Designing Real-Time Systems with Intelligent Sensors

Thesis

Presented to the Graduate Council of
Southwest Texas State University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Jerry D. Cavin
B.S., Computer Science
A.A.S. Electronic Technology

San Marcos, Texas

April, 1993

ACKNOWLEDGMENTS

I would like to express my sincere appreciation and gratitude to my thesis supervisor Dr. Janusz Zalewski, for patient assistance throughout this project. Additional thanks are due to Dr. Thomas McCabe, and Dr. Carol Hazelwood for their service as members of my thesis committee.

I would also like to thank Alvin Moczygemba, Maintenance Supervisor of the Austin Dekker Lane Power Plant, for the tour and detailed explanation of the inner working of power plant.

Table of Content

1.0. INTRODUCTION 6
1.1. THE NATURE OF INTELLIGENT SENSORS 8
1.2. EXAMPLES OF INTELLIGENT SENSORS 11
1.3. SUMMARY 17
2.0. SYSTEM DESIGN WITH INTELLIGENT SENSORS 18
2.1. DATA FUSION 18
2.1.1. PROBABILISTIC DATA FUSION 18
2.1.2. THE LEAST-SQUARE METHOD 18
2.1.3. KALMAN FILTERING 21
2.1.4 SUMMARY 22
2.2. FAULT TOLERANCE 24
2.2.1 CONFIGURATION FLEXIBILITY 24
2.2.2. SAFER SYSTEMS 26
2.2.3. SUMMARY 29
2.3.1. PHYSICAL LAYER 32
2.3.2 DATA LINK LAYER 34
2.3.3. XTP -- THE XPRESS TRANSFER PROTOCOL 37
2.3.4. NTP -- THE NETWORK TIME PROTOCOL 39
2.3.5. EXTERNAL DATA REPRESENTATION 40
2.3.6 SUMMARY 43
2.4. GLOBAL STATE AWARENESS 45
2.4.1. GLOBAL AND LOCAL STATE TABLES 45
2.4.2. SCI - SCALABLE COHERENT INTERFACE 46
2.4.3. SUMMARY 46
2.5 LOCAL PROPERTIES OF THE INTELLIGENT SENSORS 47
2.5.1 THE KERNEL 47
2.5.2. TASK SCHEDULING 48
2.5.3. TIME 48
3.0. PROBLEM FORMULATION AND PROCEDURE 50
4.0. REAL-TIME SYSTEM EXAMPLE 51
4.1. BOILER 51
4.2 REMAINING COMPONENTS 55
5.0. REAL-TIME SYSTEM DESIGN 57
5.1 THE SYSTEM OBJECT 62
5.2. THE TURBINE OBJECT 63
5.3. THE GENERATOR OBJECT 64
5.4 THE CONDENSER OBJECT 65
5.5 THE DEAERATOR OBJECT 66
5.6. THE BOILER OBJECT 67

6.0 IMPLEMENTATION 75
REFERENCES 78
APPENDIX A 80
APPENDIX B 87
APPENDIX C 156

1.0. INTRODUCTION

Contemporary industrial applications using real-time systems tend to be very large and complex. They may include hundreds of processors scattered over large areas connected together by a communication network. Many of toady's real-time projects require systems containing up to one million sensors that are collecting data at a hundred to a thousand times a second. If real-time systems are to continue to be a viable solution for massively complex systems, then the current real-time system architecture will need to be improved. This level of complexity can only be handled by distributing the intelligence, or processing capabilities, throughout the system. In the past, the majority of the processing has been done by the central computer of the real-time system as shown in Figure 1a, and more recently configured as in Figure 1b.

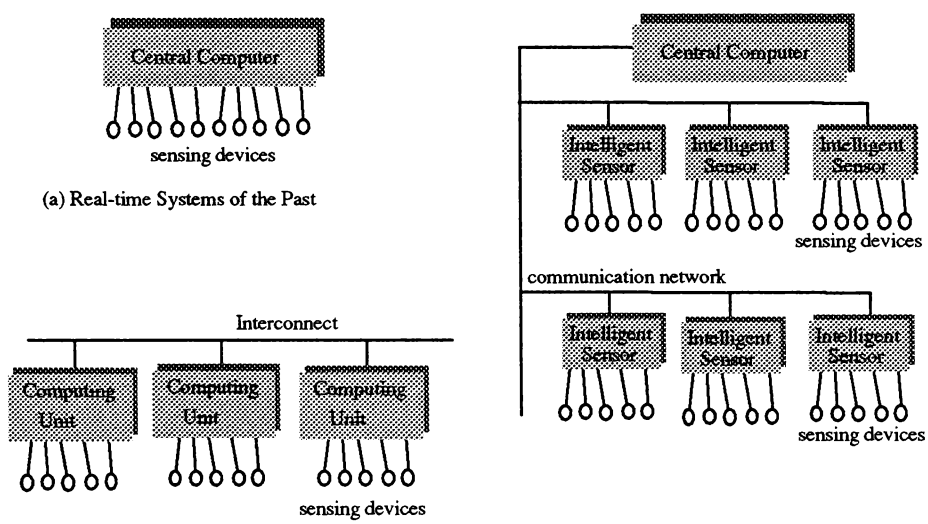

Figure 1 - Today's and Tomorrow's Real-time Systems

One of the leaders in real-time system design, John Stankovic pointed out

"The next generation of real-time-systems will be in similar application areas as current systems, but will be more complex in that they will be distributed, contain highly dynamic and adaptive behavior, exhibit intelligent behavior, have long lifetimes and be characterized as having catastrophic consequences if the logical or timing constraints of the system are not met." [12]

These future requirements will force the sensors of real-time systems to gain more intelligence (see Figure 1c). This means turning responsibility over to the sensor to perform not only measurement and control functions but also analyze of the data taken, and to some extent, partake in decision making.

1.1. THE NATURE OF INTELLIGENT SENSORS

The idea central to the intelligent sensor is that the majority of the processing is done by the intelligent sensor rather than the main processor. The central computer then becomes a site for data storage and transferal, and to maintain control over the entire system. The overall design of a system designed with intelligent sensors is a combination of the classic real-time system and a distributed control system. The classic real-time system in this sense is defined as being a system that has stringent timing constraint requirements, achievable through proper task scheduling and task pre-emption. A distributed control system is a system that maintains control over a number of instruments and sensors over a large network. The distributed control system usually has few, or no timing constraints. By combining the timing constraint requirements of the classic real-time system and the control network of a distributed control system it is possible to control a large number of intelligent sensors.

Real-time systems have been increasing in complexity every year as they are called upon to handle more sophisticated duties. The higher level of complexity requires the system designer to build systems that can meet timing constraints that were impossible a few years ago. Another problem of system designers is task pre-emption, which adds a great amount of overhead to the time required to execute a task given a signal from the outside environment. Recently this has meant simply to build faster and faster chips to be used in embedded applications. For many years CPU clock speeds have been increasing dramatically to keep up with the demand for faster machines. Today the chip designers are reaching the theoretical limits of the micro connections on the computer chips. This should be a major concern for the real-time systems designers who still have projects on the drawing boards hoping for future technologies to solve their problems. This bottleneck for real-time systems has been caused by localizing the system functions of task

scheduling and task pre-emption into a central processor. In their study of advanced sensor systems Muntz and Horowitz pointed out these as the "*deficiency of the classical execution model for hard real-time systems*" .[10] Only by using a more distributed real-time architecture will the problems of scheduling and pre-emption be solved.

The distribution of functions and delegating them to sensors. mean that intelligence of sensors must increase significantly, making them capable to perform the following:

1. *Preprocessing*

The intelligent sensor must, using a local processor, perform data preprocessing, (i.e. data fusion), upon the measurements taken from the sensors. The results of the analysis must contain information about the system's environment which can not be extracted from the measurements alone. The processor then performs actions based upon the analyzed data. The class of the local processor used in the intelligent sensor may range from highly specialized embedded micro controller to an 8, 16, or 32-bit micro computer.

2. *Fault-Tolerance*

If, for any reason, the intelligent sensor fails it must not cause a failure of the entire system. The fault should be detected and steps taken to recover from the failure, or to fail gracefully in a fail-safe manner, and the functions of the faulty unit taken over by another unit. This will have increasing importance as human life will be at stake as control systems of the future becomes more prevalent in modern society.

3. *Communications*

The intelligent sensor must communicate with the central computer and other intelligent sensors over a high speed real-time communication network, to download run-time software, and transmit and receive local state tables. The communications network must also meet the deterministic criteria of real-time systems. The system designer must be able to reliably predict packet arrivals to ensure timing constraints placed on the system. One of the most common choices for real-time networks, discussed in later sections, is the Token-Ring network.

4. *Global Awareness*

A key item in many systems is data. The data must flow unobstructed throughout the system so that it may be used to form decisions on which actions are to be taken. To accomplish this a mechanism must be established to store data at a central site or make these visible to all or most of the units in some other way. At the central site the data is time-stamped and stored in a global database. The intelligent sensor of the system can then store measured data taken. The sensor can also request data from the central computer that has been stored by intelligent sensors elsewhere in the system. If the central storage/repository is not practical or cannot be set for some reason, data stored at different places should be accessed using cache coherent technologies.

1.2. EXAMPLES OF INTELLIGENT SENSORS

The term "intelligent" is widely used in modern technology. Its use in the phrase "intelligent sensors" is intended to imply machine intelligence used by the devices which interface with the outside world. In this case it is a computational process that occurs in the electronics of a device used to collect information about the environment or to cause an event in the environment.

The circuitry of the intelligent sensor contains any number of the devices used to measure the quantity of a particular environmental characteristic. (See Appendix A) Signals of this type may cause a robotic actuator to move, or turn ON or OFF other devices that cause a physical change in the environment. Other components the intelligent sensor may contain would be RAM, ROM, and a microcomputer or micro controller. It may contain any number of other highly specialized processors such as integer processors, floating point processors, graphics processors, or image processors.

In this system a high-speed real-time communication network would connect all of the intelligent sensors with the main computer. This network would give the intelligent sensor the capability to receive commands to execute, programming instructions, or data from other intelligent sensors or the central computer. The intelligent sensor could also relay everything it receives to a neighboring sensor.

The intelligent sensors may consist of a number of specialized chips. Today there exist hundreds of types of microprocessor and micro controller chips. There are also many support chips for interfacing to computer networks, high speed numeric processing, and controllers for specialized devices. The processor used can range from small 8-bit processors to the fastest 64-bit processor with multiple arrays of specialized processors.

With the advances in today's micro electronics and data processing technology it is possible to miniaturize microprocessors along with a variety of sensors onto a single chip. Today there is a growing number of manufacturers experimenting with miniaturization of sensing devices and actuators, and more will be needed in the future.

The processor of the intelligent sensor may have a large number of roles to satisfy. One way to accomplish this task is to use a number of CPUs for each function, as illustrated in the following examples.

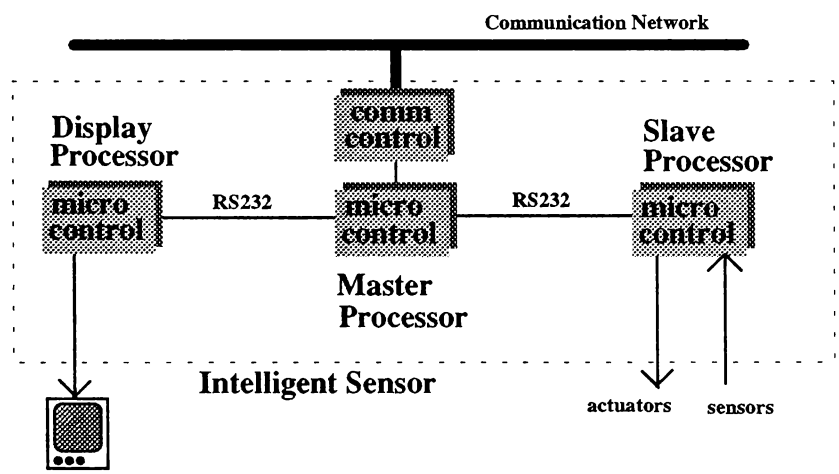

Figure 2. Example of an Intelligent Sensor using a micro controller.

Example 1. In the example in Figure 2 the master micro controller receives and transmits messages over the network. The slave micro controller is responsible for the data measurements and control. The third micro controller may be connected to a display that allows the user to view the status of the system. •••

There are many 8-bit micro controllers on the market today. The manufacturers of these chips have been adding a multitude of features to fill almost every applications need. In addition to RAM and ROM, some EEPROM (Electrically Erasable Programmable Read Only Memory) have also been recently added. Micro controllers with timers, counters, serial ports, parallel ports, interrupt controllers, DMA (Direct Memory Access) controllers, PW (Pulse Width) Modulators, ADCs (Analog-to-Digital Converters), DACs (Digital-to-Analog Converter), and interfaces to communication networks are all available today.

Example 2. For Intel 8051 micro controllers, it is possible to connect the internal processors of an intelligent sensor together through their parallel ports for one-way communications without having to provide buffers (see Figure 3a). This provides a high-speed method of transmitting micro controller data within an intelligent sensor. If bi-directional communication is desired between micro controllers a single 8255 PPI (Programmable Peripheral Interface) can be utilized (see Figure 3b). If there is a significant distance between micro controllers two 8255 PPIs may be required (see Figure 3c) to ensure proper transmission.

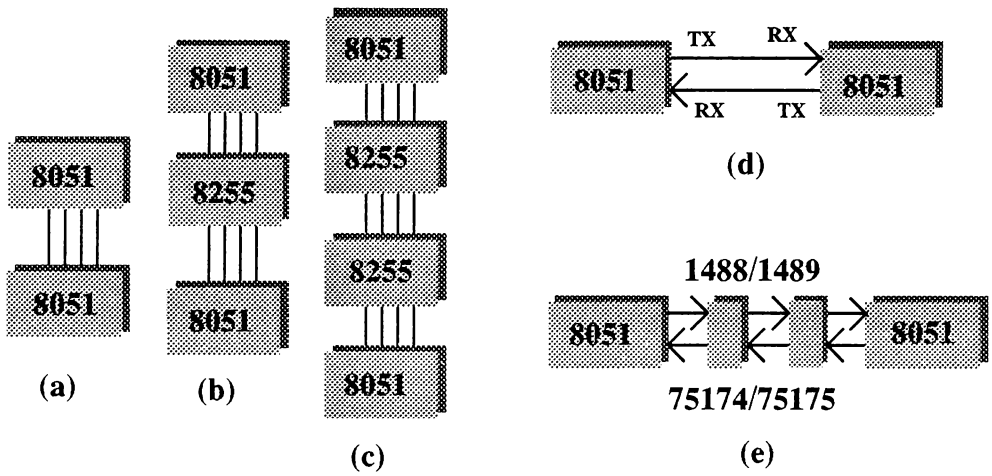

Figure 3. Inter processor Communication

For serial communications it is possible to connect micro controllers directly for short distances (see Figure 3d). If longer distance are required adding RS232 or RS422/485 buffers, as shown in Figure 3e, are appropriate. •••

Example 3. Using the UART (Universal Asynchronous Receiver/Transmitter) of the 8051 loops can be set up that echo's incoming messages to all of the processors, as shown in Figure 4a. This approach is similar to that of the Token-Ring. Each processor around the circle would echo the data to the next. This type of addressing, called SDLC (Synchronous Data Link Control), is built into the Intel 8044 micro controller. Other manufactures have done this with their chips as well, Signetics has implemented the I^2C data communications controller into some of the Signetics 8051 micro controller chips.

