating elements.

4. It was assumed that all of the chemical properties are controlled by the performance of the unibeds and microbial

check valves. (This is obviously not true. The unibeds and check valves are the first line of defense, but are by no means the only line of defense. The sole apparent purpose of the multiple heating components located throughout the system is to keep down bacterial growth. This will directly influence each of the parameters of a chemical analysis.)

5. The unibed performances were all gauged from a report describing the performance of a single, clean unibed. The initial attempt to modeling the multi-bed system on this data resulted in water emerging from the series of unibeds far to pure. A degrading scale was designed to cause the subsequent performance of an additional unibed to be bounded to an upper limit in the degree of purity that may be obtained.

The method of anomaly introduction was devised to allow easy incorporation of its effects within the nominally operating system. Each anomaly is introduced to the simulation rule base with an associated location, problem type, and quantitative description. The quantitative description descibes the magnitude of the problem in terms of a percentage of the nominal or (in the case of multiple anomalies) current parameter values. As an example, a leak in the system is defined as a volumetric percentage of the flow entering the leak area. This causes no need for things such as a hole or crack size to be needed.

ABOUT CLIPS DIAGNOSTICS

The Diagnostic System is a rule based system developed in CLIPS. This system, like the Simulation System, is initiated from HyperCard and upon compltion of its rule firings returns to HyperCard. This again will abstract the CLIPS portion of the overall system and eliminate the requirement of experience with clips for system usage. A complete listing of the rules is found in Appendix C.

Although this system does have some commonality with the Simulator since they must both have the the same system definitions and operate on the same basic assumptions, the Diagnostic System must take a entirely different approach to complete its task. Instead of knowing the problems and determing the state variable values, this system must work in the opposite direction and attempt to determine what caused the current state. This is particularly difficult since in many occasions the state of the system may not be unique to a single set of faults. Therefore the Diagnostic System attempts to determine as many of the fault combinations that will cause the resulting system state as possible. As can be imagined, this can generate a large number of possible solutions. A simple approach was taken to associate a probability to each of the solutions based on the number of hardware faults necessary for that solution compared to the other candidates.

Two interrelated methods are used in the diagnostic systems approach in isolating the cause of the current state. First, several of the state variables values at specific locations within the system are dependent upon the value at the preceding location. This suggests the use of a relative or differential approach to these parameters. Most of the chemical properties of the system can be evaluated in this manner. Otherwise a technique that compares the actual parameter reading against nominal values can be utilized to determine spatial trends in readings resulting in probable fault group scenarios. The majority of the physical parameters are analyzed using this second approach. Finally, as mention above, the two methods are interrelated and a combined approach can tie segments of the system that are separated by some form of parameter discontinuity (such as two segments separated by a back pressure valve) together.

An example of the analysis process for the chemical parameters is included below. This should demonstrate some of the evaluation considerations that must be made in order to arrive at a complete solution. In the example, references are made to individual system locations. These are defined by the sub-component of the system where the item is located. For example, MF-C1 is the first conductivity meter (C1) located in the multifiltration unit (MF).

DIAGNOSIS RULES FOR CHEMICAL PARAMETERS - TOC, C, pH, I2 conc.

Conductivity - C

(1) When delta C between MF-C1 & MF-C2 becomes too small (ie. delta C< 50% MF-C1 value), then the first unibed needs replacement.

(2) When delta C between MF-C2 & MF-C3 becomes small (I suggest delta C < 80% MF-C2 value) this is a strong indication that sorbants are becoming spent, especially if condition (1) also applies.

(3) When delta C between MF-C1 & MF-C2 is good, bad delta C's down line may indicate bad C sensor.

(4) When any C sensor reads significantly higher (eg. 2X) than the preceeding sensor(s), and the preceeding sensor(s) are probably correct, then that sensor is bad as there are no significant sources of ions in the system.

NOTE: This only applies to MF and WQM sensors.The MCV's add ions.

(5) When delta C's are OK, but MF-C4 is >1.0 umho/cm, check CS-C1 to verify that input load is within design limit.

The Unibed purification cartridges are the sole physical/chemical means of reducing the conductivity and TOC of potable loop process water to meet water quality requirements. Test data available to this group on Unibed performance was limited (principly Doc. #52) and specific resins and sorbants for the unibeds are still under selection.

A simple linear approximation of removal rates and loadings was used for this diagnostic application. With this approximation the unibeds contaminant removal performance is stable over its operating life and degrades rapidly as it is saturated. Each bed removes an arbitrary fraction of the input load it experiences. In some cases this results in levels below the theoretical limit of purity for a properly operating system, but a more sophisticated "curve fit" based on more complete performance data would rectify this shortcoming.

TOTAL ORGANIC CARBON - TOC

(6) When WQM-TOC1 > 0.45 ppm, suspect that unibeds are becoming saturated. Check delta C's in MF , if they are mariginal recommend unibed change.

(7) When WQM-TOC1 > 0.5 ppm , and MF delta C's & WQM pH are good check TOC monitor calibration. If calibration is good, Check CS-TOC1 to verify that input load is within design limits. If input load is in limits and calibration is good replace unibeds regardless of conductivity performance.

TOC removal is solely accomplished by the Unibeds in the Multifiltration Subsystem. Problems with modeling the removal performance for TOC is exactly analogous to the conductivity case, and the same solution was employed.

pH

(8) When WQM-pH1 is high or low, <6 or >8 , the unibeds may be saturated. Check input loads, TOC, and conductivities to determine probable condition of the unibeds. pH meters are notorious for drift - recommend calibration check before action when other parameter values are acceptable. The limited test data available indicates that properly functioning unibeds invariably moderate or "buffer" the potable process water to within the specified water quality limits.

I2 CONCENTRATION

(9) When IM1 is low , check pH as pH out of range can cause low I2 concentration -If pH is bad check unibed condition as in (8) - if pH is OK recommend MCV replacement.

(10) When IM1 is high, check Iodine Monitor calibration. If calibration was OK then there is a source of excess I2, suspect damaged MCV.

Calculation of Iodine species concentraion in aqueous solutions is a very involved problem. Such an effort would involve solving simultanious equalibria equations involving teperature, pressure, pH , ionic strength, and equalibrium constants . Although aqueous halogen equalibrium has been studied for years, this system is so intractable that published equalibrium constants are questionable. Solving this problem in a scientifically robust manner would require a massive research effort.