The system designer must use caution in designing these simple networks to avoid having to build sophisticated protocols to handle ACK's (Acknowledgment), NAK's (Negative Acknowledgment) and all of the overhead of handshaking protocols. Extensive communication activity would also take cycles away from their intended control

purposes. •••

Example 4. Another configuration of processors in a sensor would be the tree configuration as shown in Figure 4b. In this scheme there is a single master processor receiving message from the communication network and passing the messages to the individual processors. The way in which the processors are addressed is through the use of a ninth bit. When the ninth bit is set, it commands all of the processors to "wake-up" for the next byte, which contains the address. The processor that corresponds to the address sent will then start to receive the incoming message addressed to it, all of the others would ignore it. •••

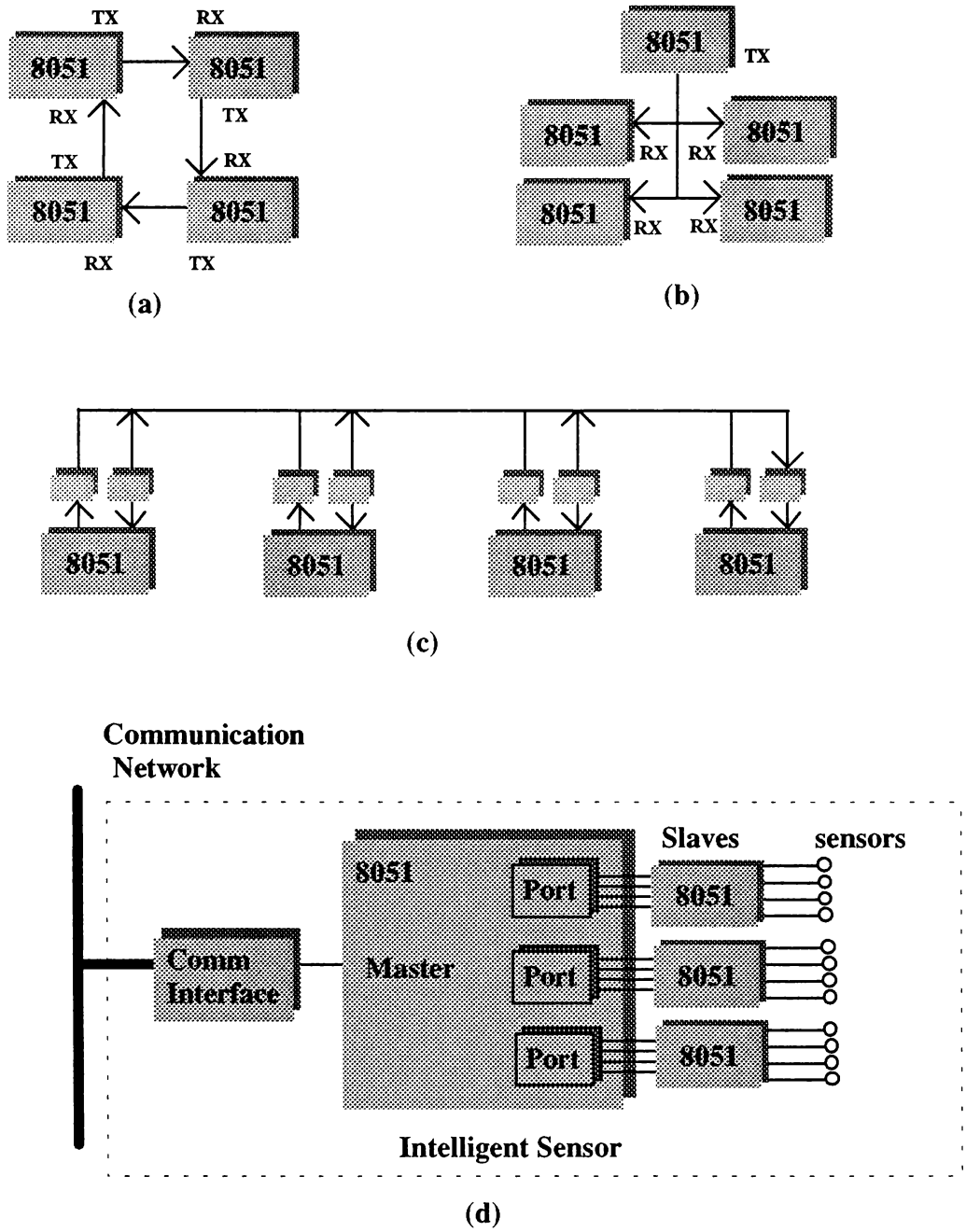

Figure 4. Intelligent Sensor Connections

Example 5. The micro controller in an intelligent sensor may also be configured to communicate with a master micro controller through hardware ports, as shown in Figure 4d. By collecting sensor information and processing it the resulting data may be

collected by the master micro controller for further use. This would effectively "hide" the complexity of the design by modularizing it into smaller tasks and distributing it to a number of smaller controllers. An example of this type of design may be seen in a tracking system that displays the x, y, and z axis position of an object. Each slave controller would be responsible for a single axis in the coordinate system. The data would be gathered by each controller and analyzed concurrently. After the calculations are made and the position determined the information is made available at each of the three ports. The central controller would record the information at the three ports as the x-y-z position. •••

1.3. SUMMARY

It is clear that most of the presented configurations do not meet all the four criteria for functionality, 1) Preprocessing, 2) Fault-tolerance, 3) Communications, and 4) Global Awareness. All of these design criteria may not be well suited, or even desirable, for some types of problems where the control mechanism is very simplistic. However, for intelligent sensors, all of them should be taken into account. It is therefore necessary to take a closer look at the functionality of intelligent sensors.

2.0. SYSTEM DESIGN WITH INTELLIGENT SENSORS

2.1. DATA FUSION

Data fusion involves taking the data collected by the sensors and performing analysis on the data to extract more information than what would be obtained by analyzing the data of a single sensor. Data fusion can be done by one sensor combining various data measurements about the same object. The data is "*fused*" by one of two fusion techniques: (1) probabilistic models by using robust statistics or Bayesian reasoning, or (2) least-square techniques, such as the Kalman filtering.

2.1.1. PROBABILISTIC DATA FUSION

Statistical estimation is based only in part upon previously measured data. There are many assumptions that must be made concerning the data about randomness, probability distribution and many other, sometimes unknown, parameters. The statistical procedures used for probabilistic data fusion must be "robust". A robust procedure signifies an insensitivity to small deviations in the assumptions. Larger deviations from the assumptions should not be catastrophic.

2.1.2. THE LEAST-SQUARE METHOD

A very important branch of inferential statistics, regression analysis, is used to compare previously collected data to discover any relationships that may exist. One of the most common procedure is the *Least-Square Method.*

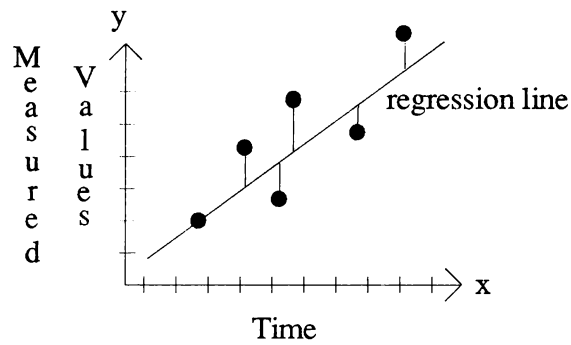

Figure 5. Direct Linear Regression Line

This method starts by creating a scatter diagram of the measurements taken. Once the diagram is established a line is computed that best "*fits*" the data as shown in Figure 10. The *regression line*, or *regression curve*, can have many shapes and characteristics depending upon the relationship of the data involved. In this example the relationship is assumed to be linear for simplicity. This type of diagram only gives a rough idea of the kind of relationship, if any, between any of the measurements. To get a more accurate of regression requires some calculation. In the linear example the equation of the line is written as

$$y = mx + b,$$

where b is the y-value at which the line intersects the vertical axis and m is the slope of the line. For each value of x the corresponding y value differs from the value it would have if the data point were exactly on the line. These differences are shown in the figure by the vertical lines. Choosing another line would probably make some of these differences greater and some smaller. The most common procedure to choose the line for which the sum of the squares of all of these differences is at the minimum, thus the name of this procedure, the method of the least squares. The method of finding the *least squares curve*

or *least square line* involves the use of optimization techniques. The variable y' is used instead of y to distinguish the predicted values from the y-values of the pre-measured data points. The least square line $y' = mx + b$ that provides the best fit to the data points (x_1, y_1), (x_2, y_2), (x_n, y_n) has a slope of

$$m = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2},$$

and the y'-intercept is

$$b = \frac{\Sigma y - m(\Sigma x)}{n},$$

Once an equation for the least square line has been found it is important to determine how well any future estimates will fit the points already observed. This can be done by calculating the *coefficient of correlation* by using the following formula,

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{n(\Sigma x^2) - (\Sigma x)^2} \cdot \sqrt{n(\Sigma x^2) - (\Sigma x)^2}}$$

The value of r is always between 1 and -1. Values that are found to be exactly 1 or -1 indicate the data points lay exactly on the least squares line. If r = 1 then the least square line has a positive slope; if r = -1 the least square line has a negative slope. If r = 0, then there is not linear relation between the data points. It could be a curve that may better represent the relationship between the data points rather than a linear construct.

2.1.3. KALMAN FILTERING

A Kalman filter is a recursive algorithm that uses all information that can be provided, regardless of precision, to estimate the current state of a system. This includes data concerning knowledge of the system under test, the dynamics of the devices performing the measurements, the description of system noises, measurement errors, and any information available about the initial state of any of the variables of interest. A recursive algorithm in this sense refers to the Kalman Filter algorithm ability to recompute the probability density given a new measurement. This eliminates the need to store a large amount of previously recorded data for the probability density to be recalculated

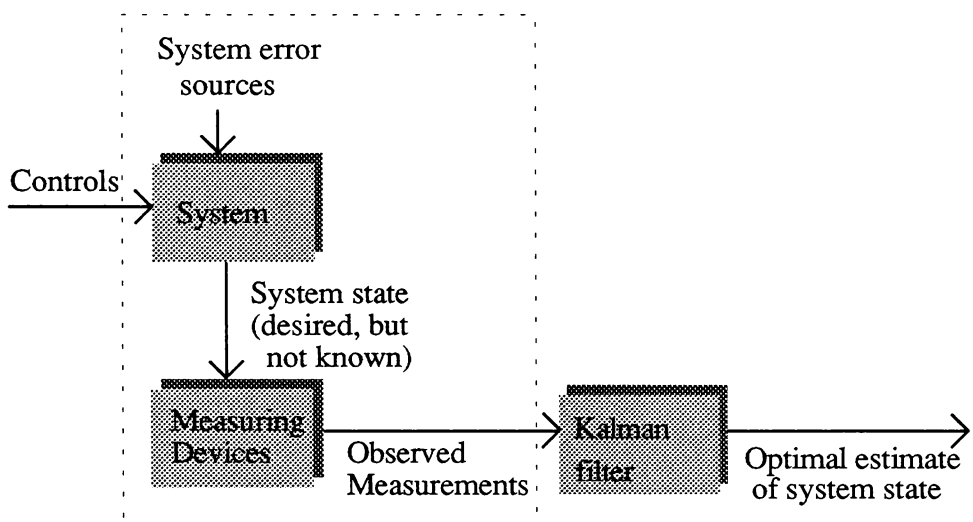

Figure 6. Kalman Filter Overview.

In this case the "*filter*" is simply a computer algorithm. The algorithm, as shown in the above figure, tries to obtain an optimal estimate of the environment. The algorithm does this by computing the *conditional probability density* of the desired quantities based upon the actual data coming from the measuring devices. Once the conditional

probabilities are computed the "*optimal*" estimate can be defined. Possible choices may include the *mean* - the "*center of the probability mass*" estimate; the *mode* - the value of x that has the highest probability, located at the peak of the density; and the *median* - the value of x so that half of the probability weight lies to the left of x and the other half to the right.

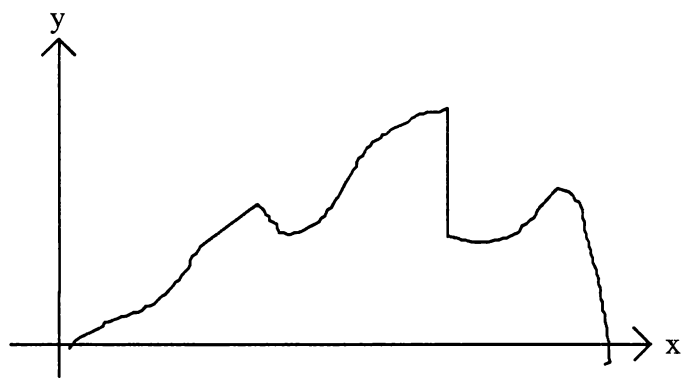

Figure 7. Example Conditional Probability Density

The Kalman filter performs this conditional probability density for problems in which the system can be described in a linear model, and where the noise measurements are classified as being white noise, that is, have no correlation with time.

2.1.4 SUMMARY

The review of data fusion techniques reveals problems with using intelligent sensors. Kalman filtering is used to predict behavior of objects in a dynamic systems. In order to accurately predict the behavior the designer must understand how all the environmental variables can affect the characteristic of the object under study. If all variables are not accounted for the predictive mechanism of the Kalman filter, it will be

unusable. There is a great need to understand how to pick these variables so the Kalman filtering accuracy can be predetermined and verified.

When working with microcontrollers used in embedded systems such as intelligent sensors there is a limit to the amount of resources available. The amount of RAM, clock speeds, and processor ability places a limit on the sophistication of algorithms used. As a result the algorithms must not only be accurate, but must be compact enough to fit into smaller memories, and have fast execution times to work on smaller CPU's. Therefore the efficiency, as well as the accuracy, of the Kalman filters must be studied to determine which should be selected dependent upon processor memory size and application.

2.2. FAULT TOLERANCE

2.2.1 CONFIGURATION FLEXIBILITY

By allowing the software of the intelligent sensors to be stored remotely and loaded via the network, the system has the capability of altering its configuration. This adaptability is non-existent in the classical form of the real-time system.

The design of a system utilizing intelligent sensors fits comfortably within the philosophy of the object-oriented design approach. The intelligent sensor acts as an "information hiding" object where the problems of timing constraints and those of task switching would become invisible outside. Any change to the environmental interface requirements of this part of the system would only affect the design of the intelligent sensor, and not the entire system.

To start this process, systems interactions with the environment are the first requirements to be identified. Once all of the interactions are known they are divided into system sub-objects based on their timing requirements and common information requirements. The requirements of these system sub-objects would become the requirements of the intelligent sensors. In this fashion a large complex system can be divided into a set of smaller sub-objects, each handled by its respective intelligent sensor.

The problems with the object-oriented design methodology are two-fold. Due to the concentration on the smaller sub-objects, the system designer has difficulty in getting an overall view of the entire system. The identification of the system timing constraints and subsequent classification of the system sub-objects has been described more as an art than a science.

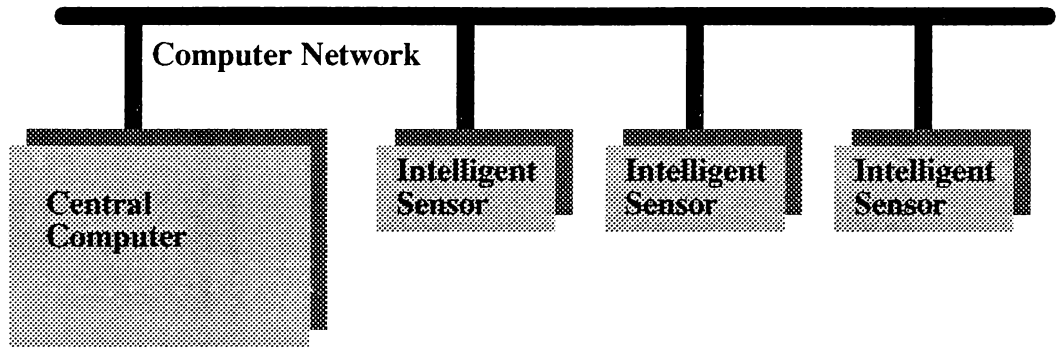

Figure 8. Inline Sensor Configuration

The inline intelligent sensor configuration allows all the intelligent sensors to communicate with the central real-time computer (Figure 8). As shown in Figure 9, each

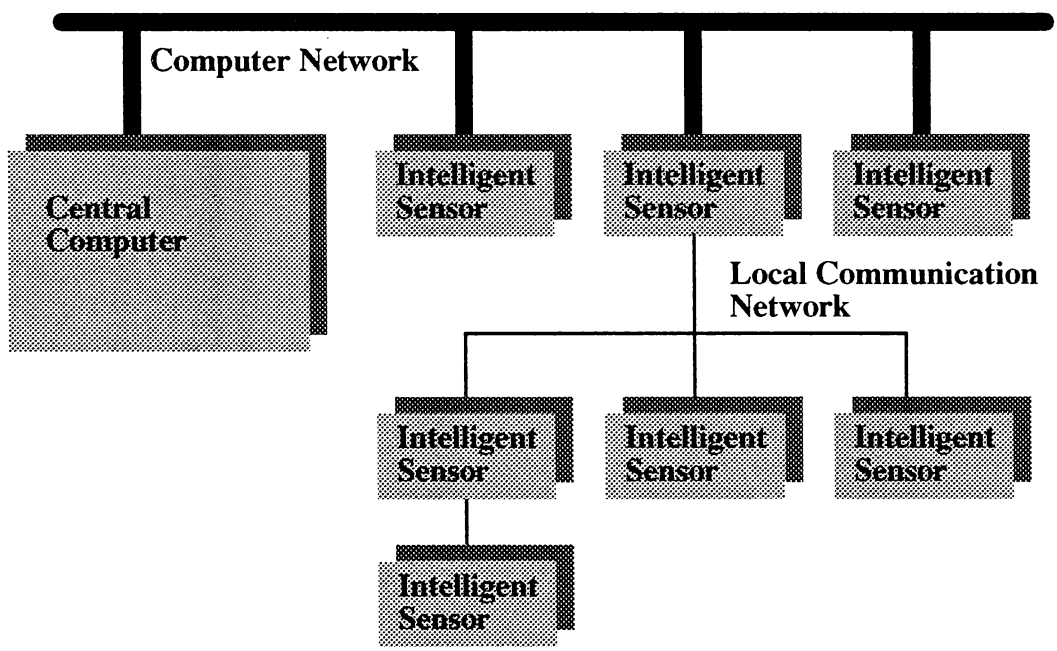

Figure 9. Sensor Tree Configuration

intelligent sensor can have any number of smaller sensors, as described in the Sensor Classification Table (see Appendix A), that they control.

In the tree type of configuration only one intelligent sensor is in communication with the central real-time computer. This intelligent sensor would be "relaying" the information (if needed) from all of its subordinate intelligent sensors. All of the sun-ordinate sensors would be carrying out their assigned task using whatever sensor/actuating interface to which they are attached to.

An intelligent sensor configuration in large is determined by the application. Some of the design characteristics the sensors have in common would be a channel of communication. By using this communication channel the intelligent sensor is capable of receiving new programming instructions and/or commands that would allow remote re-programming of the entire system. This ability is of vital importance where the remote sensors are in a very inhospitable environment, such as in a nuclear reactor, at the bottom of the ocean, on the moon, or even on another planet. The intelligent sensor would also be capable of transmitting processed data, in the form of state tables, over the communication channel to the main processor, and other sensors, as needed.

2.2.2. SAFER SYSTEMS

With the classical real-time design, a failure at the central CPU site or one of the major sensors would have catastrophic results. In this case there is nothing the system designer can do to prevent unusual behavior of the system as it loses control. In systems designed with intelligent sensors, as long as power is applied, the sensor would be able to sense the catastrophic failure when it loses communications with the central computer. It would then start a fail-safe shutdown of the parts of the system under its immediate

control.

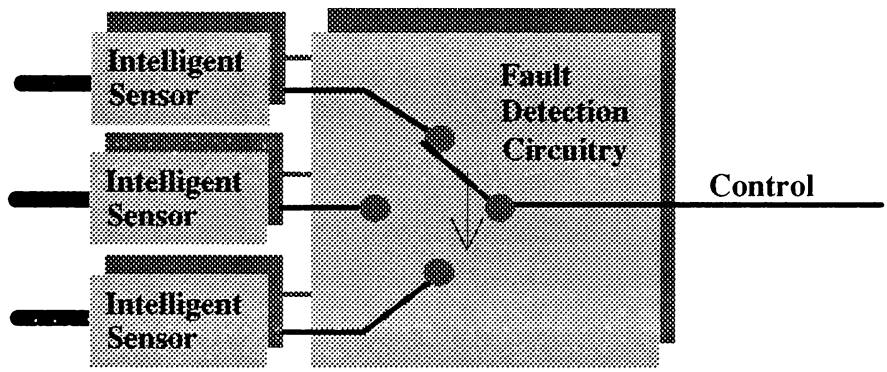

Figure 10. Hot Standby Configuration

Designing real-time systems with intelligent sensors would allow a safer system. By moving the workload to smaller and cheaper processors at the sensor level it is easier to design multiply redundant systems where faults can be detected and appropriate diagnostics used to prevent system failure. In Figures 10 and 11 two different types of fault recovery by use of two types of redundancy can be seen.

The first form of redundancy is the "hot standby", Figure 10. In this configuration a sensor has a number duplicates at a single site. If a fault is detected on one sensor, another sensor can be electronically "switched" to take its place. The control of the electronic switch can be a fault detection circuit wired to the front end of the sensor.

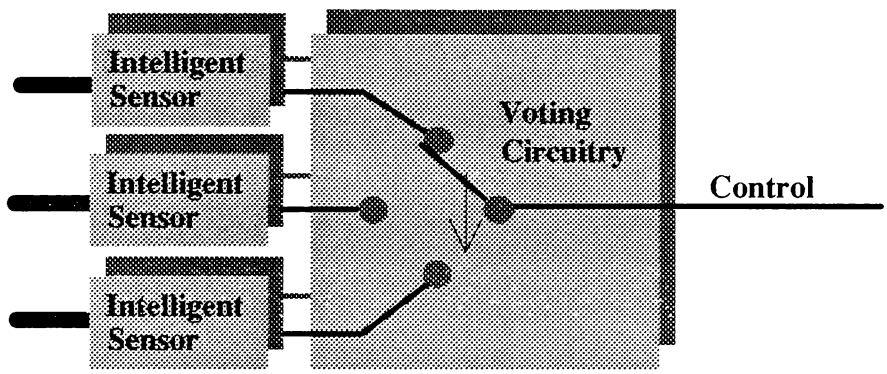

Figure 11. Multiply Active Redundant Configuration

Another form of redundancy is the "voting" sensor, Figure 11. In this case multiple sensors are again found at a site, but all of the sensors are active and operating. When data is collected or an action is needed the sensors all "vote" to see if all of the other sensors come to the same agreement. This will reduce the odds that one of the sensors is acting erratically.

Diagnostics are an important topic when designing real-time systems. The system must be able to detect problems and alert human operators, if necessary, in order to handle them appropriately. In particular, the following checks can be made to assure system safety.

(1) Analog-to-digital and digital-to-analog devices can be checked by applying known voltage to them and in turn read back the data to verify they are working properly. In addition to the diagnostics, the designer must design in fail safes to the converters. This is dependent upon the application whether it is safe to fail at the maximum value, the

minimum value, or some other predetermined value.

(2) Sensors can be checked to insure they are within proper range and that the rates of change of the values appear within normal accepted limits.

(3) Hardware options can be checked to insure they are working on powerup. Memory tests can produce checksum values to insure ROM/RAM/EEPROM are valid.

(4) Watch dog timers can be used to check for proper operation. These devices must periodically be reset by the running processor. If it is not reset within a specified amount of time it would represent a failure of the sensor. In the case of a failure a redundant unit can be electronically switched on, and a warning message sent to the central computer to the operator that a hardware failure has been detected and compensated.

(5) Voltage monitors may be used to continually check the voltage supplies of the sensor. It is also preferable to design the control circuitry with power supplies independent from the power used by the rest of the sensor. Safe outputs can then be generated even when the rest of the sensor has lost power.

The intelligent sensor must be able to report the existence and type of failure to the human interface and the central system and, if necessary, shut down itself in an orderly, fail-safe manner.

2.2.3. SUMMARY

Fault-tolerance properties of intelligent sensors need further attention. When an

intelligent sensor does fail a decision must be made, based on the current state of the entire system, to determine what kind of action must result. If an intelligent sensor fails at a critical moment during system operation far more drastic action may be taken than when the intelligent sensor fails in a non-critical mode. In order to implement fault-tolerance in this type of system this issue needs to be fully understood.

The implementation of fault-tolerance can add significant cost to building the system. Such systems can be doubled, and even triply redundant, causing the cost of the hardware to increase with each step in redundancy. Cheaper forms of fault tolerance need to be investigated to make fault-tolerance more financial appealing.

In the design of real-time systems intelligent sensors would play a variety of critical and non-critical roles of control. Clearly not all of these areas need utilize fault-tolerant design methods. The failure of a non-critical sensor may be of little consequence to the operation of the system as a whole. One problem needing much further research is the criteria to decide which intelligent sensors will have fault-tolerance, and which will not.

2.3. REAL-TIME COMMUNICATIONS

The most important criterion for the selection of the communication protocol for a real time system is that it must be deterministic in a sense that it guarantees delivery time. The designer must be able to determine the time it takes from an instant when an initial event occurs, the data is collected, until the final event occurs in the sequence.

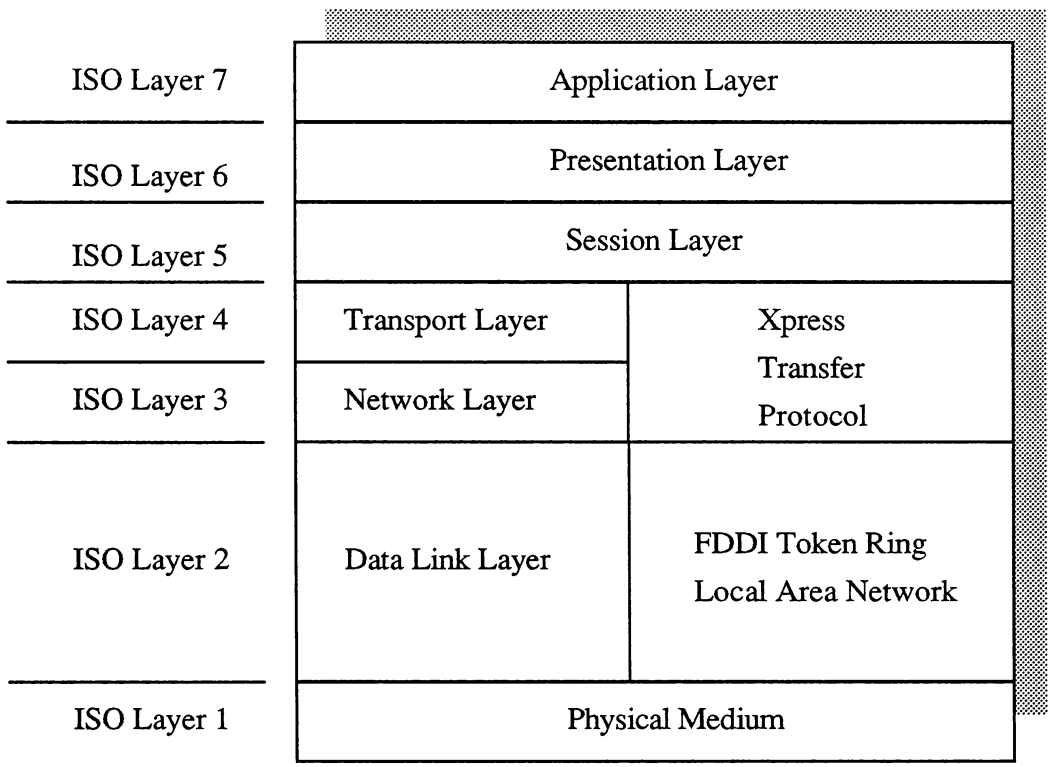

Figure 12. The Seven-Layer ISO Standard Protocol

The common model of considering non-real-time systems is the seven layer ISO Standard. In order to meet the criteria for high-speed real-time systems developers use the Token Ring for the Data Link Layer, see Figure 13. The Token Ring network passes a data packet, or token around a circular network. When a device attached to the network

see its address in the token it grabs it, and places another token onto the network. The delivery time of the token in this type network can therefore be calculated, making it ideal for time constrained operations. FDDI (Fiber Distributed Data Interface) is a well established method of implementing token ring architecture for Token Ring. At ISO Level 3 and 4 is the Xpress Transfer Protocol. This protocol was developed to improve network efficiency for pulling the bits from the network at high speed. In a fiber optic token ring network the packets have the ability to travel at the speed of light. Because of this, the packets must be "slowed" down in order for each device to electronically capture the network data and for the device software to interpret the information to see to which device the packet has been addressed. XTP speeds this process by implementing the protocol entirely in hardware to allow the networks to run at very high speeds.

One of the few protocols that is capable of deterministic behavior is that of the Token Ring networks. The token ring network has been very well defined in the standard *IEEE Standard 802.5: Token-Passing Ring Access Method and Layer Specification.*

2.3.1. PHYSICAL LAYER

The lowest level of the communication protocol refers to the medium by which the signal is sent from one location to be received by another. The three most reliable means of communications for real-time systems are twisted pair, coax cable, and fiber optic cable. Each has contrasting advantages and disadvantages that must be weighed during the selection process in every project.

Twisted pair is by far the lowest cost medium to build a network. It consists of wires wrapped in a shield to protect it from outside electrical interference, and a tough

plastic coating. They are very easily installed and maintained. One of the major disadvantages of this medium is that it is very poor at transmitting information over large distances. Also in some production floors where there is a great amount of electrical interference from large transformers this medium is completely unusable.

Coax cable is somewhat more difficult to install than twisted pair, and is more expensive. But it does not suffer as much from the electrical interference problem. Transmitting for greater distances is also an advantage.

The state of fiber optic has increased rapidly as its use has steadily increased. Its immunity to electrical interference, tolerance of high temperature, and other extremes in its environment, its ability to transfer enormous quantities of data, and its compactness has made it a very attractive choice despite its high cost. Fiber optic cable that is used today can transmit data at 3.4 Gigabits per second.

On the physical layer, the primary logical unit of information transmitted is a sequence of bits. The most common protocol for such transmission is RS232. This protocol is an interface between DTE (Data Terminal Equipment) and (DCE) Data Communications Equipment. The EIA (Electronic Industries Association) document defines the mechanical description of this standard as a 25 pin female connector for the DCE, and a male connector for the DTE. The twenty-five "interchange circuits" are given in the Table 1.

PIN	CIRCUIT	NAME
1	AA	Protective Ground
2	BA	Transmitted Data
3	BB	Received Data
4	CA	Request to Send
5	CB	Clear to Send
6	CC	Data Set Ready
7	AB	Signal Ground or Common
8	CF	Received Line Signal Detect
9	--	Reserved for testing
10	--	Reserved for testing
11	--	Unassigned
12	SCF	Secondary Received Line Signal Detect
13	SCB	Secondary Clear to Send
14	SBA	Secondary Transmitted Data
15	DB	Transmission Signal Element Timing
16	SBB	Secondary Received Data
17	DD	Receiver Signal Element Timing
18	--	Unassigned
19	SCA	Secondary Request to Send
20	CD	Data Terminal Ready
21	CG	Signal Quality Detector
22	CE	Ring Indicator
23	CH/CI	Data Signal Element Timing
24	DA	Unassigned
25	--	

Table 1. Pin assignments for RS232[2]

2.3.2 DATA LINK LAYER

One of the few protocols that is capable of deterministic behavior is that of the Token Ring networks. The token ring network has been very well defined in the standard *IEEE Standard 802.5 Token-Passing Ring Access Method and Layer Specification.* It has also been one of the most studied networks and will not be discussed here. According

to Kang Shin and Chao-Ju Hou the best token ring derivation for real-time application is the token ring scheduling protocol.[11]

FDDI -- THE FIBER DISTRIBUTED DATA INTERFACE

The FDDI Token Ring Protocol is a high performance interface that runs at 100 Mega bits per second over distances up to 200 kilometers. The fiber optic protocols are still evolving as researchers continue to improve its speed and reliability. At AT&T's Bell Labs scientists hope to soon accomplish speeds in the range of tens of gigabits-per-second.[17]

IEEE P1394 SERIAL BUS

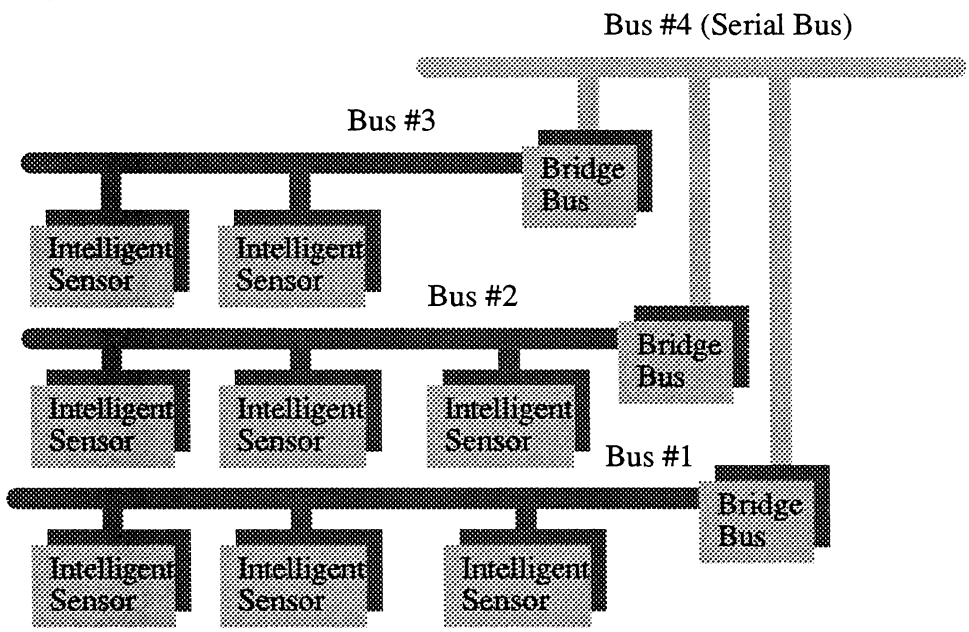

Figure 13. Example of Serial Bus Hierarchy

The High Performance Serial Bus in Figure 13 can be used as an alternative to interconnection between intelligent sensors and other devices. This specification allows speeds from 50 megabits per second to 100 megabits per second for distance of 25 meters.

The American National Standards Institute is still developing this standard, which looks like a very promising protocol for high-speed communication applications.

FIBER CHANNEL

The Fiber Channel is a serial I/O channel specification that boosts data communication to speeds from 100 Megabits per second to over 1 Gigabit per second for distances of four to ten kilometers. This protocol can be used over fiber optics or copper cable. Hewlett-Packard (HP) and International Business Machines (IBM) have joined to produce a simple chip set that implements this protocol in hardware. Since its acceptance by the American National Standards Institute (ANSI), it has been growing in popularity.

HIPPI - HIGH PERFORMANCE PARALLEL INTERFACE

The Los Alamos National Laboratories (LANL) is one of the very few research labs that needs a network with an 800 megabit per second bandwidth. With the backing of several industrial partners, the LANL was able to put pressure on the American National Standards Institute (ANSI) to adopt the HIPPI as standard X3T.3/88-023. The High Performance Parallel Interface is a point-to-point high-speed channel that can deliver 800 megabit per second on a shielded twisted pair cable at a maximum distance of 25 meters. It is also capable of 1,600 megabits per second over two cables.

Applied Micro Circuits Corporation has introduced S2020 and S2021 chip set as the source and destination for the HIPPI standard. The 32-bit chips use ECL chip technology to meet all of the ANSI requirements to provide the HIPPI signaling protocol. These chips are currently available for $195 in lots of 100.

2.3.3. XTP -- THE XPRESS TRANSFER PROTOCOL

The XTP protocol is being developed as a reliable, real-time, transfer protocol for use with the next generation of high speed networks such as the FDDI. XTP is can be implemented as part of the International Standardization Organization's (OSI) Open System Interconnection Reference Model, where it takes over the Transport and Network Layer services.