For the purpose of this simulation/diagnostic effort it was assumed that properly functioning MCV's would continuously supply iodine at a 0.5 to 1.0 ppm equalibrium value. In other words, any process water entering an MCV with less than 0.5-1.0 ppm iodine would leave with this level, and process water which already contained this level would not receive additional iodine. Data from MCV performance tests indicate that MCV's do behave in this manner, imparting 0.5-1.0 ppm residual iodine regardless of the number of "passes" through the MCV.
We reasoned that if the iodine monitor is functioning and the iodine level is below 0.5 ppm , the MCV is spent or defective.

APPENDIX A

HYPERTALK SCRIPTS CODE

HYPERCARD STACK FILE LAST MODIFIED 1989 NOV 25 (SAT) 13:51:05

File name "syseclss"
The number of backgrounds is 1.
The number of cards is 11.

STACK SCRIPT

This script is executed when stack is opened. There are three possible options. At the initial opening the "Intro" card is used. If the simulator has just been used, results is performed. If the diagnostics has just been used, evaluate is performed.

```
on openStack
        set visible of field state to false
        set the userLevel to 5
        get first word of line 1 of field "status" of card "selector"
        if it is "Write" then
                results
        end if
        if it is "Evaluate" then
                evaluate
        end if
end openStack
```

This script is used to set the values for the sensors in each card based on the values in the current status file.

```
on results
        global g_select, g_statusMes
        put "Get state" into field "status"
        put empty into data
        put "current status" into fileName
        open file fileName
        read from file fileName until numToChar(0)
        put it into field "state"
        show field "state"
        put "data" into g_select
        put 1 into count
        repeat forever
                get line count of field "state"
                if it is empty then exit repeat
                delete first char of it
                delete last char of it
                put it into message
                if word 1 of it is "im1" then put "pdl-im1" into word 1 of it
                put it into message
                put offset("-", it) into pos
                delete char pos of it
                put " " after char pos-1 of it
                if word 1 of it is "plsa" then put "pdl" into word 1 of it
                put the length of word 3 of it into cSize
                if cSize > 6 then
                        repeat cSize - 6 times
                        delete last char of it
                        end repeat
                end if
                put word 3 of it into card field word 2 of it of card word 1 of it
                put count+1 into count
        end repeat
        show field "state"
        put "Sim data" into g_statusMes
        put "Sim data" into field "status"
        go to card 2
end results
```

This script is used to set the explanations of diagnostic results found in the diagnosis file.

```
on evaluate
        global g_select, g_statusMes
        put "Get diags" into field "status"
        put empty into data
        put "diagnosis" into fileName
        open file fileName
        read from file fileName until numToChar(0)
        put it into field "state"
        show field "state"
        put "diag" into g_select
        close file fileName
        put 1 into count
        repeat forever
                get line count of field "state"
                if it is empty then exit repeat
                delete first word of it
                delete last char of it
                if word 2 of it is "im1" then put "pdl-im1" into word 2 of it
                put it && the number of words of it into message
                put the number of words of it into faultList
                put offset("-", word 2 of it) into pos
                delete char pos of word 2 of it
                put " " after char pos-1 of word 2 of it
                if word 2 of it is "plsa" then put "pdl" into word 2 of it
                go to card word 2 of it
```

```
            put "The problem is" && word 1 of it & "." into line 1 of field 4
            put "Range is" && faultList-1 & "." into line 2 of field 4 <cont>
            put "The range is from" && word 2 of it & "-" & word 3 of it &&
        "to" && last word of it & "." into line 3 of field 4
            show card button word 3 of it
            set highlight of card button word 3 of it to true
            go to card "selector"
            put count+1 into count
        end repeat
end evaluate
```

OBJECT TYPE BKGD ID

BUTTON NUM 1 ID NAME "NEXT"

Button used to send the user to the next card.

SCRIPT

```
on mouseUp
        push card
        go to next card
end mouseUp
```

BUTTON NUM 2 NAME "PREV"

Button used to send the user to the previous card.

SCRIPT

```
on mouseUp
        go to prev card
end mouseUp
```

BUTTON NUM 3 NAME"RETURN"

Button used to send the user to the "selector" card.

SCRIPT

```
on mouseUp
        go to card "selector"
end mouseUp
```

CARD SCRIPT

A background script that is executed whenever an openCard message is generated.

```
on openCard
        global g_select, g_statusMes
        put g_statusMes into field "status"
        if g_select is "Res" then
                put the number of card fields into x
                repeat with count = 1 to x
                        set the lockText of card field count to true
                end repeat
                put the number of card buttons into y
                repeat with bcount = 1 to y
                        hide card button bcount
                end repeat
        end if
        pass openCard
end openCard
```

This background script is used by all sensor buttons to get data.

```
on getData
        global g_butName
        put empty into line 1 of field 4
        put "Select a fault" & return into line 1 of field 4
        if line 2 of field 4 is empty then delete line 2 of field 4
        put word 3 of g_butname into temp2
        delete first char of temp2
        delete last char of temp2
        put second word of the name of this card into temp1
        delete first char of temp1
        delete last char of temp1
        put "(problem" && temp1 & "-" & temp2 after last char of field 4
        show card button "Faults?"
        show card field "instruct"
        put empty into card field "instruct"
        put "Click on the 'Faults?' button" into first line of card field <cont>
        "instruct"
end getData
```

OBJECT TYPE CARD ID 2865 CARD NAME IS "INTRO"

CARD SCRIPT

This script adjusts the graphics presentation of the first card when then stack is first opened.

```
on openCard
        hide background button 1
        hide background button 2
        hide background button 3
        hide background picture
        hide background field 1
        hide background field 2
        hide background field 3
        hide background field 4
        show card field 1
        show card field 2
        pass openCard
end openCard
```

This script adjusts the graphics for when exiting the card.

```
on mouseDown
        show background picture
        show background button 1
        show background button 2
        show background button 3
        show background field 1
        show background field 3
        show background field 4
        go to last card
end mouseDown
```

OBJECT TYPE CARD ID 3611 CARD NAME IS "CS"

CARD SCRIPT

These scripts adjust the visibility of the navigation buttons.

```
on openCard
        hide background button 2
        pass openCard
end openCard

on closeCard
        show background button 2
end closeCard
```

SENSOR BUTTONS

BUTTON NUM 1 NAME "TOC1"
BUTTON NUM 2 NAME "C1"
BUTTON NUM 3 NAME "L1"
SCRIPT<SEE BELOW>

OTHER BUTTONS

```
BUTTON NUM 4 HIDDEN NAME "FAULTS?"
BUTTON NUM 5 HIDDEN NAME "VALUE"
BUTTON NUM 6 NAME "INFORMATION"
    SCRIPT<SEE BELOW>
```