The network layer and transport layer, shown in Figure 12, were designed in an era of relatively slow and unreliable networks. At their time they supplied state of the art characteristics such as error detection, re transmission, flow control and data re sequencing, but they are missing many of the newer communication concepts. They do not provide rate control, selective re transmission, or reliable multicasting. Their packets are complex and require extensive parsing because of variable header lengths. These protocols also require timers at both sender and receiver to implement data acknowledgments. With all of this overhead significant amount of time is required to parse the header, determine the message time and respond accordingly.

With the XTP protocol the processing time for incoming and outgoing messages is guaranteed to be no greater than the transmission time. XTP also contains error, multicasting, flow and rate control mechanisms. Due to the severe timing requirements XTP is implemented in VLSI hardware as the PE1000 chip set[14] (see Figure 14) that can be added to existing hardware to interface to the networks. The estimated throughput of the PE1000 chip set is 200 mega bits per second for chip clock rates of 25 Megahertz. Since the chip set was not developed to implement only the XTP protocol, it can also be used to accelerate the throughput of other protocols. The PE1000 is a highly parallelized VLSI set that allows several types of data packet processing to occur at once, such as

buffering, processing the addresses, and performing the checksum calculation. The chip consists of four different chips, the MAC port, Host Port, the Buffer Controller, and an optional Control Processor.

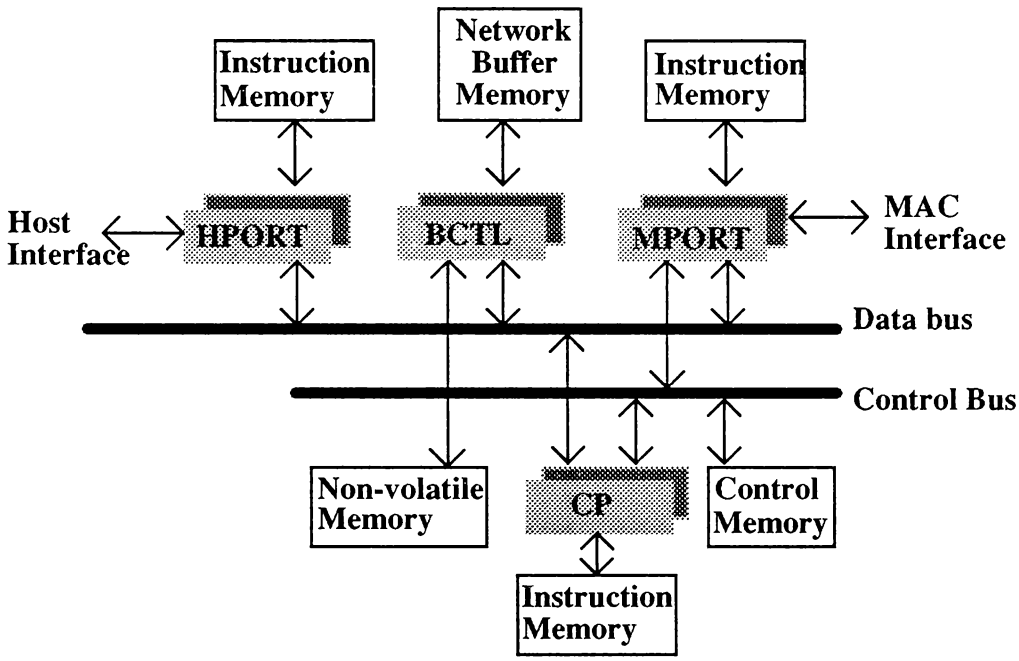

Figure 14. PE1000 System Overview

Flow control allows the receiver to tell the sender about the state of its receive queue. It tells the sender the number of packets it is able to receive so that the receive queue does not overflow. Therefore the sender can hold the packets until it knows that the data delivered has been processed. When the sender sees that the flow control has been increased it can then send more packets.

In situations when flow control is not enough the XTP protocol uses *rate control* to restrict the size and timing of the packets the sender transmits. This problem is independent of the flow control problem. The receiver may have adequate buffer space

available, but back-to-back packets may arrive faster than they can be processed. Together these two control algorithms allow the receiver to tune transmission rates to an acceptable level.

When errors occur in transmission XTP, like other protocols, must perform *error control* to detect the errors, and correct or request re transmission of the data packet. XTP uses two checksums that appear at the end of the data packets. The first checksum is calculated by using the exclusive-OR operation. This represents the vertical parity of the bits. The second checksum is the intermediate result shifted left before the XOR operation with the next word. The checksum is computed via hardware as the data packet arrives, therefore saving the time of re-scanning the entire packet for the computation. If the two checksums are invalid the packet is immediately discarded. In the mean time the sender is still sending packets. When these packets are received, they contain a sequence tag indicating their order. If any packets are missing the receiver *selectively requests* packets that were received with errors.

2.3.4. NTP -- THE NETWORK TIME PROTOCOL

One of the most important resources of a real time system is time. Events throughout the system must be synchronized with the occurrence of other events. Data must be cataloged and time stamped for timeliness and proper analysis.

Time is of equal importance in a system designed with intelligent sensors. Here the problem of the distribution of time through the system has been somewhat complicated by placing the data collection and analysis at the sensor level. Clearly all sensors must be synchronized in order to insure time precision and chronological correctness of the

collected data.

The problem of time in distributed systems has been well studied. In Lamport's paper[7] the distributed system has a master computer containing a clock, in our case the main computer, and all other computers, the sensors in our case, would periodically check a time stamp present in the time messages sent throughout the system. If the slave finds the time stamp is less than that of the masters, the slave reset its clock with the value of the time stamp message.

A well-defined practical time protocol is that of NTP (Network Time Protocol). NTP has been used to synchronize the hosts and gateways of Internet network. NTP is designed to distribute time information over large, heterogenous networks. It requires a number of time keepers that deliver continuous local time based on UTC (Universal Time Coordinates), even when leap seconds are inserted in the UTC timescale. The time keepers must have data smoothing and de-glitching algorithms to compensate for the delay times on the transmission paths. The time keepers must have very stable local clocks. Finally it must operate efficiently when it's time clients are continually polling. During its use on the Internet NTP has maintained time to within a few tens of milliseconds [9] even when there is a failure of clocks, time servers, or networks.

2.3.5. EXTERNAL DATA REPRESENTATION

A problem that plagues heterogeneous systems is that of data representation. Some systems use the "*big-endian*" representation where the most significant bytes of data are stored first, in lower memory, and the least-significant bytes are stored last, in higher

memory. Others use the "*little-endian*" representation which reverses the order, and some do not store them in contiguous memory. Each computer architecture may have its own unique manner of storing its data in memory.

For programmers of heterogeneous systems this can be a very formidable task to keep the data consistent throughout the system. If the programmer needs to establish communication between all of the different architecture's the programmer must construct

$$\frac{N^2 - N}{2}$$

versions of any programs that exchanges data. This is known as the *n-square conversion problem* [3]. Creating, testing, maintaining and managing N^2 version of a program would be a very difficult and time consuming task.

To eliminate the need for N^2 versions of a single program programmers need to write the programs in a manner that avoids data conversion for each architecture. Sun Microsystems has devised a method, called XDR (eXternal Data Representation)[16], which has become a standard through the industry. XDR is a set of routines written for each architecture that converts the data being sent into a common defined representation called the *external data representation.* This simplifies programming by requiring each program to convert data from only one representation. It also simplifies network administration because the packets sent over the network are in one format. The list of predefined data types are listed in Table 2. The only disadvantage to this method is that computers with similar architectures still must convert the data to be transmitted to the external data representation, and then convert them back when received. However the problems this solves are far more important than the computational overhead it causes.

Data Type	Size	Description
int	32 bits	32-bit signed binary integer
unsigned int	32 bits	32-bit unsigned binary integer
bool	32 bits	Boolean value (false,0 or true,1)
enum	arb	Enumeration type with values defined by integers(e.g.RED=1,WHITE=2,BLUE=3)
hyper	64 bits	64-bit signed binary integer
unsigned hyper	64 bits	64-bit unsigned binary integer
float	32 bits	Single precision floating point number
double	64 bits	Double precision floating point number
opaque	arb	Unconverted data (i.e., data in the sender's native representation)
string	arb	String of ASCII characters
fixed array	arb	A fixed-size array of any other data type
counted array	arb	Array in which the type has a fixed upper limit, but individual arrays may vary up to that size
structure	arb.	A data aggregate, like C's struc
discriminated union	arb	A data structure that allows one of several alternative forms, like C's union or Pascal's variant record
void	0	Used if no data is present where a data item is optional (e.g. in a structure)
symbolic constant	arb	A symbolic constant and associated value
optional data	arb	Allows zero or one occurrences of an item

Table 2. Sun Microsystems XDR Data Representation[3]

Each architecture would contain an XDR library that converts data being transmitted and received. The first call to the library would be *xdrmem_create* to create a memory buffer that holds the incoming and outgoing data. If the program wanted to transmit data the third parameter of the routine would be a *XDR_ENCODE,* if the program wanted to translate data coming in, the third parameter would be *XDR_DECODE.* After this routine is called the program is ready to convert the data

found in the buffer. The routines in Table 3 are then called in a pre-planned manner to properly decode/encode the data.

Procedure	parameters	data type converted
xdr_bool	xdrs, ptrbool	Boolean (int in C)
xdr_bytes	xdrs, ptrstr strsize, maxsize	Counted byte string
xdr_char	xdrs, ptrchar	Character
xdr_double	xdrs, ptrdouble	Double precision floating point
xdr_enum	xdrs, ptrint	Variable of enumerated data type (an int in C)
xdr_float	xdrs, ptrfloat	Single precision floating point
xdr_int	xdrs, ip	32-bit integer
xdr_long	xdrs, ptrlong	64-bit integer
xdr_opaque	xdr, ptrchar, count	Bytes sent without conversion
xdr_pointer	xdrs, ptrobj, objsize, xdrobj	A pointer (used in linked data structure like lists or trees)
xdr_short	xdrs, ptrshort	16-bit integer
xdr_string	xdrs, ptrstr, maxsize	A C string
xdr_u_char	xdrs, ptruchar	Unsigned 8-bit integer
xdr_u_int	xdrs, ptrint	Unsigned 32-bit integer
xdr_u_long	xdrs, ptrulong	Unsigned 64-bit integer
xdr_u_short	xdrs, ptrushort	Unsigned 16-bit integer
xdr_union	xdrs, ptrdiscrim, ptrunion, choicefcn, default	Discriminated union
xdr_vector	xdrs, ptrarray, size, elemsize, elemproc	Fixed length array
xdr_void	-none-	Not a conversion (used to denote empty part of a data structure).

Table 3. XDR Data Conversion Library [3]

2.3.6 SUMMARY

There are several problems in data communications among intelligent sensors. When large volumes of data is exchanged the process of converting all of the data into XDR representation will slow the communications greatly. Software data conversion may

not be the desired alternative.

As networks become overloading due to large data requirements the characteristics of the network may change. In token-ring networks, there may be a limit on the data throughput of the system. The designer must be aware of the physical limits of the network in order to accurately predict its worst-case behavior.

As communication networks become more sophisticated more complex and expensive hardware must be used to support the interfaces between the network and the devices on the network. There is some trade-off point where the type and speed of the network is outweigh by the tremendous cost of attaching large number of devices to the network. Practical guidelines need to be established to show the trade-off between network types, speed, number of devices, etc.

2.4. GLOBAL STATE AWARENESS

2.4.1. GLOBAL AND LOCAL STATE TABLES

The manner in which many distributed systems exchange information is through state tables. Each of the intelligent sensors may contain a piece of the overall state of the system. When required, through timing constraint, or explicit request, the intelligent sensors can send their local state tables to the central computer for storage and analysis. The intelligent sensor could also request new information from the central computer, so it may make decisions based on the big picture.

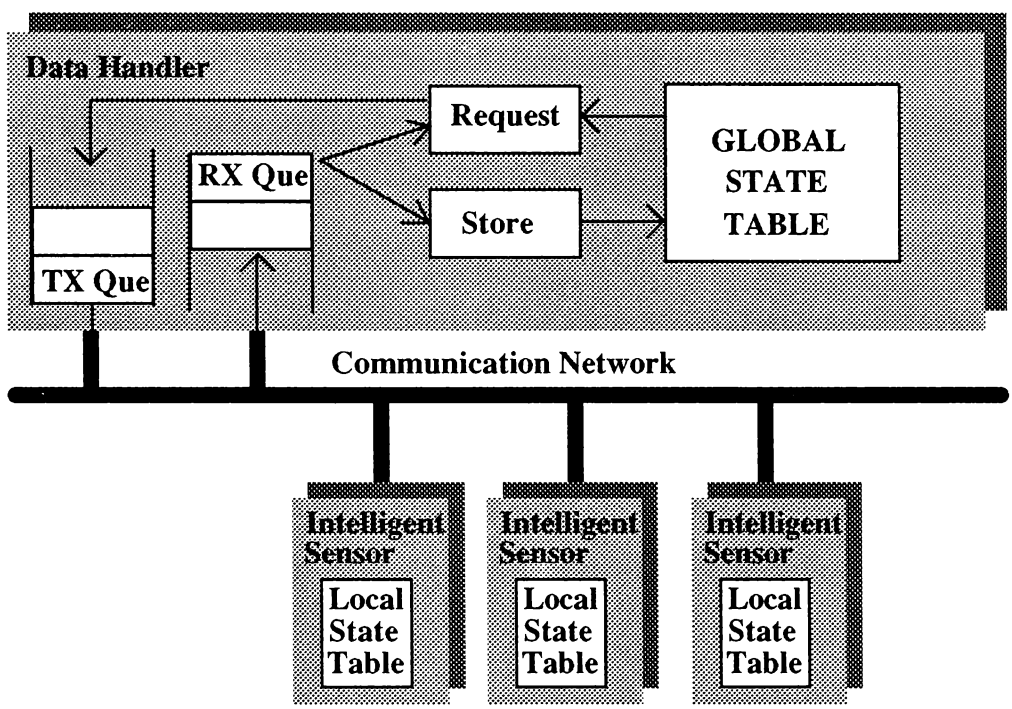

Figure 15. Global Awareness Using State Tables

In the example above there are intelligent sensors each with Local State Tables. The Local State Table can represent a small collection of measurements or a large

database depending upon the application. When the sensor receives a measurement from the outside environment, it performs data fusion and stores it in the Local State Table. At a predetermined point in time, or by request, the intelligent sensor then sends that information to the central computer where it is received into an RX Queue (receive queue). The RX Queue holds requests for data from all of the intelligent sensors and information that is being written to the global data base. There is also a TX Queue (transmit queue) that holds the requested data that is being sent to the intelligent sensor requesting the data.

2.4.2. SCI - SCALABLE COHERENT INTERFACE

SCI is a high speed point-to-point communication protocol between nodes of a network. A typical link is 2 bytes wide and allows for data to be transferred at speeds up to 1 gigabyte per second at a clock rate of 250 mega hertz. The nodes are arranged in a ring-topology. SCI uses a register insertion technique to place packets onto the ring. The register insertion technique allows for multiple nodes to be transmitting at the same time. SCI preemptive arbitration and queuing allows upto 256 levels of priority. This allows it to be compatible with many of the real-time scheduling algorithms, such as the Rate Monotonic Scheduling. This protocol is still under development and has not been fully implemented, but it looks like another viable solution to the problems of real-time communications.

2.4.3. SUMMARY

The primary problem is maintaining awareness of the global state is consistency of information. This includes the database integrity and cache coherence. As new communication forms become available the designer must be able to calculate the time from packet transmission to packet arrival in order to insure deterministic behavior.

2.5 LOCAL PROPERTIES OF THE INTELLIGENT SENSORS

The properties of an intelligent sensor can vary widely from application to application. The most simplistic form of an intelligent sensor may involve polling a number of data ports to gather information to be used in calculations and then sending the results periodically to the central computer.

On the other hand an intelligent sensor may be a classic example of a real-time system with a real-time kernel, task scheduling and timing constraints. An excellent example of this is the μCOS real-time kernel. This real-time kernel is especially designed for building small real-time systems and the source has been released to the public.

2.5.1 THE KERNEL

The main software component of an intelligent sensor is called the kernel or executive. It can be divided into two functional areas, task scheduling and data communication. The kernel is the area that has gotten the greatest attention of researchers. If an improvement of speed or efficiency is accomplished in any area of the kernel, large increases in application execution speed can result.

The classical approach to the design of these systems is to use one or more computers containing a number of sensors and actuators. The sensor measure changes in the environment. The system stores the collected data for analysis or uses the data to make a decision to trigger an event or task. The actuator is a software command to the hardware to change the environment. This may be as simple as an electronic signal sent to

a port that signals a device to open a valve or to move.

2.5.2. TASK SCHEDULING

The scheduling software must determine if there is already a task executing and if the new task has a higher priority. If a higher priority task needs to run, all of the information concerning the current task is saved so it can resume later. This is known as task pre-emption. As the new software task starts execution it may take other measurements or issue a command for a physical action to occur through the use of an actuator. When it has completed executing, the task that was pre-emptied, if any, is allowed to resume execution.

One of the most promising methods of scheduling is the Rate Monotonic algorithm. This algorithm refers to organizing the tasks is such a way that the tasks with the smallest periods have the highest priorities. A plot of the task priorities versus the task rates will then follow a monotonically ascending path.

2.5.3. TIME

Time is the mechanism used to synchronize tasks in the real-time system. It is utilized by the real-time designer to insure the appropriate tasks are executed with proper time bounds. It is also the principal criterion used to verify system correctness.

There are three types of timing requirements that are common to real-time systems. The latency time is the amount of time that elapses between the triggering of an interrupt, by an outside event, until the corresponding task is scheduled and is ready to start executing. This is also called responsiveness. The time at which the task is ready to

start execution is called the minimum critical time. The maximum critical time is the time at which the task ends execution. The amount of time between the triggering of the interrupt, until the end of execution of the appropriate task is called the time constraint.

If the timing requirements are not met the system must be able to degrade gracefully. This is accomplished by designing algorithms so that when the expected conditions are not met, they are handled in a logically predictable fashion. The important point remains that the design of the system must be able to meet the timing constraints.

Although these issues are important to real-time systems design, they will not be covered here because they are not specific for intelligent sensors.

3.0. PROBLEM FORMULATION AND PROCEDURE

The purpose of this work is to establish and verify an industrial oriented design methodology for developing real-time systems with intelligent sensors. Such methodology should use well established design methods supported by off-the-self software tools so designers can build and simulate systems which use intelligent sensors for real problems.

Designing real-time systems with intelligent sensors is feasible if the designer looks at the system in a hierarchical manner. Such an approach is emphasized in top-down methodologies. The intelligent sensors then become objects at the bottom of this hierarchy having the ability to exchange messages with any other nodes in the hierarchy, on the same or different levels. This type of approach is supported by the HOOD[5] (Hierarchical Object Oriented Design) methodology. It uses the concept of hierarchical objects to develop details of the system design.

In this work I am using the HOOD approach as the design concept vehicle. First the design is produced of a real-world problem, in this case an electricity generating power plant. The design is then verified using the industrial-level tool Teamwork/HOOD from Cadre. A portion of the power plant design, the boiler, is then chosen for implementation and its correctness verified using the boiler simulation provided by the Canadian National Research Council (NRC).

4.0. REAL-TIME SYSTEM EXAMPLE

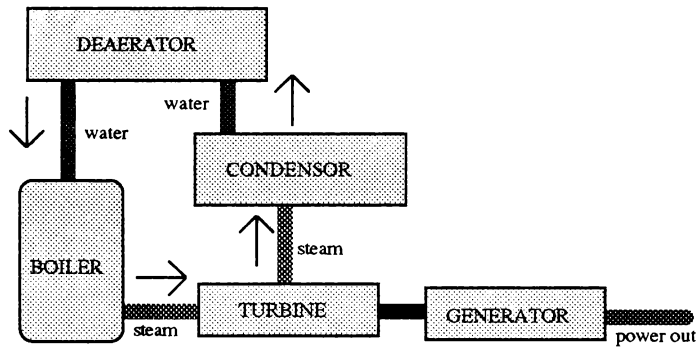

Figure 16. Conceptual Diagram of Power Plant

This is an example of a very complex real-time system. The control of a power plant may contain thousands of devices, that can affect the lives and well-being of millions of people. It is highly critical that proper control be maintained.

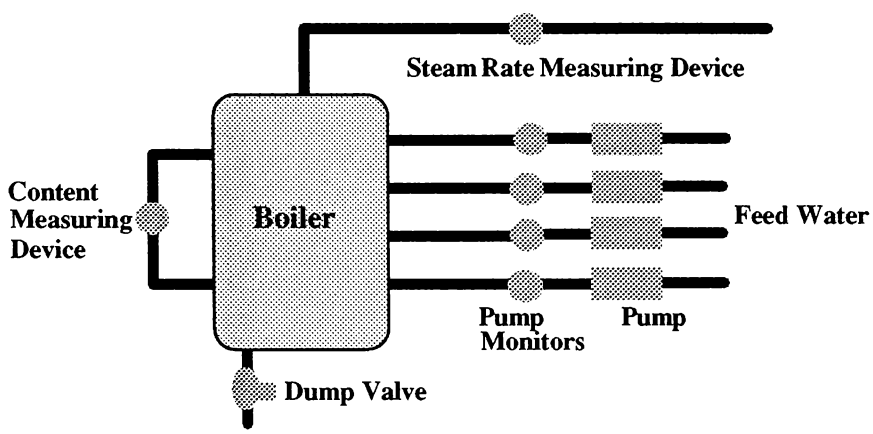

Figure 17. Boiler Configuration

4.1. BOILER

The boiler itself is an object which is "controlled" by an intelligent sensor (instrumentation system) of which the program in Appendix C is an implementation. The

boiler of the Steam Power Plant example is taken from the simulation created by the Canadian National Research Council. The specifications were defined as follows:

1) The Boiler Device

The boiler device is the central holding tank for the water. Its operating characteristic are as follows:

a. Total capacity: 130,000 pounds of water

b. Minimum amount of water for safe operation (all conditions of operation): 23,000 pounds.

c. Maximum amount of water for safe operation (all conditions of safe operation): 110,000 pounds.

d. Maximum achievable steaming rate: 700,000 pounds per hour.

e. Maximum achievable rate of increase of steaming rate: 4,200,000 pounds per hour per hour. (independent of boiler operating point)

f. Maximum achievable rate of decrease of steaming rate: 84,000,000 pounds per hour per hour. (independent of boiler operating point)

g. Relationship of water content requirements to boiler dynamics: the value at b. above has been chosen such that if the boiler content is just at the value, the boiler will not be damaged for another six seconds if feed water fails at maximum steaming rate; the value at c. above has been chosen such that if the boiler content is just at the value, the boiler will not be damaged if all feed pumps are operating for six seconds at full rated capacity with the steaming rate at zero.

2) The Pump Devices

The pump devices are used to pump water into the boiler tank. Their characteristics are as follows:

a. Type of operation: off/on

b. Rated water output: 275,000 pounds per hour (+ 0 %, - 5 %)

c. Running indication: motor on/off output provided

d. Start-up characteristics: 4 to 6 seconds for pump to develop enough pressure to overcome boiler pressure, full flow then developed essentially instantaneously

e. Shutdown characteristics: instantaneous

f. Feed water Temperature: 18 (+5, -10) degrees Centigrade is reference temperature and 5 MPa is reference pressure to be used for water weight-to-volume conversion.

g. Feed water Temperature Rise through Pump: Need not be considered

3. The Pump Monitor Device

The pump monitor devices are used to sense changes in the pump operations. Its characteristics are as follows:

a. Type of output indication: water feeding / water not feeding

b. Water feeding indication set point: water flow equal to or greater than 425 imp. gallons per minute.

c. Set point accuracy: (+ 1 %, - 2 %)

d. Water not feeding indication: water flow less then the set point specified above

e. Measurement lag time: negligible

4. Boiler Water Content Measuring Device

The boiler water content measuring device measured the water level inside of the boiler. Its characteristics are as follows:

a. Nominal calibration: 2.50 units per 1000 pounds of water

b. Range of calibration constant: 1.95 to 2.88 units per 1000 pounds of water

c. Accuracy when calibrated: (+ 0.5 %, - 0.7 %)

d. Range of valid outputs: 30.00 to 360.00 units (inclusive) with output resolution of 0.01

e. Calibration constant availability: on query to water content measuring device

f. Calibration constant variability: calibration constant does not vary (set, recorded, and locked by instrumentation technicians)

g. Compensation for water density and water column effects: integral to the device

h. Measurement lag time: negligible

5. The Steaming Rate Measurement Device

The steam rate measuring device measures the pressure of the steam coming from the boiler. Its characteristics are as follows:

a. Measurement range: 0 - 850,000 pounds per hour.

b. Accuracy: + 2,000 pounds per hour., -3,000 pounds per hour (but a negative steaming rate cannot be put out by a serviceable device)

c. Output Resolution: one unit per 500 pounds per hour

d. Measurement lag time: negligible

4.2 REMAINING COMPONENTS

The turbine contains several sensors to monitor its performance. The speed of the turbine, when engaged, must be maintained at a constant 3600 RPM to ensure 60 cycle electrical output. If the turbine varies within a preset level from this the turbine lockout is automatically tripped. Vibration sensors placed at several points on the turbine guarantee that the bearings are within specs. The vibration information is compared with vibration data of the past to determine wear of the bearing. Pressure and temperature sensors monitor the steam flowing into the turbine. Any variance in temperature and/or pressure has an effect on the RPM of the turbine. If the RPM falters, the temperature and/or the pressure of the incoming steam may be altered to compensate the RPM. Several safety features allow for the steam to be rerouted or cut off when the turbine needs to be stopped due to mechanical failure.

GENERATOR

The generator sensor system is similar to that of the turbine in many ways. The RPM of the generator must be maintained at 3600 RPM to guarantee a 60 cycle electric current. There are also many vibration sensors which maintain a history of vibration data so that wear of the bearings can be computed.

CONDENSER

The condenser has a series of electrical pumps that are used to pump fresh water, usually from a near-by lake, in order to cool the steam as it leaves the turbine. The condensation usually takes place in a tank called the hotwell. As water fills the hotwell it is pumped into the deaerator.

DEAERATOR

The deaerator is a series of tanks which remove the high levels of oxygen from the water. Some deaerators also serve as water filters as well. There are another series of pumps and valves that are controlled by this object that pumps the water back into the boiler after the sensors have determined the proper levels of oxygen in the water have been reached.

5.0. REAL-TIME SYSTEM DESIGN

HOOD methodology was used for the design of the real-time system. The main elements of HOOD (Hierarchical Object Oriented Design) are objects. HOOD has six different types of objects.

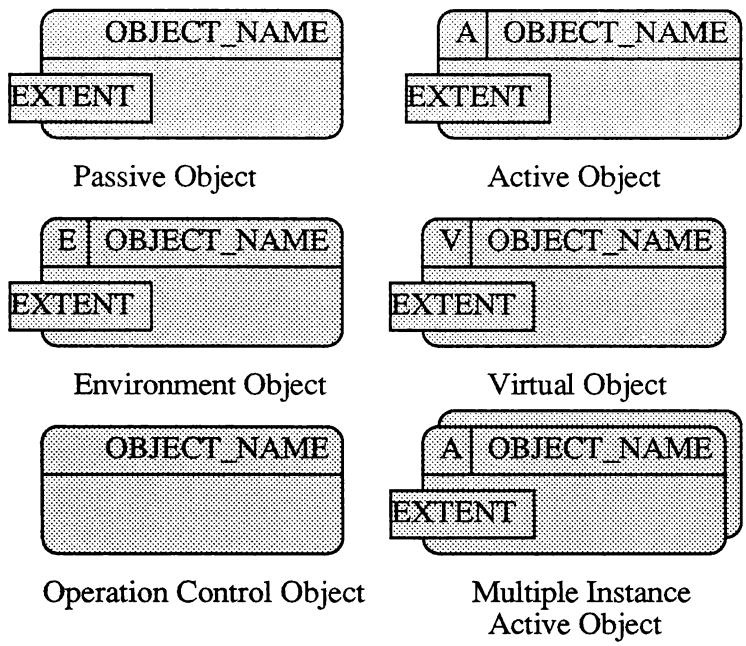

Figure 18. HOOD Objects

A *Passive Object* is an object to which execution control can be transferred immediately. This corresponds to a procedure invocation in a computer language. The Passive Object uses a " " in the upper left hand corner of the object icon.

An *Active Object* has a thread of control to execute but its execution depends upon the constraints. This corresponds to an Ada task and select and accept statements with the constraints determining which select code is executed. Active Objects can be identified by the "A" in the upper left hand corner of the object icon.

An *Environmental Object* is used to define an interface that is not present. This can be an interface to a console, keyboard or ports to the outside world. The Environmental Objects can be identified by the "E" in the upper left hand corner of the object icon.

A *Virtual Node Object* is used in distributed software design to indicate software that is designed to run on an external microprocessor. This object can be identified by the "V" in the upper left hand corner of the object icon.

A *Operation Control Object* is used to manipulate the state of an object. This object, like the passive object, has a " " in the upper left hand corner for an identifier.

HOOD, and its supporting toll, Teamwork/HOOD from Cadre, enforce quality assurance upon the designer, by providing output (i.e. code, reports, documents) in a standard way, by enforcing rules allowing only standard operations and naming conventions, by checking that rules are met, and by giving the designer a list of valid options to create an object.

The tool representation of a HOOD object is called a Hierarchy Structure Graph (HSG). As the designer enters each HSG it must go through a list of consistency checks before it is placed into the system database. Figure 19 is a representation of the "Check Model" window:

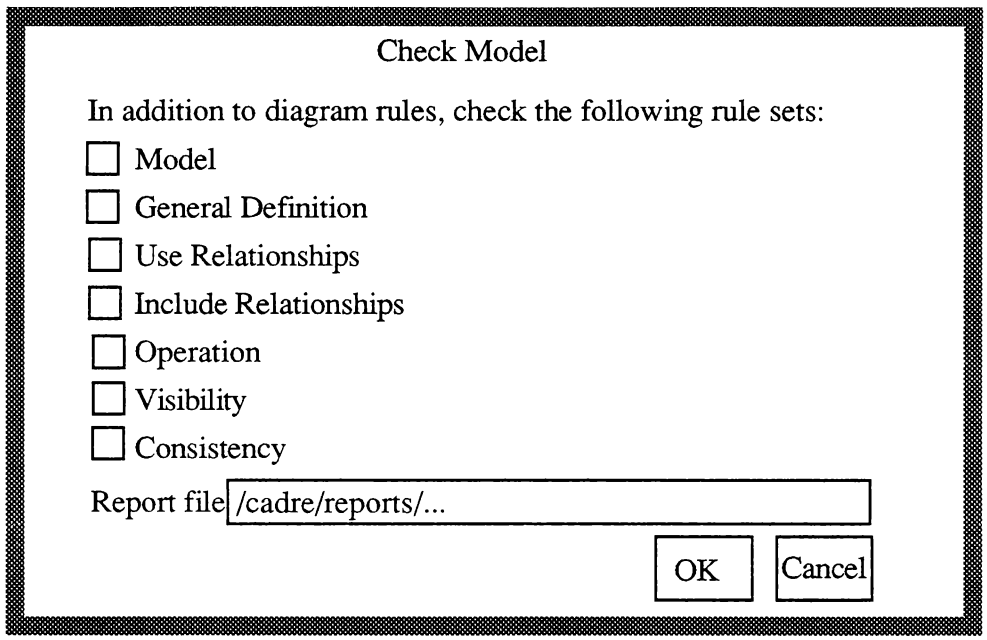

Figure 19. Model Check List

These model checks includes checking for inconsistencies in many different aspects of HOOD design such as the following:

Interface Checks.

One of the checks done by HOOD is to insure the interface between the objects are consistent. This should be done early in the development stage, for massive changes may be required to repair errors found late in the design.

Requirement Trace and Cross-reference Reports.

Quality assurance usually has the task of ensuring that all requirements are satisfied. HOOD allocates the requirements to each object as they are being designed to make sure the designer takes all of the requirements into consideration. This is then supported by a Cross-reference Report of Requirements to Objects.

Object completeness.

HOOD checks that each object has valid operations. An object that represents a piece of data would therefore need a constructor to put data into it, as well as a constructor to take data out.

Object/Operation Cross-reference Reports.

Checks are made to ensure that each provided operation is used somewhere in the design, and that required operations are provided somewhere in the design.

Justification

When a designer makes a choice as to which objects are used to build a system there must be some justification why these objects are selected. If a designer choose too many active objects a system may be become too complex. HOOD performs justification checks to ensures the system design being implemented has a logical purpose. For example, an object may be required to be active because it handles an interrupt, or it needs its own control flow to constrain two operations, or it is at a different priority level than other objects.

Completeness

Checks the HOOD design has no missing parts.

Naming standards

All names are checked to ensure they are all valid Ada names.

Coding standards

Coding standards are enforced by the Ada code generator, an optional package at

this time.

DFD checking

Checks that each data flow diagrams is correct, and that each entity is mapped into the HOOD design.

CORE threads

To ensure the complete mapping of core threads, (i.e. individual requirement names) into the design.

5.1 THE SYSTEM OBJECT

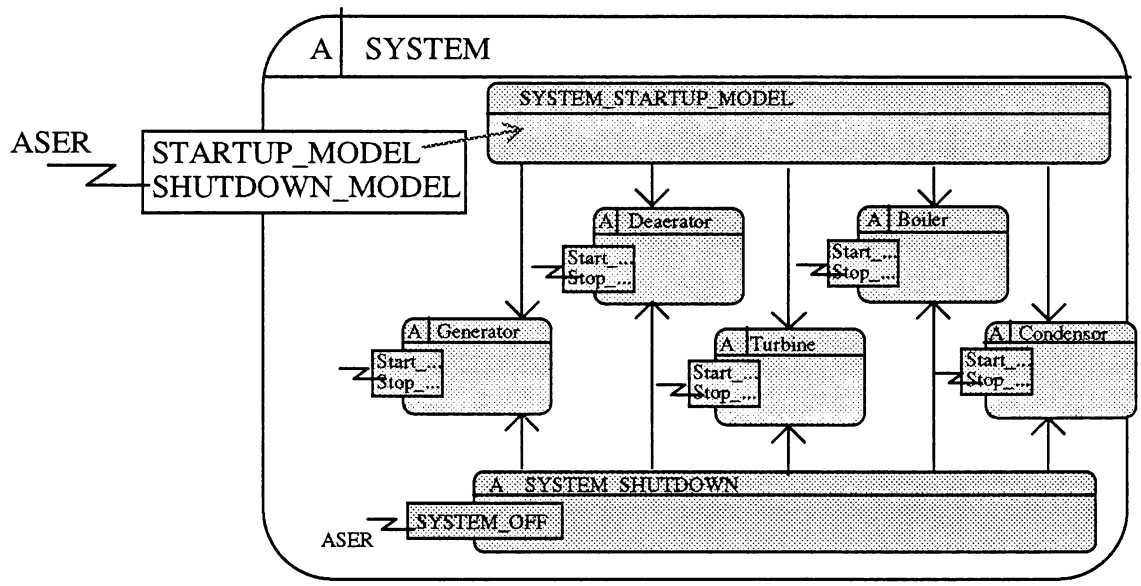

Figure 20. The SYSTEM Object.