OBJECT TYPE CARD ID 2202 CARD NAME IS "HES"

```
SENSOR BUTTONS
 BUTTON NUM 1 NAME "P1"
 BUTTON NUM 2 NAME "F1"
 BUTTON NUM 3 NAME "T1"
 BUTTON NUM 4 NAME "HE1"
 BUTTON NUM 5 NAME "HT"
 BUTTON NUM 6 NAME "T2"
 BUTTON NUM 7 NAME "RV"
 BUTTON NUM 8 NAME "F2"
 BUTTON NUM 9 NAME "T3"
 BUTTON NUM 10 NAME "P2"
 BUTTON NUM 11 NAME "P3"
 BUTTON NUM 12 NAME "F3"
 BUTTON NUM 13 NAME "PMP"
 BUTTON NUM 14 NAME "BPV"
     SCRIPT<SEE BELOW>

OTHER BUTTONS
 BUTTON NUM 15 HIDDEN NAME "FAULTS?"
 BUTTON NUM 16 HIDDEN NAME "VALUE"
 BUTTON NUM 17 NAME "INFORMATION"
     SCRIPT<SEE BELOW>
```

OBJECT TYPE CARD ID 4642 CARD NAME IS "MF"

CARD SCRIPT

This script adjusts the visibility of various fields on the card.

```
on openCard
 set visible of card field "fil" to false
 set visible of card field "ub1" to false
 set visible of card field "ub2" to false
 set visible of card field "ub3" to false
 set visible of card field "ub4" to false
 set visible of card field "ub5" to false
pass openCard
end openCard
```

SENSOR BUTTONS

```
 BUTTON NUM 1 NAME "P1"
 BUTTON NUM 2 NAME "C1"
 BUTTON NUM 3 NAME "NEW BUTTON"
 BUTTON NUM 4 NAME "C3"
 BUTTON NUM 5 NAME "C4"
 BUTTON NUM 6 NAME "P2"
 BUTTON NUM 7 NAME "FT1"
 BUTTON NUM 8 NAME "FIL"
 BUTTON NUM 9 NAME "UB1"
 BUTTON NUM 10 NAME "UB2"
 BUTTON NUM 11 NAME "UB3"
 BUTTON NUM 12 NAME "UB4"
 BUTTON NUM 13 NAME "UB5"
     SCRIPT<SEE BELOW>
```

OTHER BUTTONS

```
 BUTTON NUM 14 HIDDEN NAME "FAULTS?"
 BUTTON NUM 15 HIDDEN NAME "VALUE"
 BUTTON NUM 16 NAME "INFORMATION"
     SCRIPT<SEE BELOW>
```

OBJECT TYPE CARD ID 5211 CARD NAME IS "WQM"

CARD SCRIPT

This script adjusts the visibility of various fields on the card.

```
on openCard
 set visible of card field "pmp" to false
 set visible of card field "str" to false
pass openCard
end openCard
```

SENSOR BUTTONS

```
 BUTTON NUM 1 NAME "P1"
 BUTTON NUM 2 NAME "P2"
```

BUTTON NUM 3 NAME "PH1"
BUTTON NUM 4 NAME "P3"
BUTTON NUM 5 NAME "T1"
BUTTON NUM 6 NAME "C1"
BUTTON NUM 7 NAME "TOC1"
BUTTON NUM 8 NAME "PMP"
BUTTON NUM 9 NAME "STR"
SCRIPT<SEE BELOW>

OTHER BUTTONS
BUTTON NUM 10 HIDDEN NAME "FAULTS?"
BUTTON NUM 11 HIDDEN NAME "VALUE"
BUTTON NUM 12 NAME "INFORMATION"
SCRIPT<SEE BELOW>

OBJECT TYPE CARD ID 5528 CARD NAME IS "MPSA"

CARD SCRIPT

This script adjusts the visibility of various fields on the card.

```
on openCard
set visible of card field "he" to false
set visible of card field "ht1" to false
set visible of card field "ht2" to false
set visible of card field "acc" to false
set visible of card field "bpv" to false
pass openCard
end openCard
```

SENSOR BUTTONS
BUTTON NUM 1 NAME "T1"
BUTTON NUM 2 NAME "P1"
BUTTON NUM 3 NAME "T2"
BUTTON NUM 4 NAME "BPV"
BUTTON NUM 5 NAME "T3"
BUTTON NUM 6 NAME "HE"
BUTTON NUM 7 NAME "HT1"
BUTTON NUM 8 NAME "ACC"
BUTTON NUM 9 NAME "HT2"
SCRIPT<SEE BELOW>

OTHER BUTTONS
BUTTON NUM 10 HIDDEN NAME "FAULTS?"
BUTTON NUM 11 HIDDEN NAME "VALUE"
BUTTON NUM 12 NAME "INFORMATION"
SCRIPT<SEE BELOW>

OBJECT TYPE CARD ID 6244 CARD NAME IS "PPS"

SENSOR BUTTONS
BUTTON NUM 1 NAME "P1"
BUTTON NUM 2 NAME "F1"
BUTTON NUM 3 NAME "L1"
BUTTON NUM 4 NAME "L2"
BUTTON NUM 5 NAME "L3"
BUTTON NUM 6 NAME "L4"
SCRIPT<SEE BELOW>

OTHER BUTTONS
BUTTON NUM 7 HIDDEN NAME "FAULTS?"
BUTTON NUM 8 HIDDEN NAME "VALUE"
BUTTON NUM 9 NAME "INFORMATION"
SCRIPT<SEE BELOW>

OBJECT TYPE CARD ID 6825 CARD NAME IS "PDL"

CARD SCRIPT

This script adjusts the visibility of various fields on the card.

```
on openCard
 set visible of card field "pmp" to false
 set visible of card field "he" to false
 set visible of card field "ht1" to false
 set visible of card field "ht2" to false
 set visible of card field "acc" to false
 set visible of card field "bpv" to false
pass openCard
end openCard
```

SENSOR BUTTONS
BUTTON NUM 1 NAME "I1"

BUTTON NUM 2 NAME "C1"
BUTTON NUM 3 NAME "T2"
BUTTON NUM 4 NAME "T1"
BUTTON NUM 5 NAME "P1"
BUTTON NUM 6 NAME "T3"
BUTTON NUM 7 NAME "P2"
BUTTON NUM 8 NAME "F1"
BUTTON NUM 9 NAME "HE"
BUTTON NUM 10 NAME "HT1"
BUTTON NUM 11 NAME "HT2"
BUTTON NUM 12 NAME "ACC"
BUTTON NUM 13 NAME "PMP"
BUTTON NUM 14 NAME "BPV"
SCRIPT<SEE BELOW>