The system object is normally defined at the top of the HOOD hierarchy. This object is referred to as the root. The root is then hierarchically decomposed into others called parent objects. The parent objects are then in turn decomposed into their child objects and relevant environmental objects, each defined in its own HSG (Hierarchy Structure Graph).

5.2. THE TURBINE OBJECT

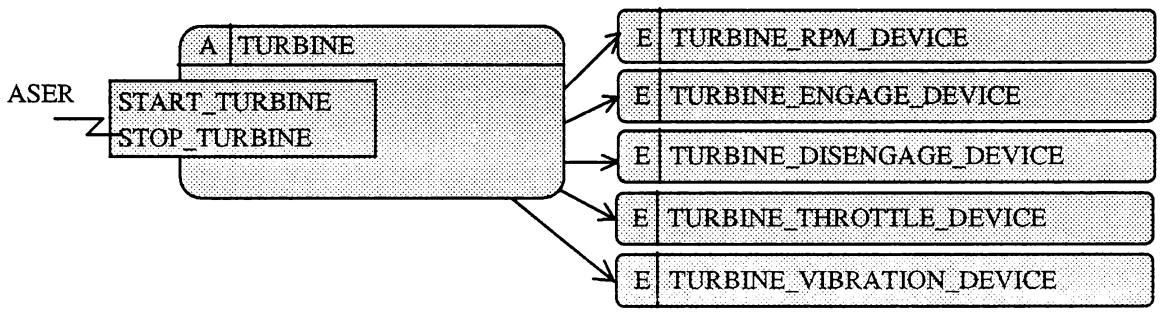

Figure 21. The Turbine Object.

The turbine must maintain a speed of 3600 rpm as the generator is loaded. The temperature of the steam entering the turbine must be maintained at 1005° F. Any variation in the steam temperature or pressure may affect the RPM of the turbine. The steam pressure is varied with the turbine throttle so that the RPM can be kept to within ± 3%. At 110% the turbine over speed trips and the turbine is automatically shutdown. Vibration sensors are used to determine if there is a problem developing with the bearings in the turbine.

5.3. THE GENERATOR OBJECT

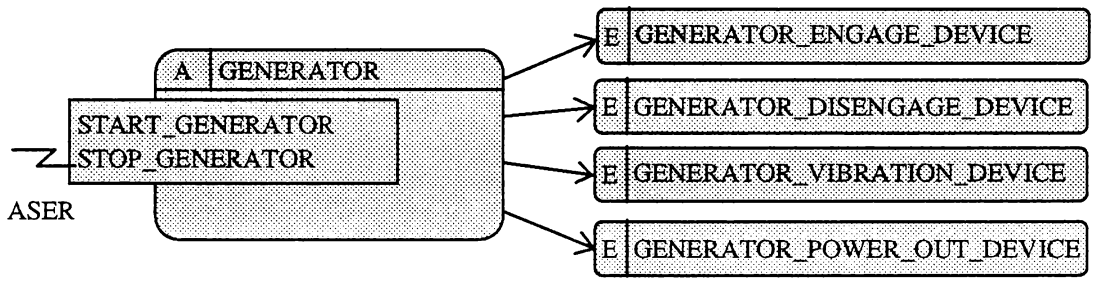

Figure 22. The Generator Object.

The generator maintains a constant output of 400 megawatts. It also has vibration sensors to monitor the bearings. It is important for the generator to use power out monitors to determine the electrical current generated is within specifications so it may be switched into the power grid.

5.4 THE CONDENSER OBJECT

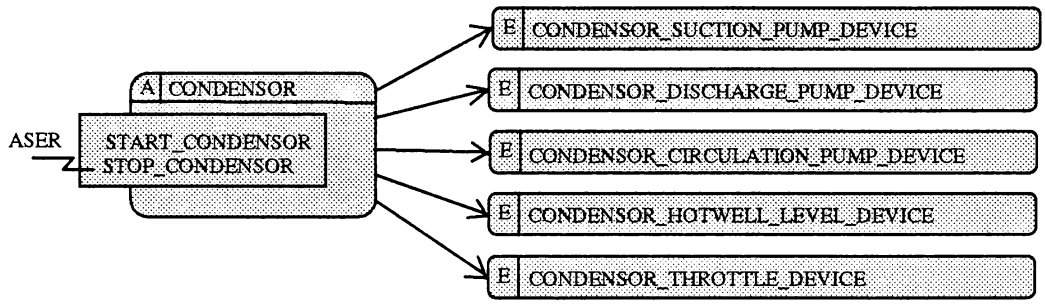

Figure 23. The Condenser Control Object.

The condenser is used to convert the steam back into water after it leaves the turbine. The condenser has a circulating suction pump for bringing in an external water supply, usually lake water to cool the hotwell vessel containing the steam. There is a circulating discharge pump for removing the water. The hotwell level is monitored to avoid to much water from collecting.

A pump empties the water from the hotwell after it reaches a specified level. This pump operates at 300-400 psi. If the pressure gets any higher than the normal operating pressure, a steam blow off valve is opened to remove the excess pressure.

5.5 THE DEAERATOR OBJECT

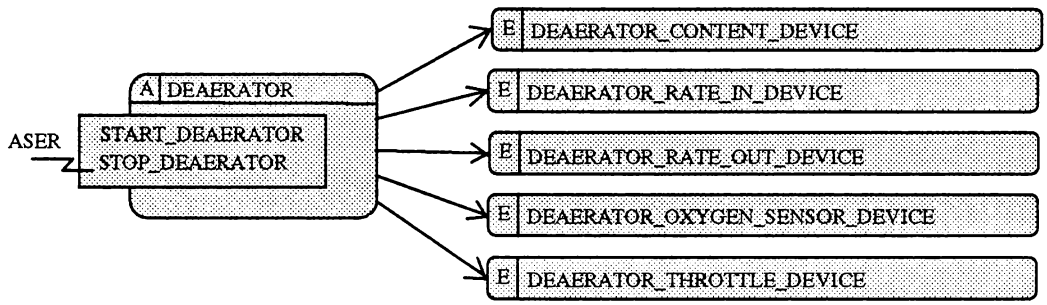

Figure 24. The Deaerator Control Object.

The deaerator is used to remove excess oxygen in the water returning from the condenser. There are a number of sensors present in the deaerator, the first to measure oxygen content. The water level is monitored by a sensor, when the level is too high the loading of the turbine is lowered to compensate. The input condensate flow as well as the feed water flow are both measured.

5.6. THE BOILER OBJECT

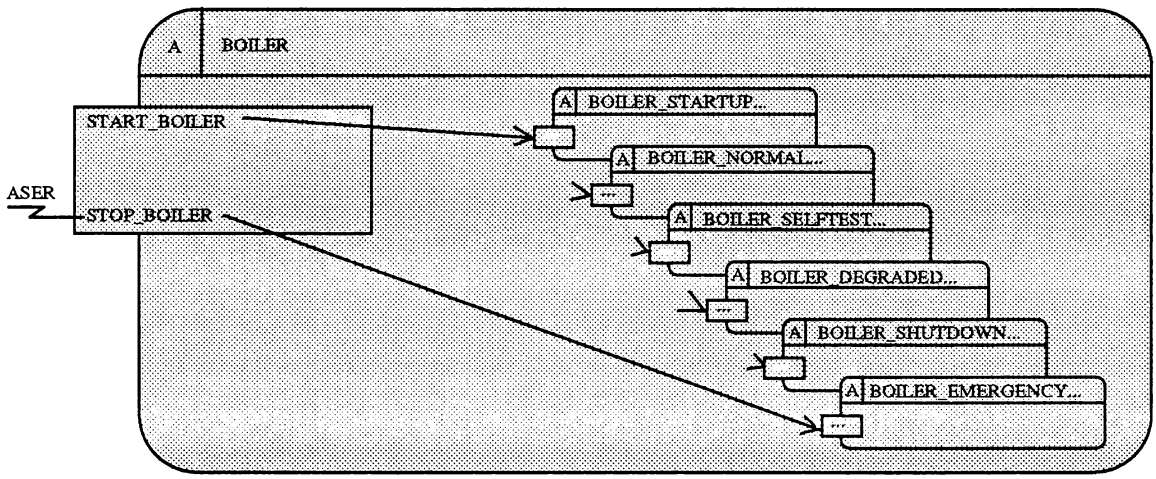

Figure 25. The Boiler Object.

The Boiler object is fashioned after the boiler simulator produced by NCR. The boiler is capable of six different modes of operation depending upon the state of the sensors and their measurements.

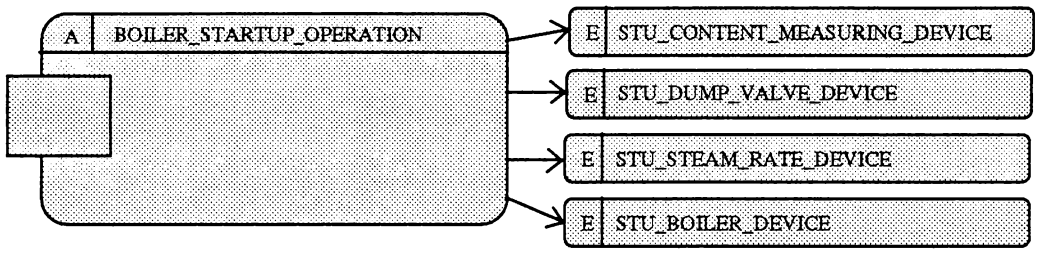

Figure 26. The Boiler Startup Object

During the boiler startup operation there are a number of operations to complete. First, after the boiler is initially powered up, it sends a signal that it is on standby. After this signal arrives, the boiler content device is checked for normal operation. Next the steam rate device is checked to make sure it is reading zero. If the boiler content device sensors reading is over 60,000 pounds, the normal maximum operating level, the boiler is too high and the dump valve is turned on and it waits until the water content has been adjusted to 60,000 pounds. If the boiler content is below 40,000 pounds then the any one of the water feed pumps is turned on until the water rises to 40,000 pounds. Next all of the feed pumps are turned on for at least 30 seconds and no more than 40 seconds. The water rate increase is then measured to ensure all of the feed pumps are working properly. If the water rate increase does not match the assumed calculation, the pumps are then turned on and off one at a time in order to find the faulty pump.

If the water content measuring or the steaming rate device is found to be faulty, then the boiler will go into shutdown mode. It will also go into shutdown mode if less than three pumps are in working order.

If all of the devices are found to be in working order, then the boiler is ready and goes into the normal mode. If three or more feed pumps or feed pump monitors are in

working order the boiler goes into the degraded mode.

It is only possible to go into the system startup mode from the self test mode.

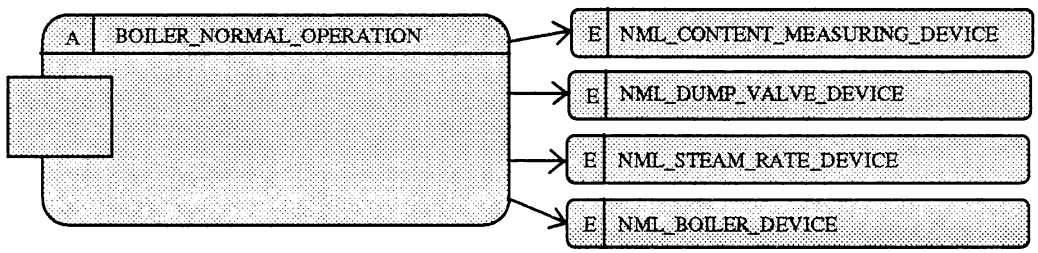

Figure 27. The Boiler Normal Object

In the normal operating mode the water level is between 65,000 and 85,000 pounds. It is maintained at this level by switching the feed pumps on and off.

If the water content measuring device fails, the boiler goes into the emergency mode. If any other device fails, the boiler goes into the degraded mode. If the water exceeds the limits of the safe operation of the boiler, the boiler goes into the shutdown mode.

The normal mode can be entered from the startup mode or the degraded mode.

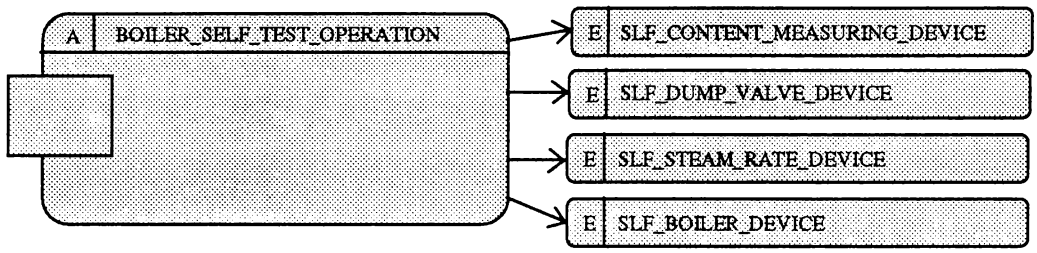

Figure 28. The Boiler Self Test Object

The self test mode first ensures that the communication links are all in proper working order. It tests all of the hardware connected to the boiler to ensure all of the devices are in working order. If any of the hardware fails, a message is displayed to the operator describing the device and how it failed. Control is then given to the operator.

If all of the hardware devices passes the test the boiler enters the startup mode. The self test mode can be entered from the shutdown mode, or by operator command from the keyboard.

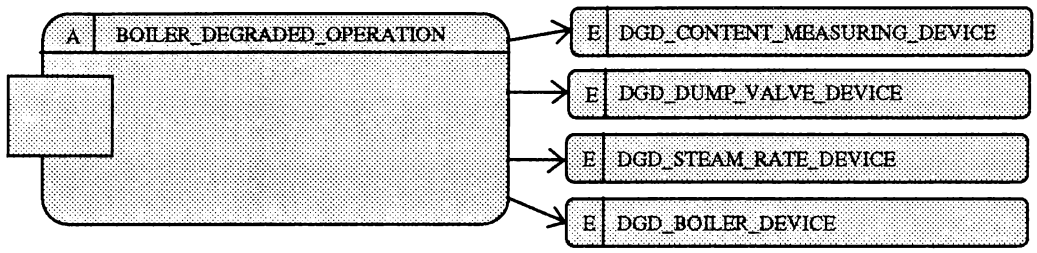

Figure 29. The Boiler Degraded Object

When the boiler is operating in the degraded mode, if the water content measuring device fails, the boiler goes into the emergency mode. If all failed devices are reported repaired, the boiler goes into the normal mode.

If the water content measuring device reports that the water level has exceeded the limits, the boiler goes into the shutdown mode.

The degraded mode can be entered from the normal mode, emergency mode, and the startup mode.

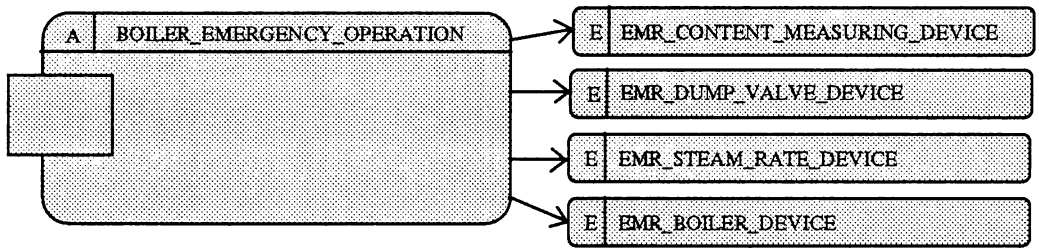

Figure 30. The Boiler Emergency Object

During the emergency mode, calculations are made to predict the worst possible behavior from all system devices. Using this information it attempts to operate the boiler within the safe limits specified in the boiler specification.

The boiler can exit the emergency mode and enter the degraded mode when the water content measuring device becomes available. If the water level starts to exceed the limits of safe operation, or the water content calculations are not possible, the boiler then moves into the shutdown mode. Shutdown mode is also entered if the steam rate device fails.

Emergency mode can be entered from the normal mode or degraded mode.

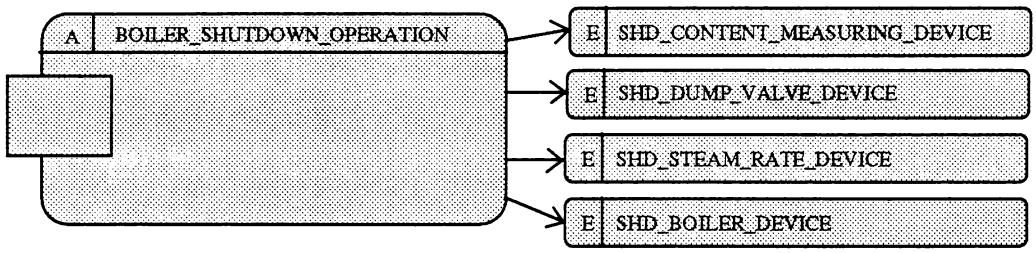

Figure 31. The Boiler Shutdown Object

When shutdown mode is entered a message must be displayed to inform the operator why the boiler is being shut down. The operator can then confirm shut down and if desired enter into the boiler self test mode.

After the shutdown mode is entered, the boiler can only go into the self test mode. Shutdown mode can be started by any other mode.

5.7. SUMMARY

Using the HOOD rules as a guideline, the entire power plant real-time system was designed and checked for consistency. From this design work several hundred pages of Object Description Skeleton was generated for all objects in the system. In Appendix B the ODS for the boiler object is presented.

6.0 IMPLEMENTATION

To design a system with intelligent sensors the designers must first break the entire system up into smaller objects. This is done in a manner which minimizes the number of "*threads*" or messages that are passed between the objects in the system. In order to decrease the number of threads each object , or "*intelligent sensor*", may contain a hierarchy of smaller objects, or "*embedded sensors*", within the intelligent sensor. These embedded sensors may talk to each other through serial or parallel ports discussed earlier. Data that is needed by the overall system can be sent by the intelligent sensor to any other object in the network. This information can then be stored in a database at a central location and displayed to the human operators.

In an industrial setting the objects of the system would then be split up and given to each programming team to implement. The boiler object can then be easily coded so that it will run on a dedicated intelligent sensor with multiple micro controllers and report the boiler state to the other intelligent sensors and central computer.

The boiler object was chosen to implement an example of a control program for an intelligent sensor. By using the NCR boiler simulator as the actual hardware device, an Ada program, as shown in Appendix C, has successfully been implemented to control the various aspects of boiler operations.

As shown in Figure 32, the program consists of a Receive task, a Transmit task, a User Interface task, and a set of tasks collectively called the Modes tasks. Receive and Transmit tasks communicate via the IBM PC COM1 port with the boiler simulator receiving on another PC.

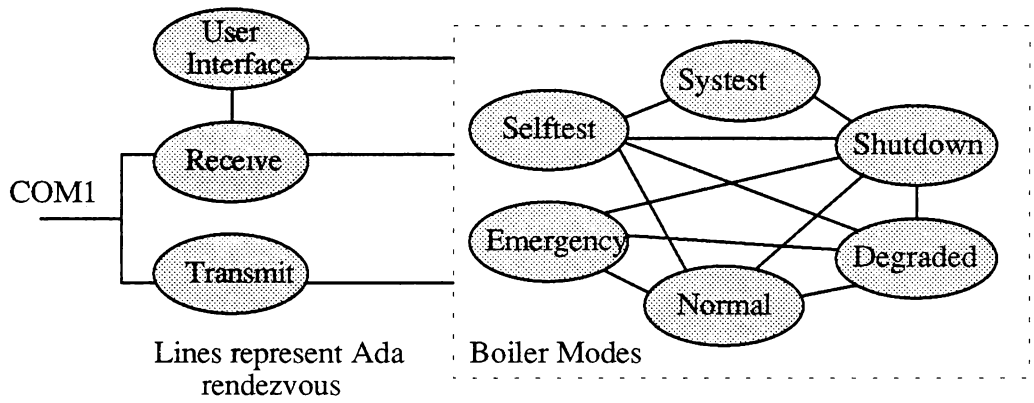

Figure 32. Boiler Control Program.

The lines between the tasks represent messages that are exchanged between the tasks. These messages consist of state table information. Each task has a "local state table" that needs to be updated periodically. The receive task is considered to be the holder of the "global state table" since it sits at the point where all of the incoming message traffic enters the program. It is responsible for "updating" the local state table of all of the other tasks.

The user interface task contains DOS calls to write the most current information to the display. It also reads the keyboard and carries out the user commands.

The boiler modes packages contain all of the "modes" of the boiler. Only one of these modes may be active at a time during execution. When the proper checks are made a task may enter another mode by making a rendezvous with the task. The calling task will go into a "wait" state until a task performs a rendezvous with it.

7.0 CONCLUSION

In this thesis the definition of the intelligent sensor with its four functional properties, preprocessing, fault-tolerance, communications, and global awareness, has been presented, along with several example sensor configurations. The major components of a real-time system using intelligent sensors have been identified and discussed.

The power plant model has been introduced as an example of the necessity of reliable control systems. This model has been used to verify the approach taken to design such systems. The power plant design was produced using the HOOD methodology and verified using the Teamwork/HOOD tool. A single object from the plant model, the boiler, was selected to implement the Ada code necessary to control the intelligent sensor. The implementation was checked using the Canadian NRC boiler simulator.

The presented procedure shows the feasibility of intelligent sensors design in modern real-time systems with off-the-shelf hierarchical object oriented design tools. It has been demonstrated that the use of such procedure leads to an immediate and successful implementation in Ada. The areas I foresee requiring future attention in designing real-time systems with intelligent sensors are faster and more reliable deterministic communications networks and new statistical tools for better and more efficient prediction models.

REFERENCES

1. Burns, A., and Wellings, A., *Real-time Systems and Their Programming Languages,* Addison-Wesley, Reading, MA. 1989.

2. Campbell, J., *C Programmer's Guide to Serial Communications,* Howard W. Sams & Company, Indianapolis, Indiana, 1988.

3. Comer, D., Steven, D. L., *Internetworking with TCP/IP, Volume 3*, Prentice Hall, Englewood Cliff, New Jersey, 1993.

4. Gomaa, H., *Software Design Methods for Real-time Systems*, Curriculum Module Report SEI-CMU-22-1.0, Software Engineering Institute, Pittsburgh, PA, December 1989.

5. HOOD Working Group, *HOOD Reference Manual, Issue 3.0,* European Space Agency, WME/89-173/JB, 1989.

6. IEEE P1596.x-02Oct92-docTBD, *SCI/RT Scalable Coherent Interface For Real-Time Applications.*

7. Lamport, L., *Time, Clocks and the Ordering of Events in Distributed Systems*, Comm. of the ACM, Vol 21, No. 7, pp 558-565, 1978.

8. Lee, I., King, R. B., Paul, R. P., *A Predictable Real-time Kernel for Distributed Multisensor Systems,* Computer,Vol 22, No.6, pp. 78-83, June 1989.

9. Mills, D., *Internet Time Synchronization: the Network Time Protocol*, DARPA Network Working Group, RFC-1129, M/A-COM Linkabit, October 1989.

10. Muntz, A., Horowitz, E., *A Framework for Specification and Design of Software for Advanced Sensor Systems*, Proceedings of the 10th Real-time Systems Symposium, Santa Monica, CA, pp 204-213, Dec 5-7th, 1989.

11. Shin, K., and Hou, C., *Analysis of Three Contention Protocols in Distributed Real-time Systems,* Proceedings of the 11th Real-time Systems Symposium, Lake Buena Vista, Florida, pp 136-145, Dec 5-7th, 1990.

12. Stankovic, J., *Misconception Abount Real-time Computing: A Serious Problem for Next-Generation Systems*, Computer, Vol 20, No. 10, pp. 10-19, October 1988.

13. Stankovic, J. *Real-time Computing Systems: The Next Generation*, Technical Report TR-88-06, COINS Dept., University of Massachusetts at Amherst, January 1988.

14. Strayer, W. Timothy, Dempsey, Bert J., Weaver, Alfred C., *XTP, The Xpress Transfer Protocol*, Addison-Wesley, Reading MA, 1992.

15. Stewart, D. B., Volpe, R.A., and Khosla, P.K., *Integration of Real-time Software Modules for Reconfigurable Sensor-Based Control Systems*, Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '92), Raleigh, NC, pp. 139-146, July 7-10, 1992.

16. Sun Microsystems, *Network Programming Guide*, Part Number: 800-3850-10, Revision A of March 1990.

17. Taylor, G., Ramblin, and Ciufo, C., Designing a High-Performance VME-to-FDDI Controller Board, VMEbus Systems, February 1992.

18. Natarajan, S., and Zhao, W., *Issues in Buiding Dynamic Real-time Systems*, IEEE Software, Vol. 9, No. 5, pp. 16-21, September 1992.

19. Ohba, R., Apps, F. R. D., *Intelligent Sensor Technology*, John Wiley & Sons. New York 1992

APPENDIX A

Sensor Classification Table

Classification		Types
Dynamic	Displacement, length, volume, location, level	Sliding resistors, spring type quantities variable resistors, resistance strain gauges, foil type strain gauges. semiconductor strain gauges, resistance thickness gauges, electrolyte level indicators, displacement meters using electrolytes, movable electrode electron tubes, capacitance type micrometers/thickness gauges/liquid level gauges, inductance type micrometers/thickness gauges, liquid level gauges (glass gauge types, differential pressure types, floating particle types, display types, sounding types, ultrasonic types, electrostatic capacitance types, radiation types), angle displacement detection, differential transformers, syncro, magnesyn, inductosyn, hall elements, radiation thickness meters, radiation level indicators, snow gauges, interference rulers, ultrasound level indicators, displacement measurement using radar, sonar, loran or tracer, parameter modulation in oscillation circuits, ultrasound thickness gauges, thermal micrometers, thermal angle displacement meters, thermal level indicators, pressure level indicators, springs, nozzle flappers (air micrometers), hydraulic spray tubes, air pressure level indicators, Moiré' fringes, floats, glass gauges, optical levers, microscopes, holography.
	Rotation	Inductance angle displacement

	detection, rotational speed, gyroscopes (mechanical, ring lasers, optical fiber's)
Direction	Direction sensors (geomagnetic detection, gas rate gyro)
Distortion	Strain gauges (resistance wire, semiconductor, optical fiber, photo elasticity)
Pressure, force/torque, twisting, weight	Manganin pressure gauges, pressure diodes, ionization vacuum gauges, inductance gravimeters, magnetic modulation transformers (Hillery effect), photo elastic elements, vibrotrons, Pirani gauges, seismographs, dynamometers, springs, elastic excess weight detectors, quartz wiring, torque tubes, span bands, bellows, diaphragms, capsules, Bourdon tubes, U-tubes, incline tubes, ring balances, Bell differential pressure gauges, refractive pressure gauges, tension sensors, composite semiconductor sensors (differential pressure, static pressure, temperature), load cells.
Speed, acceleration	Piezo elements, drift potential accelerometers, carbon board resistors, carbon grain resistors, capacitance speedometers, interference flow meters (LDV), Mossbauer effect (detection of very small differences in speed using gamma-ray resonance absorption by the atomic nucleus), thermal anemometers, throttle mechanisms (orifices, venturis, nozzles), pitot tubes, viscosity flow-meters, differential flow meters, turbine mass flow meters, tachometer blowers, flow meters and flow rate meters using Coriolis force,

		gyroscopes, impeller mass flow meters, Laser radar (lidar)
	Flow	Rotating generators, rotating disk speedometers, vibrometers (Ballistocardiographs), electro-magnetic flow meters, vortex flow meters, flow meters using electrolytes, ultrasound flow meters, Doppler radar, microwave speedometers, LDV, Laser gyros, flow meters using tracers, Thomas flow meters, thermal liquid flow meters, rotameters, wet gas meters, dry gas meters, turbine flow meters, water meters, weirs, fine flow rate sensors, wind speed sensors, water-leak sensors.
	Sound and vibration spectra	Microphones, pick-ups, vibration, noise, AE sensors
	Dew-point/humidity	Resistance hygrometers, dew-point meters, hair hygrometers, cloud sensors, ceramic humidity sensors.
	Proximity/passage ON/OFF	Metal detectors, reed switches, limit switches
Heat/energy	Temperature	Thermocouples, optical thermometers, color thermometers, radiation thermometers, pyro-electric elements, thermoelectromagnetic effect, temperature-measuring resistors (platinum, nickel), thermisters, posistors, ceramistors, solution thermometers, dielectric thermometers, inductance thermometers, quartz thermometers, gas thermometers, liquid pressure thermometers, vapor pressure thermometers, bimetallic thermometers, thermography, thermoelectric sensors, thermal

		resistor heat flow meters (thermopile types, temperature-measuring resistor types)
	Heat	Calorimeters, heat flow meters
Electromagnetic optical quantities	Voltage, current, frequency phase	Electron tube amplifiers, semiconductor amplifiers capacitance frequency meters, magnetic amplifiers, rotating amplifiers, optoelectronic Kerr effect elements, electric field light emission, LEDs, frequency modulation circuits, voltage/frequency conversion using integrators, vacuum thermocouples, Moving coil/iron leaf instruments, inductance instruments, electrodynamic instruments, static electric voltmeters, current balances and other similar electrical instruments, vibrating reed frequency meters, electrical distortion elements, piezo elements (reverse voltage effect), oscilloscopes, differential voltage instruments
	Visual/images (image sensors) Light (infra-red visible, radiation)	Photoelectric cells, silicon conductor photoconductive cells, photoelectric tubes (electric eyes), photoelectric multipliers, photo diodes, photo-transistors, semiconductor radiation detectors, ionization chambers, proportional counter tubes, Geiger counters, scintillators, bolometers, exposure photometers, laser sensors (plasma, emission spectrum analysis, Raman spectrum, high resolution spectrum)
	Magnetism	Hall elements, bismuth elements, semiconductor magnetic reluctance elements, magnetrons, thermion beam tubes, magnetic diodes, nuclear magnetic resonance (NMR), SQUID, fluxmeters, (carbon constitute

		meters), Faraday effect elements, magneto-optical Kerr effect elements, magnetic transistors
Chemical quantities	Time	Time measurement using integrators
	Ion Sensors	Glass thin-film ion sensors (pH, hydrogen ions, sodium, potassium, lithium, ammonia, cesium, etc.), Solid-state membrane ion sensors (cyanogen, chlorine, bromine, sulfur, iodine, fluorine, silver, copper, lead, cadmium, etc.), liquid film ion sensors (nitric acid, perchloric acid, chlorine, copper, calcuim, etc.) Enzymatic ion sensors (urea, glucose, etc.) Diaphragmatic ion sensors (ammonia, cyanogen, etc.), Conductivity sensors (bridge method, electromagnetic induction method) Various ion sensors (cyan-AgI, Cadmium-CdS, Chlorine-AgC1, Bromine-AgBr, Iodine-AgI, Flourine-LaF, Silver-AgS, Copper-Cus, Lead-ZnS, Nitric acid-$Ni(O\text{-}Phen)^{+}{}_{32}$, Perchloric acid-$FE(O\text{-}Phen)^{+}{}_{32}$, Calcium-$(RO)_2PO_2^-$
	Gas Sensors	Gas sensors - contact combustion sensors (combustible gases), semiconductor sensors (main constituents SnO_2) (combustible gases), electro-chemical sensors (exposed electrode types, diaphragm electrode types) (Toxic gases, etc., NH_4, SO_2, CO_2, HCN, H_2S, Cl_2, Br_2, O_2), zirconia oxygen concentration meters, oxide semiconductor gas sensors (Redox gases ZnO thin films, combustible gases - SnO_2), Redox gases - Oxide thin films (SnO_2, CdO, Fe_2O_3, NiO), reducing gas - oxides (WO_3, MoO, CrO, etc.) catalyst (Pt, Ir, Rh, Pd, etc.), hydrogen gas / hydrocarbon

	- In_2O_3+Pt, reducing gas - SnO_2+Pd, hydrogen gas - WO_3+Pt, Ethanol -oxide complexes ($LaNiO_3$ etc.), nitrogen oxide - V_2O_5 + Ag, Oxygen - CoO, Ethane / Butane etc. ZnO + Pt, Hydrogen gas / carbon monoxide - ZnO + Pd, reducing gases - Sn_2 + transition metals) Ultraviolet analysis meters (dissolved ozone, active chlorine and other such oxidizing substances: organic pollutants, COD: ammoniachlorine, mercury, nitrogen dioxide, sulpher dioxide, hydrogen sulphide, etc.) Infra-red gas analysis meters (CO, CO_2, methane, ethane, propane, butane, acetylene and other such hydrocarbons, nitrix oxide, nitrogen dioxide, hydrogen chloride, ammonia).
Analytical sensors (gas, smell, concentration, pH, smoke, moisture content)	Glass electrodes, hydrogen electrodes, antimony electrodes, Rodex potential constituent membrane potential constituent meters, solution densitometers, electromagnetic densitometers, nuclear magnetic resonance (NMR), electron spin resonance (ESR), voltametry., Polarographs, rotating viscometers, electromotive force solid constituent meters, saccharimeters, X-ray characteristics (X-ray microanalysers), radio-chemical spectroscopy, emission spectroscopy, fluorescence spectroscopy, ultrasound gas constituent meters, ultrasound liquid densito-meters, ultrasound viscometers, time of flight (TOF) mass spectrographs, gas chromatographs, liquid chromatographs, photographic density meters, absorption spectrometers, colorimeters,

	hydrogen sulphide meters, infra-red spectrometers, radiation densimeters, radiation sulphur spectro-meters, meutron moisture meters, microwave moisture meters, heat condution gas constituent meters, magnetic oxygen meters, combusion gas constituent meters, chlorine densitometers, floating gravimeters, pressure densimeters, gas constituent meters using density differences, airpressure gravimeters, oscillating viscometers, refractive gas spectrometers, mass spectometers, oxygen sensors, viscosity sensors.
Biological quantities	Blood pressure sensors, blood flow sensors, electromagnetic blood flow meters, LD blood flowmeters, electronic clinical thermometers, enzymatic ion sensors (urea, glucose, etc.), biosensors, lactic acid sensors, uric acid sensors.
Sensory quantities	Touch, sight, hearing