OTHER BUTTONS
BUTTON NUM 15 HIDDEN NAME "FAULTS?"
BUTTON NUM 16 HIDDEN NAME "VALUE"
BUTTON NUM 17 NAME "INFORMATION"
SCRIPT<SEE BELOW>

OBJECT TYPE CARD ID 4320 CARD NAME IS "PUS"

CARD SCRIPT

These scripts adjust the visibility of the navigation buttons.

```
on openCard
        hide background button 1
        pass openCard
end openCard

on closeCard
        show background button 1
end closeCard
```

SENSOR BUTTONS
BUTTON NUM 1 NAME "FT1"
BUTTON NUM 2 NAME "C1"
BUTTON NUM 3 NAME "P1"
SCRIPT<SEE BELOW>

OTHER BUTTONS
BUTTON NUM 4 HIDDEN NAME "FAULTS?"
BUTTON NUM 5 HIDDEN NAME "VALUE"
BUTTON NUM 6 NAME "INFORMATION"
SCRIPT<SEE BELOW>

SCRIPT FOR SENSOR BUTTONS

SCRIPT

This script is used by all sensors. It calls getData background script to do the actual work.g_butName is a global variable used in that script.

```
on mouseUp
        global g_butName
        put the name of me into g_butName
        getData
end mouseUp
```

SCRIPTS FOR OTHER BUTTONS

HIDDEN NAME "FAULTS?"

SCRIPT

This script actuates the appropriate menus for the sensor or device identified and generates part of the problem definition.

```
on mouseDown
        global g_butName, lyst, g_range
        put third word of g_butName into test
        delete first char of test
        delete last char of test
        if last char of test is in "1,2,3,4,5,6,7,8,9,0" then
                delete last char of test
        end if
        if test is "ub" then
                put "leak,blockage,spent" into lyst
        end if
        if test is "vlv" or test is "bpv" or test is "fil" or <cont>                test is "str" or
test is "pmp" then
                put "leak,blockage" into lyst
        end if
        if test is "he" or test is "ht" then
                put "leak,blockage,temperature" into lyst
        end if
        if test is "l" or test is "p" or test is "c" or test is "f" or <cont>            test is "t" or
test is "im" or test is "toc" or <cont>                          test is "ph" or test is "ft" then
                put "leak,blockage,bad-gauage" into lyst
        end if
        put the mouseloc into myPlace
        put item 1 of myPlace + 20 into horiz
        put item 2 of myPlace + 120 into vert
        get PopUpMenu(lyst, 6, vert, horiz)
        if test is "ub" and it is 3 then
                put " spent" after last character of last line of field 4
                put 100 into g_range
        else
                if (test is "he" or test is "ht") and it is 3 then
                        put " temperature" after last character of last line <cont>
        of field 4
                        put 100 into g_range
                else
                        if it is 3 then
                                put " bad-guage" after last character of last <cont>
                        line of field 4
                                put 200 into g_range
                        end if
                end if
        end if
        if it is 1 then
                put " leak" after last character of last line of field 4
                put 100 into g_range
        end if
        if it is 2 then
                put " blockage" after last character of last line of field 4
                put 100 into g_range
        end if
        put "Indicate a percentage" into line 1 of field 4
        hide card button "Faults?"
        show card button "value"
        show card field "instruct"
end mouseDown

on mouseUp
        put "Use guage for percent." into line 1 of card field "instruct"
end mouseUp
```

HIDDEN NAME "VALUE

SCRIPT

This script generates the values that complete the fault definition for the selected sensor or device.

```
on mouseDown
        global g_range
        set cursor to 2
        get the rect of me
        barbutton  3,g_range,0
        put " " & (.01 * the result) & ")" & return after last char of field 4
        hide card field "instruct"
        hide card button "value"
end mouseDown
```

NAME "INFORMATION"

SCRIPT

This script directs the access to the information card.

```
on mouseUp
        push card
        go to card "Info"
end mouseUp
```

OBJECT TYPE CARD ID 7761 CARD NAME IS "INFO"

CARD SCRIPT

These scripts control the navigation buttons for this card.

```
on openCard
        hide background button "return"
        show card button "return"
        pass openCard
end openCard

on closeCard
        show background button "return"
        pass closeCard
end closeCard
```

BUTTON NUM 1 ID NAME "RETURN"

SCRIPT

This script sends the user back to the card from which the info card was called.

```
on mouseUp
        pop card
end mouseUp
```

OBJECT TYPE CARD ID 7405 CARD NAME IS "SELECTOR"

CARD SCRIPT

This script arranges for the appropriate elements to be made visible to the user.

```
on openCard
        global g_statusMes
        put the number of card buttons into butTotal
        repeat with butNo = 1 to butTotal
                show card button butNo
        end repeat
        hide background button "return"
end openCard

on closeCard
        set visible of field "state" to false
        show background button "return"
        pass closeCard
end closeCard
```

BUTTON NUM 1 NAME "SET SIMULATION"

SCRIPT

This script initializes the fields and buttons of the various cards so that the user may begin to define faults.

```
on mouseUp
        global g_select, g_statusMes
        put "sim" into g_select
        put "Simulation" into g_statusMes
        put "Wait" into field status
        set lockScreen to true
        repeat with cardNo = 2 to 9
                go to card cardNo
                put "Simulation" into field "status"
                if g_select is "sim" then
                        put the number of card fields into x
                        repeat with count = 1 to x
                                put empty into card field count
                        set lockText of card field count to false
                        put the name of card field count into nameField
                        get third word of nameField
                        delete first char of it
                        delete last char of it
                        put it into card field count
                end repeat
                put the number of card buttons into y
                repeat with bcount = 1 to y
                        show card button bcount
                        set highlight of card button bcount to false
                end repeat
        end if
        hide card button "Faults?"
        hide card button "value"
        hide card field "instruct"
        put empty into background field 4
    end repeat
    put "Simulation" into field "status" of card "selector"
    go to card 2
    set lockScreen to false
end mouseUp
```

BUTTON NUM 2 NAME "RUN SIMULATOR"

SCRIPT

This script gathers the user defined faults, writes them to a file and calls CLIPS with the appropriate file.The openStack script will direct the application of the results handler.

```
on mouseUp
    global g_status, g_select, g_statusMes
    put "Get faults" into g_statusMes
    put "Res" into g_select
    repeat with cardNo = 2 to 9
        go to card cardNo
        get line 1 of field 4
        if last char of it is ")" then put it & return after last <cont>
    char of field 4 of card "selector"
        get line 2 of field 4
        if last char of it is ")" then put it & return after¬
        last char of field 4 of card "selector"
    end repeat
    go to last card
    put "Write faults" into field "status"
    open file "problems"
    put the number of lines of field 4 into n
    repeat with i = 1 to n
        write line i of field 4 to file "problems"
        write return to file "problems"
        end repeat
        close file "problems"
        push card
        open "eclss simulator" with "clips"
end mouseUp
```

BUTTON NUM 3 NAME "RUN DIAGNOSTICS"

This script calls CLIPS with the appropriate file that performs the diagnostics on the current state of the system.The openStack script will direct the application of the evaluate handler.

SCRIPT

```
on mouseUp
        push card
        evaluate
end mouseUp
```

BUTTON NUM 4 NAME "CLEAR"

This script clears syseclss and prepares it to be used again.

SCRIPT

```
on mouseUp
        global g_select, g_statusMes
        put empty into g_select
        put "Clear" into g_statusMes
        put g_statusMes into field "status"
        repeat with cardCount = 2 to 9
                go to card cardCount
                put empty into field 4
                put the number of card fields into countFields
                repeat with fieldCount = 1 to countFields
                        put empty into card field fieldCount
                end repeat
        end repeat
        put empty into field "state"
        go to last card
        put empty into field 4
        doMenu "Compact Stack"
end mouseUp
```

APPENDIX B

SIMULATOR RULES AND SYSTEM DEFINITION

SYSTEM DEFINITION

```
(flowrate fi .00775)
(toc fi 20)
(iodine fi 2)
(conductivity fi 120)
(ph fi 6.0)
(pressure fe 35)
(temperature fi 45)
(nodes A fi cs-toc1 cs-c1 cs-l1 hes-p1 hes-f1 hes-t1 hes-he1 hes-ht hes-t2
hes-rv hes-f2 hes-he2 hes-t3 hes-p2 hes-bpv hes-p3 hes-f3 mf-fil mf-p1
mf-c1 mf-ub1 mf-c2 mf-ub2 mf-c3 mf-ub3 mf-ub4 mf-ub5 mf-c4 mf-ft1
mf-p2 mf-vlv wqm-pmp wqm-p1 wqm-str wqm-p2 wqm-ph1 wqm-p3 wqm-
t1 wqm-c1 wqm-toc1 mpsa-t1)
(nodes B mpsa-t1 mpsa-he1 mpsa-ht mpsa-t2 mpsa-acc mpsa-he2 mpsa-p1
mpsa-t3 mpsa-bpv pps-p1 pps-f1 pps-l1 pps-l2 pps-l3 pps-l4 plsa-t1
plsa-he1 plsa-ht plsa-t2 plsa-acc plsa-he2 plsa-p1 plsa-t3 plsa-bpv im1
pdl-c1 pdl-p1 pdl-f1pus-ft1 pus-c1 pus-p1 fe)
(gauges pressure hes-p1 hes-p2 hes-p3 mf-p1 mf-p2 wqm-p1 wqm-p2 wqm-
p3 mpsa-p1
 pps-p1 plsa-p1 pdl-p1 pus-p1)
(gauges temperature hes-t1 hes-t2 hes-t3 wqm-t1 mpsa-t1 mpsa-t2 mpsa-t3
plsa-t1 plsa-t2 plsa-t3)
(gauges flowrate hes-f1 hes-f2 hes-f3 pps-f1 pdl-f1)
(gauges conductivity cs-c1 mf-c1 mf-c2 mf-c3 mf-c4 wqm-c1 pdl-c1 pus-
c1)
(gauges ph wqm-ph1)
(gauges toc cs-toc1 wqm-toc1)
(gauges level cs-l1 pps-l1 pps-l2 pps-l3 pps-l4)
(gauges iodine im1)
(gauges flow-total mf-ft1 pus-ft1)
(heaters hes-he1 hes-ht hes-rv hes-he2 mpsa-he1 mpsa-ht mpsa-acc mpsa-
he2 plsa-he1 plsa-ht plsa-acc plsa-he2)
(heater-delta-t 20 40 20 -40 20 40 20 -40 20 40 20 -40)
(delta-p relative wqm-str .25)
(delta-p absolute hes-bpv mpsa-bpv plsa-bpv 37.3)
(delta-p relative mf-fil .75)
(delta-p relative mf-ub1  2.5)
(delta-p relative mf-ub2  2.5)
(delta-p relative mf-ub3  2.5)
(delta-p relative mf-ub4   2.5)
(delta-p relative mf-ub5  2.5)
(spent mf-ub1 0)
(spent mf-ub2 0)
(spent mf-ub3 0)
(spent mf-ub4 0)
```

```
        (spent mf-ub5 0)
```

Simulator Rules

```
;;  Eclss Monitoring Prototype - Simulator

;; Physical Components

(defrule no-leak
        (flowrate ?location ?value)
        (not (problem ?location leak ?percentage))
        (nodes ?key $?any ?location ?next $?rest)
        (not (flowrate ?next ?))
=>
        (assert (flowrate ?next ?value)))

(defrule leak ;"Reflect flow changes due to Leaks"
        ?problem <- (problem ?location leak ?percentage)
        ?original     <- (flowrate ?location ?value)
=>
        (retract ?original ?problem)
        (bind ?value (* (- 1 ?percentage) ?value))
        (assert (flowrate ?location ?value)))

(defrule temperature-problem
        ?prob <- (problem ?location heater ?percentage)
        (heaters $?start)
        ?vals <- (heater-delta-t $?heat)
=>
        (retract ?prob ?vals)
        (bind ?pos (member ?location  $?start))
        (bind ?val (* (nth ?pos $?heat) ?percentage))
        (bind $?fseq (mv-subseq 1 (- ?pos 1) $?heat))
        (bind $?lseq (mv-subseq (+ 1 ?pos) (length $?heat) $?heat))
        (assert (heater-delta-t $?fseq ?val $?lseq)))

(defrule propagate-temperature
        (temperature ?location ?value)
        (not (heaters $?start ?location  $?end))
        (nodes ?key $?any ?location ?next $?rest)
        (not (temperature ?next ?))
=>
        (assert (temperature ?next ?value)))

(defrule  temperature-changes-at-a-source-or-sink
        ?heater <- (heaters $?start ?location  $?rest)
        (nodes ? $? ?location ?next $?)
        (heater-delta-t  $?heat)
  ?original     <- (temperature ?location ?value)
=>
        (bind ?delta (nth (+ 1 (length $?start)) $?heat))
        (retract ?original ?heater)
        (bind ?value (+  ?delta ?value))
        (assert (temperature ?location ?value)
                (heaters $?start used $?rest)))

(defrule blockage ;reflect pressure change due to blockage
        ?problem <- (problem ?location blockage ?percentage)
        ?original <- (pressure ?location ?value)
=>
        (retract ?original ?problem)
        (bind ?value (* ?value (+ 1 ?percentage)))
        (assert (pressure ?location ?value)))

(defrule pressure-change-at-filters-and-unibeds
        ?node <- (pressure ?location ?value)
        ?press <- (delta-p ?type $?before ?location $?after ?delta)
=>
        (retract ?node ?press)
        (if (eq ?type relative)
            then
                (bind ?value (+ ?value ?delta))
            else
                (bind ?value ?delta))
        (assert (pressure ?location ?value)
                (delta-p ?type $?before $?after ?delta)))

(defrule back-propagate-pressure
        (pressure ?location ?value)
        (not (delta-p ?location ?delta))
        (not (problem ?location blockage ?percentage))
        (nodes ?key $?any ?next ?location $?rest)
        (not (pressure ?next ?))
=>
        (assert (pressure ?next ?value)))

(defrule bad-gauge
        (declare (salience -100))
        ?prob <- (problem ?location bad-gauge ?percentage)
        (gauges ?type $? ?location $?)
        ?press <- (?type ?location ?value)
=>
        (retract ?prob ?press)
        (bind ?value (* ?value ?percentage))
        (assert (?type ?location ?value)))

;; System Chemistry

(defrule spent-unibed
        (declare (salience 1000))
        ?prob <- (problem ?location spent ?percentage)
        ?uni <- (spent ?location ?value)
=>
        (retract ?prob ?uni)
        (assert (spent ?location ?percentage)))

(defrule propagate-chemical-properties
        (iodine ?location ?iodine)
        (conductivity ?location ?conductivity)
        (ph ?location ?ph)
        (toc ?location ?toc)
        (not (spent ?location ?percentage))
        (nodes ?key $?any ?location ?next $?rest)
        (not (toc ?next ?))
=>
        (assert (iodine ?next ?iodine)
                (conductivity ?next ?conductivity)
                (toc ?next ?toc)
                (ph ?next ?ph)))

(defrule chemical-change-at-unibeds
        ?iold <- (iodine ?location ?iodine)
        ?cold <- (conductivity ?location ?conductivity)
        ?pold <- (ph ?location ?ph)
        ?told <- (toc ?location ?toc)
        (nodes ?key $?any ?location ?next $?rest)
        (spent ?location ?percentage)
        (not (toc ?next ?))
=>
        (retract ?told ?iold ?cold ?pold)
        (bind ?percentage (log10 (+ 1 (* ?percentage 9))))
        (bind ?toc (* (+ .25 (* .75 ?percentage)) ?toc))
        (bind ?ph (+ 7.0 (* (+ .5 (* .5 ?percentage)) (- ?ph 7.0))))
        (bind ?conductivity (* (- 1 (* .95 (- 1 ?percentage))) ?conductivity))
        (bind ?iodine ?iodine)
        (assert (iodine ?location ?iodine)
                (conductivity ?location ?conductivity)
                (toc ?location ?toc)
                (ph ?location ?ph))
        (assert (iodine ?next ?iodine)
                (conductivity ?next ?conductivity)
                (toc ?next ?toc)
                (ph ?next ?ph)))


;; Data Transfer and Initial Set-up

(defrule Initial-facts
        (declare (salience 9000))
        (initial-fact)
=>
        (load-facts "system node definitions")
        (load-facts "problems"))

(defrule open-output-file
        (declare (salience -1000))
        (initial-fact)
=>
        (open "current status" output "w"))

(defrule close-output-file
        (declare (salience -10000))
        (initial-fact)
=>
        (close))

(defrule  write-output-to-file
        (declare (salience -2000))
        (initial-fact)
        (gauges ?type $? ?location $?)
        (?type ?location ?value)
=>
        (fprintout output "(" ?location " " ?value ")" crlf))
```

APPENDIX C

DIAGNOSTIC RULES

```
;;

(defrule read-in-data
                (initial-fact)
=>
                (load-facts "Dan's data set"))

(defrule  find-min-same-node-list-differential
        (declare (salience 5000))
        (not (clue ?loc ?))
        (differential $? ?type $?)
        (gauges ?type $? ?prev ?loc $?)
        (nodes ?key  $?first ?prev $?middle ?loc $?)
        (delta-lower ?type ?key $?vals)
=>
        (bind ?pos (+ 1 (length $?first)))
        (bind ?count (length $?middle))
        (bind ?sum 1)
        (while (> ?count -1)
                        (bind ?pos (+ 1 ?pos))
                        (bind ?sum (* (- 1 (nth ?pos $?vals)) ?sum))
                        (bind ?count (- ?count 1)))
        (bind ?result (- ?old ?sum))
        (if (> ?result ?new)
                                        (assert (clue ?loc low))
;
        (nominal minimum ?loc ?result))
                else
                                        (assert (check-high ?loc))))

(defrule  find-min-different-node-list-differential
        (declare (salience 5000))
        (not (clue ?loc ?))
        (gauges ?type $? ?prev ?loc $?)
        (differential $? ?type $?)
        (?prev ?old)
        (?loc ?new)
        (nodes A $?first ?prev $?mid)
        (nodes B $?next ?loc $?)
        (delta-lower ?type A $?val1)
        (delta-lower ?type B $?val2)
=>
        (bind $?middle (mv-append $?mid $?next))
        (bind $?vals (mv-append (mv-subseq (+ 2 (length $?first))
                                                                                           (+ 1
(length  $?first)(length  $?mid))



        $?val1)


        (mv-subseq 1  (length $?next)  $?val2)))
        (bind ?pos 0)
        (bind ?count (length $?middle))
        (bind ?sum 1)
        (while (> ?count 0)
                        (bind ?pos (+ 1 ?pos))
                        (bind ?sum (* (- 1 (nth ?pos $?vals)) ?sum))
                        (bind ?count (- ?count 1)))
        (bind ?result (- ?old ?sum))
        (if (> ?result ?new)
                                        (assert (clue ?loc low))
;
        (nominal minimum ?loc ?result))
                else
                                        (assert  (check-high  ?loc))))


(defrule  find-max-same-node-list-differential
        (declare (salience 5000))
        ?high <- (check-high ?loc)
        (gauges ?type $? ?prev ?loc $?)
        (?prev  ?old)
        (?loc  ?new)
        (nodes ?key  $?first ?prev $?middle ?loc $?)
        (delta-lower ?type ?key $?vals)
=>
 (retract ?high)
        (bind ?pos (+ 1 (length $?first)))
        (bind ?count (length $?middle))
        (bind ?sum 1)
        (while (> ?count -1)
                        (bind ?pos (+ 1 ?pos))
                        (bind ?sum  (* (- 1 (nth ?pos $?vals)) ?sum))
                        (bind ?count (- ?count 1)))
        (bind ?result (* ?old ?sum))
        (if (> ?result ?new)
                                        (assert (clue ?loc agree))
                else
                                        (assert (clue ?loc high))))

(defrule  find-max-different-node-list-for-differential
        (declare (salience 5000))
        ?high <- (check-high ?loc)
        (gauges ?type $? ?prev ?loc $?)
        (?prev  ?old)
        (?loc  ?new)
        (nodes  A  $?first  ?prev  $?mid)
        (nodes  B  $?next  ?loc  $?)
        (delta-upper  ?type  A  $?val1)
        (delta-upper  ?type  B  $?val2)
=>
        (retract  ?high)
        (bind  $?middle  (mv-append  $?mid  $?next))
        (bind  $?vals  (mv-append  (mv-subseq  (+  2  (length  $?first))


                                                                        (+  1
(length  $?first)(length  $?mid))



        $?val1)


        (mv-subseq  1  (length  $?next)  $?val2)))
        (bind  ?pos  0)
        (bind  ?count  (length  $?middle))
        (bind  ?sum  1)
; (fprintout  t  "variables  "  $?middle  crlf  "values"  $?vals  crlf)
        (while  (>  ?count  0)
                        (bind  ?pos  (+  1  ?pos))
                        (bind  ?sum  (*  (-  1  (nth  ?pos  $?vals))  ?sum))
                        (bind  ?count  (-  ?count  1)))
        (bind  ?result  (-  ?old  ?sum))
        (if  (>  ?result  ?new)
                                        (assert  (clue  ?loc  agree))
;
        (nominal  minimum  ?loc  ?result))
                else
                                        (assert  (clue  ?loc  high))))

(defrule  group-gauges-phase-I
                (declare  (salience  4900))
                (gauges  ?type  $?  ?loc  $?)
                ?old  <-  (clue  ?loc  ?clue)
                ?new  <-  (unsorted-gauge-clues  ?type  $?clues)
        =>
                (retract  ?old  ?new)
                (bind  $?clues  (mv-append  $?clues  ?loc  ?clue))
                (assert  (unsorted-gauge-clues  ?type  $?clues)))

(defrule  sort-grouped-gages
                (declare  (salience  4500))
                ?old  <-  (unsorted-gauge-clues  ?type  $?clues)
                (gauges  ?type  $?gauges)
=>
                (retract  ?old)
                (bind  ?count  1)
                (bind $?total nil)
                (until (> ?count (length $?gauges))
                                         (bind ?val (nth (+ 1 (member (nth ?count $?gauges)
$?clues)) $?clues))
                                         (bind $?total (mv-append $?total ?val))
                                         (bind ?count (+ 1 ?count)))
                (assert (gauge-clues ?type $?total)))

(defrule compare-gauges-against-absolute-readings
                (?loc ?value)
                (nominal ?loc ?nom)
                (gauges ?type $?first ?loc $?)
                (delta ?type $?deltas)
=>
                (bind ?diff (- ?value ?nom))
                (bind ?del (nth (+ (length $?first) 1) $?deltas))
                (if (> ?diff ?del)
                                                (assert (clue ?loc high))
                        else
                                                (if (< (+ ?diff ?del) 0)
                                                                        (assert (clue ?loc
low))
                                                    else (assert (clue ?loc agree)))))

(defrule Check-differential-gauge-pattern-changes
                (gauge-clues ?type $?pattern)
                (gauges ?type $?gauges)
                (differential-gauges $? ?type $?)
=>
                (bind ?n 1)
                (bind ?mode agree)
                (bind $?changes nil)
                (while (<= ?n (length $?pattern))
                     (bind ?current (nth ?n $?pattern))
                     (if (not (equal ?current agree))
                        (bind ?j ?n)
                        (if (null $?changes)
                           (bind $?changes (mv-append $?changes
                            (- ?n 1)))
                            (if (equal ?current high)
                               (bind ?test low)
                          else
                               (bind ?test high))
                            (until (or (= ?j (length $?pattern))
                                   (equal (nth (+ ?j 1) $?pattern)
                                       ?test))
                                (bind ?j (+ ?j 1)))
                           (bind $?changes (mv-append $?changes
                                                                        (+1 (- ?j ?n))))))
                (assert (gauge-changes ?type $?changes)))

(defrule check-differential-gauge-patterns-for-bad-gauges
                (gauge-clues ?type $?pattern)
                (gauges ?type $?gauges)
                (gauge-changes ?type $?changes)
                (differential-gauges $? ?type $?)
=>
                (bind ?n 1)
                (bind ?pos 1)
                (if (not (null $?changes))
                    (while (< ?n 3)
                              (bind ?current (nth ?pos $?pattern))
                              (if (equal ?current agree)
                                    (if (equal (nth (+ ?pos (nth 2 $?changes))
                                                    $?pattern) high)
                                          (bind ?current low)
                                 else
                                          (bind ?current high))
                                          (bind ?j ?n)
                                          (bind ?new-pos ?pos)
                                          (bind $?hard-faults-loc nil)
                                          (until (>= ?j (length $?pattern))
                                                  (bind ?end-pos (+ ?new-pos
                                                                    (nth ?j $?pattern)))
                                                  (bind $?hard-faults-loc (mv-append
                                                $?hard-fault-loc
                                                        (mv-subseq ?new-pos
                                                            (+ ?new-pos ?end-pos) $?gauges)))
                                                  (bind ?new-pos (+ ?end-pos
                                                                (nth (+ j 1) $?pattern) 1))
                                                  (bind ?j (+ j 2)))
                                         (assert (problem bad-gauges ?current
                                                                $?hard-faults-loc))
                              (bind ?pos (+ ?pos (nth 2 $?changes)))
                              (bind ?n (+ 1 ?n)))))

(defrule group-pressure-gauge-patterns
                (gauges pressure $?gauges)
                (gauge-clues pressure $?changes)
                (bpvs $?boundary)
                (nodes A $?nodeA)
                (nodes B $?nodeB)
=>
                (bind $?nodes (mv-append $?nodeA $?nodeB))
                (bind ?n 1)
                (bind ?j 1)
                (bind ?upper (member (nth 1 $?boundary) $?nodes))
                (bind $?pgseq nil)
                (bind $?pgclue nil)
                (while (> (length $?gauges) ?n)
                          (bind ?n (+ 1 ?n))
                          (bind ?pg (nth ?n $?gauges))
                          (bind ?clue (nth ?n $?changes))
                          (bind ?loc (member ?pg $?nodes))
                          (if (> ?loc ?upper)
                              (if (and (not (null $?pgseq))
                                          (< ?j (length $?boundary)))
                                  (assert (pressure-sequence $?pgseq clue $?pgclue))
```

```
                                        (bind $?pgseq nil)
                                         (bind $?pgclue nil))
                                   (while (or (> ?loc ?upper)
                                                 (< ?j (length $?boundary)))
                                             (bind ?j (+ 1 ?j))
                                             (bind ?upper (member (nth ?j $boundary)
                                                                     $?nodes)))
                                   (bind ?n (- ?n 1))
                              else
                                   (bind $?pgseq (mv-append $?pgseq ?pg))
                                    (bind $?pgclue (mv-append $?pgclue ?clue))))
                    (if (not (null $?pgseq))
                         (assert (pressure-sequence $?pgseq clue $?pgclue))))

(defrule  Analyze-pressure-gauge-readings-for-bad-gauges
                    (pressure-sequence $?gauges clue $?clues)
=>
                    (if (not (null $?clues))
                         (bind ?n 1)
                         (bind $?choices (mv-append low agree high))
                         (while (< ?n 4)
                                   (bind ?choice (nth ?n $?choices))
                                   (bind ?pos 1)
                                   (bind ?times 0)
                                   (Until (> ?pos (length $?clues))
                                             (bind ?current (nth ?pos $?clues))
                                             (if (not (equal ?current ?choice))
                                                 (bind $?bad-gauges
                                                        (mv-append $?bad-gauges
                                                                  (nth ?pos $?gauges)))
                                             else
                                                 (bind ?times (+ ?times 1)))
                                             (bind ?pos (+ 1 ?pos)))
                                   (bind ?n (+ 1 ?n))
                                   (if (and (not (null $?bad-gauges))
                                               (or (> ?times 0)
                                                    (equal ?choice agree)))
                                        (assert (problem bad-gauges $?bad-gauges))))))

(defrule  group-temperature-gauge-patterns
                    (gauges temperature $?gauges)
                    (gauge-clues temperature $?changes)
                    (bpvs $?boundary)
                    (nodes A $?nodeA)
                    (nodes B $?nodeB)
=>
                    (bind $?nodes (mv-append $?nodeA $?nodeB))
                    (bind ?n 1)
                    (bind ?j 1)
                    (bind ?upper (member (nth 1 $?boundary) $?nodes))
                    (bind $?pgseq nil)
                    (bind $?pgclue nil)
                    (while (> (length $?gauges) ?n)
                              (bind ?n (+ 1 ?n))
                    (bind ?pg (nth ?n $?gauges))
                   .(bind ?clue (nth ?n $?changes))
                    (bind ?loc (member ?pg $?nodes))
                    (if (> ?loc ?upper)
                        (if (and (not (null $?pgseq))
                                     (< ?j (length $?boundary)))
                             (assert (temperature-sequence $?pgseq clue
                                          $?pgclue))
                             (bind $?pgseq nil)
                             (bind $?pgclue nil))
                        (while (or (> ?loc ?upper)
                                     (< ?j (length $?boundary)))
                                 (bind ?j (+ 1 ?j))
                                 (bind ?upper (member (nth ?j $boundary
                                                         $?nodes)))
                                 (bind ?n (- ?n 1))
                               else
                                   (bind $?pgseq (mv-append $?pgseq ?pg))
                                   (bind $?pgclue (mv-append $?pgclue
                                                                ?clue))))
            (if (not (null $?pgseq))
                (assert (temperature-sequence $?pgseq clue $?pgclue))))

(defrule  check-temperature-sequence-patterns-for-bad-gauges
            (gauge-clues ?type $?pattern)
            (gauges ?type $?gauges)
            (gauge-changes ?type $?changes)
            (differential-gauges $? ?type $?)
=>
            (bind ?n 1)
            (bind ?pos 1)
            (if (not (null $?changes))
                (while (< ?n 3)
                        (bind ?current (nth ?pos $?pattern))
                       (if (equal ?current agree)
                             (if (equal (nth (+ ?pos (nth 2 $?changes))
                                                 $?pattern) high)
                                 (bind ?current low)
                          else
                                 (bind ?current high))
                                 (bind ?j ?n)
                                 (bind ?new-pos ?pos)
                                 (bind $?hard-faults-loc nil)
                                 (until (>= ?j (length $?pattern))
                                          (bind ?end-pos (+ ?new-pos
                                                              (nth ?j $?pattern)))
                                           (bind $?hard-faults-loc
                                                   (mv-append $?hard-fault-loc
                                                        (mv-subseq ?new-pos
                                                              (+ ?new-pos ?end-pos)
                                                               $?gauges)))
                                          (bind ?new-pos (+ ?end-pos (nth (+ j 1)
                                                                  $?pattern) 1))
                                          (bind ?j (+ j 2)))
(assert (problem bad-gauges ?current
                             $?hard-faults-loc))
(bind ?pos (+ ?pos (nth 2 $?changes)))
 (bind ?n (+ 1 ?n)))))
```

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle ECLSS Advanced Automation: Preliminary Requirements --Final Report		5. Report Date 19 December 1989
		6. Performing Organization Code
7. Author(s) James W. McKee		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address University of Alabama, Huntsville		11. Contract or Grant No. NAS8-36955
12. Sponsoring Agency Name and Address NASA/MSFC		13. Type of Report and Period Covered Plan 4-1-89 thru 10-31-89
		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

In analyzing the baseline ECLSS command and control architecture, we may find various processes which would be enhanced by the use of knowledge based system methods of implementation. The purpose of this project is to document the most suitable process for prototyping using rule based methods, while also considering domain knowledge resources and other practical considerations.

Requirements for a prototype rule based software system will be documented. These requirements will reflect Space Station Freedom ECLSS software and hardware developments efforts, and knowledge based system requirements. A quick prototype knowledge based system environment will be researched and developed.

17. Key Words (Suggested by Author(s)) ECLSS, Final report	18. Distribution Statement Unclassified-unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 160	22. Price

NASA FORM 1626 OCT 86