APPENDIX B

ODS (Object Description Skeleton) files generated by Cadre Teamwork/HOOD for the classic design approach.

```
-------------------------------------------------------------------------
-- boiler_BOILER.ods
-------------------------------------------------------------------------

OBJECT BOILER IS ACTIVE

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      EMERGENCY_OPERATION
         ;
      --|here|--

      DEGRADED_OPERATION
         ;
      --|here|--

      SELF_TEST_OPERATION
         ;
      --|here|--

      NORMAL_OPERATION
         ;
      --|here|--

      STOP_BOILER
         ;
```

```
      --|here|--

      START_BOILER
         ;
      --|here|--

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE

   EXCEPTIONS
      NONE

DATAFLOWS
   NONE

OBJECT_CONTROL_STRUCTURE

   DESCRIPTION
      --|here|--

   CONSTRAINED_OPERATIONS
         STOP_BOILER CONSTRAINED_BY (ASER
         --|here|-- );

   IMPLEMENTED_BY
```

```
      BOILER_SHUTDOWN_OPERATION;

INTERNALS

   OBJECTS
      BOILER_STARTUP_OPERATION;
      --|here|--
      BOILER_NORMAL_OPERATION;
      --|here|--
      BOILER_SELFTEST_OPERATION;
      --|here|--
      BOILER_DEGRADED_OPERATION;
      --|here|--
      BOILER_EMERGENCY_OPERATION;
      --|here|--
      BOILER_SHUTDOWN_OPERATION;
      --|here|--

   DECLARATIONS
      NONE

   OPERATIONS
      NONE

OPERATION_CONTROL_STRUCTURE

   OPERATION EMERGENCY_OPERATION
      IS

      DESCRIPTION
         --|here|--

         USED_OPERATIONS
            NONE

         EXCEPTIONS
            NONE

      IMPLEMENTED_BY BOILER_EMERGENCY_OPERATION.EMERGENCY

   END_OPERATION EMERGENCY_OPERATION

   OPERATION DEGRADED_OPERATION
      IS
```

```
  DESCRIPTION
    --|here|--

    USED_OPERATIONS
      NONE

    EXCEPTIONS
      NONE

  IMPLEMENTED_BY BOILER_DEGRADED_OPERATION.DEGRADED

END_OPERATION DEGRADED_OPERATION

OPERATION SELF_TEST_OPERATION
  IS

  DESCRIPTION
    --|here|--

    USED_OPERATIONS
      NONE

    EXCEPTIONS
      NONE

  IMPLEMENTED_BY BOILER_SELFTEST_OPERATION.SELFTEST

END_OPERATION SELF_TEST_OPERATION

OPERATION NORMAL_OPERATION
  IS

  DESCRIPTION
    --|here|--

    USED_OPERATIONS
      NONE

    EXCEPTIONS
      NONE

  IMPLEMENTED_BY BOILER_NORMAL_OPERATION.NORMAL

END_OPERATION NORMAL_OPERATION
```

```
   OPERATION STOP_BOILER
      IS

      DESCRIPTION
         --|here|--

         USED_OPERATIONS
            NONE

         EXCEPTIONS
            NONE

      IMPLEMENTED_BY
BOILER_SHUTDOWN_OPERATION.SHUTDOWN_BOILER

   END_OPERATION STOP_BOILER

   OPERATION START_BOILER
      IS

      DESCRIPTION
         --|here|--

         USED_OPERATIONS
            NONE

         EXCEPTIONS
            NONE

      IMPLEMENTED_BY BOILER_STARTUP_OPERATION.STARTUP_BOILER

   END_OPERATION START_BOILER

END_OBJECT BOILER
```

```
-------------------------------------------------------------------------
-- boiler_BOILER_DEGRADED_OPERATION.ods
-------------------------------------------------------------------------

OBJECT BOILER_DEGRADED_OPERATION IS ACTIVE

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      DEGRADED
         ;
      --|here|--

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      DGD_CONTENT_MEASURING_DEVICE;
      DGD_STEAM_RATE_DEVICE;
      DGD_DUMP_VALVE_DEVICE;
      DGD_BOILER_DEVICE;

   CLASS_OBJECTS
      NONE
```

TYPES
NONE

OPERATIONS
NONE

EXCEPTIONS
NONE

DATAFLOWS
NONE

OBJECT_CONTROL_STRUCTURE

DESCRIPTION
--|here|--

CONSTRAINED_OPERATIONS
NONE

CODE
--|here|--

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE

OPERATION DEGRADED
IS
DESCRIPTION
--|here|--
USED_OPERATIONS
NONE
EXCEPTIONS
NONE

```
    CODE
      --|here|--
      EXCEPTION_HANDLER
        NONE

  END_OPERATION DEGRADED

END_OBJECT BOILER_DEGRADED_OPERATION
```

```
-------------------------------------------------------------------
-- boiler_BOILER_EMERGENCY_OPERATION.ods
-------------------------------------------------------------------

OBJECT BOILER_EMERGENCY_OPERATION IS ACTIVE

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      EMERGENCY
         ;
      --|here|--

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      EMR_CONTENT_MEASURING_DEVICE;
      EMR_STEAM_RATE_DEVICE;
      EMR_DUMP_VALVE_DEVICE;
      EMR_BOILER_DEVICE;

   CLASS_OBJECTS
      NONE
```

TYPES
NONE

OPERATIONS
NONE

EXCEPTIONS
NONE

DATAFLOWS
NONE

OBJECT_CONTROL_STRUCTURE

DESCRIPTION
--|here|--

CONSTRAINED_OPERATIONS
NONE

CODE
--|here|--

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE

OPERATION EMERGENCY
IS
DESCRIPTION
--|here|--
USED_OPERATIONS
NONE
EXCEPTIONS
NONE
CODE

```
        --|here|--
        EXCEPTION_HANDLER
          NONE

    END_OPERATION EMERGENCY

END_OBJECT BOILER_EMERGENCY_OPERATION
```

```
-------------------------------------------------------------------------
-- boiler_BOILER_NORMAL_OPERATION.ods
-------------------------------------------------------------------------

OBJECT BOILER_NORMAL_OPERATION IS ACTIVE

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NORMAL
         ;
      --|here|--

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NML_CONTENT_MEASURING_DEVICE;
      NML_STEAM_RATE_DEVICE;
      NML_BOILER_DEVICE;
      NML_DUMP_VALVE_DEVICE;

   CLASS_OBJECTS
      NONE
```

```
   TYPES
      NONE

   OPERATIONS
      NONE

   EXCEPTIONS
      NONE


DATAFLOWS
   NONE

OBJECT_CONTROL_STRUCTURE

   DESCRIPTION
      --|here|--

   CONSTRAINED_OPERATIONS
      NONE

   CODE
      --|here|--

INTERNALS

   OBJECTS
      NONE

   DECLARATIONS
      NONE

   OPERATIONS
      NONE

OPERATION_CONTROL_STRUCTURE

   OPERATION NORMAL
      IS
      DESCRIPTION
         --|here|--
         USED_OPERATIONS
            NONE
         EXCEPTIONS
            NONE
```

```
      CODE
        --|here|--
        EXCEPTION_HANDLER
          NONE

    END_OPERATION NORMAL

END_OBJECT BOILER_NORMAL_OPERATION
```

```
-------------------------------------------------------------------------
-- boiler_BOILER_SELFTEST_OPERATION.ods
-------------------------------------------------------------------------

OBJECT BOILER_SELFTEST_OPERATION IS ACTIVE

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      SELFTEST
         ;
      --|here|--

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      SLF_CONTENT_MEASURING_DEVICE;
      SLF_STEAM_RATE_DEVICE;
      SLF_DUMP_VALVE_DEVICE;
      SLF_BOILER_DEVICE;

   CLASS_OBJECTS
      NONE
```

```
    TYPES
      NONE

    OPERATIONS
      NONE

    EXCEPTIONS
      NONE

DATAFLOWS
    NONE

OBJECT_CONTROL_STRUCTURE

    DESCRIPTION
      --|here|--

    CONSTRAINED_OPERATIONS
      NONE

    CODE
      --|here|--

INTERNALS

    OBJECTS
      NONE

    DECLARATIONS
      NONE

    OPERATIONS
      NONE

OPERATION_CONTROL_STRUCTURE

    OPERATION SELFTEST
      IS
      DESCRIPTION
        --|here|--
        USED_OPERATIONS
          NONE
        EXCEPTIONS
          NONE
      CODE
```

```
        --|here|--
        EXCEPTION_HANDLER
          NONE

    END_OPERATION SELFTEST

END_OBJECT BOILER_SELFTEST_OPERATION
```

```
-----------------------------------------------------------------------
-- boiler_BOILER_SHUTDOWN_OPERATION.ods
-----------------------------------------------------------------------

OBJECT BOILER_SHUTDOWN_OPERATION IS ACTIVE

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      SHUTDOWN_BOILER
         ;
      --|here|--

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      SHD_DUMP_VALVE_DEVICE;
      SHD_CONTENT_MEASURING_DEVICE;
      SHD_STEAM_RATE_DEVICE;
      SHD_BOILER_DEVICE;

   CLASS_OBJECTS
      NONE
```

```
   TYPES
      NONE

   OPERATIONS
      NONE

   EXCEPTIONS
      NONE

DATAFLOWS
   NONE

OBJECT_CONTROL_STRUCTURE

   DESCRIPTION
      --|here|--

   CONSTRAINED_OPERATIONS
         SHUTDOWN_BOILER CONSTRAINED_BY (ASER
         --|here|-- );

   CODE
      --|here|--

INTERNALS

   OBJECTS
      NONE

   DECLARATIONS
      NONE

   OPERATIONS
      NONE

OPERATION_CONTROL_STRUCTURE

   OPERATION SHUTDOWN_BOILER
      IS
      DESCRIPTION
         --|here|--
         USED_OPERATIONS
            NONE
         EXCEPTIONS
            NONE
```

```
      CODE
        --|here|--
        EXCEPTION_HANDLER
          NONE

    END_OPERATION SHUTDOWN_BOILER

END_OBJECT BOILER_SHUTDOWN_OPERATION
```

```
-------------------------------------------------------------------------
-- boiler_BOILER_STARTUP_OPERATION.ods
-------------------------------------------------------------------------

OBJECT BOILER_STARTUP_OPERATION IS ACTIVE

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      STARTUP_BOILER
         ;
      --|here|--

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      STU_DUMP_VALVE_DEVICE;
      STU_CONTENT_MEASURING_DEVICE;
      STU_STEAM_RATE_DEVICE;
      STU_BOILER_DEVICE;

   CLASS_OBJECTS
      NONE
```

```
    TYPES
      NONE

    OPERATIONS
      NONE

    EXCEPTIONS
      NONE

DATAFLOWS
    NONE

OBJECT_CONTROL_STRUCTURE

    DESCRIPTION
      --|here|--

    CONSTRAINED_OPERATIONS
      NONE

    CODE
      --|here|--

INTERNALS

    OBJECTS
      NONE

    DECLARATIONS
      NONE

    OPERATIONS
      NONE

OPERATION_CONTROL_STRUCTURE

    OPERATION STARTUP_BOILER
      IS
      DESCRIPTION
        --|here|--
        USED_OPERATIONS
          NONE
        EXCEPTIONS
          NONE
      CODE
```

```
        --|here|--
        EXCEPTION_HANDLER
          NONE

    END_OPERATION STARTUP_BOILER

END_OBJECT BOILER_STARTUP_OPERATION
```

```
-------------------------------------------------------------------------
-- boiler_DGD_BOILER_DEVICE.ods
-------------------------------------------------------------------------

OBJECT DGD_BOILER_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT DGD_BOILER_DEVICE

```
-------------------------------------------------------------------------
-- boiler_DGD_CONTENT_MEASURING_DEVICE.ods
-------------------------------------------------------------------------

OBJECT DGD_CONTENT_MEASURING_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT DGD_CONTENT_MEASURING_DEVICE

```
-------------------------------------------------------------------------
-- boiler_DGD_DUMP_VALVE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT DGD_DUMP_VALVE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT DGD_DUMP_VALVE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_DGD_STEAM_RATE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT DGD_STEAM_RATE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT DGD_STEAM_RATE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_EMR_BOILER_DEVICE.ods
-------------------------------------------------------------------------

OBJECT EMR_BOILER_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT EMR_BOILER_DEVICE

```
-------------------------------------------------------------------------
-- boiler_EMR_CONTENT_MEASURING_DEVICE.ods
-------------------------------------------------------------------------

OBJECT EMR_CONTENT_MEASURING_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT EMR_CONTENT_MEASURING_DEVICE

```
-------------------------------------------------------------------------
-- boiler_EMR_DUMP_VALVE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT EMR_DUMP_VALVE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT EMR_DUMP_VALVE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_EMR_STEAM_RATE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT EMR_STEAM_RATE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT EMR_STEAM_RATE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_NML_BOILER_DEVICE.ods
-------------------------------------------------------------------------

OBJECT NML_BOILER_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT NML_BOILER_DEVICE

```
-------------------------------------------------------------------------
-- boiler_NML_DUMP_VALVE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT NML_DUMP_VALVE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT NML_DUMP_VALVE_DEVICE

```
------------------------------------------------------------------------
-- boiler_NML_STEAM_RATE_DEVICE.ods
------------------------------------------------------------------------

OBJECT NML_STEAM_RATE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT NML_STEAM_RATE_DEVICE

```
------------------------------------------------------------------------
-- boiler_SHD_BOILER_DEVICE.ods
------------------------------------------------------------------------

OBJECT SHD_BOILER_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT SHD_BOILER_DEVICE

```
-------------------------------------------------------------------------
-- boiler_SHD_CONTENT_MEASURING_DEVICE.ods
-------------------------------------------------------------------------

OBJECT SHD_CONTENT_MEASURING_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT SHD_CONTENT_MEASURING_DEVICE

```
-------------------------------------------------------------------------
-- boiler_SHD_DUMP_VALVE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT SHD_DUMP_VALVE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT SHD_DUMP_VALVE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_SHD_STEAM_RATE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT SHD_STEAM_RATE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT SHD_STEAM_RATE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_SLF_BOILER_DEVICE.ods
-------------------------------------------------------------------------

OBJECT SLF_BOILER_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT SLF_BOILER_DEVICE

```
-----------------------------------------------------------------------
-- boiler_SLF_CONTENT_MEASURING_DEVICE.ods
-----------------------------------------------------------------------

OBJECT SLF_CONTENT_MEASURING_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT SLF_CONTENT_MEASURING_DEVICE

```
-------------------------------------------------------------------------
-- boiler_SLF_DUMP_VALVE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT SLF_DUMP_VALVE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT SLF_DUMP_VALVE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_SLF_STEAM_RATE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT SLF_STEAM_RATE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT SLF_STEAM_RATE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_STU_BOILER_DEVICE.ods
-------------------------------------------------------------------------

OBJECT STU_BOILER_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT STU_BOILER_DEVICE

```
-----------------------------------------------------------------------
-- boiler_STU_CONTENT_MEASURING_DEVICE.ods
-----------------------------------------------------------------------

OBJECT STU_CONTENT_MEASURING_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT STU_CONTENT_MEASURING_DEVICE

```
-------------------------------------------------------------------------
-- boiler_STU_DUMP_VALVE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT STU_DUMP_VALVE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT STU_DUMP_VALVE_DEVICE

```
-------------------------------------------------------------------------
-- boiler_STU_STEAM_RATE_DEVICE.ods
-------------------------------------------------------------------------

OBJECT STU_STEAM_RATE_DEVICE IS ENVIRONMENT

DESCRIPTION
   --|here|--

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   --|here|--

PROVIDED_INTERFACE

   TYPES
      NONE

   CONSTANTS
      NONE

   OPERATIONS
      NONE

   OPERATION_SETS
      NONE

   EXCEPTIONS
      NONE

REQUIRED_INTERFACE

   OBJECTS
      NONE

   ENVIRONMENT_OBJECTS
      NONE

   CLASS_OBJECTS
      NONE

   TYPES
      NONE

   OPERATIONS
      NONE
```

EXCEPTIONS
NONE

DATAFLOWS
NONE

INTERNALS

OBJECTS
NONE

DECLARATIONS
NONE

OPERATIONS
NONE

OPERATION_CONTROL_STRUCTURE
NONE

END_OBJECT STU_STEAM_RATE_DEVICE

APPENDIX C

```
--*************************************************************************
--
--                      BOILER
--
--      This program interfaces to the NRC's (National Research
--    Council) Boiler simulation. The model consists of a natural-
--    gas fired water tube boiler producing saturated steam through
--    seperators at the top of a drum. The steam flow may vary
--    rapidly and irregularly between zero and a maximum, following
--    a varying external demand.
--
--    written by: Jerry D. Cavin            date: April 3rd, 1993
--
--*************************************************************************

with tty;
with text_io;              -- text Input/Output package

with TX;                   use TX;
with RX;                   use RX;
with UI;                   use UI;
with OPERATIONAL_MODES;    use OPERATIONAL_MODES;

procedure BOILER is
   READY_TO_STOP, READY_TO_START : boolean := false;
begin
  text_io.put_line( "Start of Boiler Control Program." );

  -- start the user interface running
  USER_INTERFACE.CLEAR_SCREEN;
  USER_INTERFACE.UPDATE_LABELS;

  -- startup the communications
  RECEIVE.ENABLE(1);
  TRANSMIT.ENABLE(1);
  TRANSMIT.UNTIL_READY;

  while (READY_TO_STOP = false) loop
     USER_INTERFACE.CHECK_FOR_INITIATION(READY_TO_START);
     if READY_TO_START then
          STARTUP_OPERATIONS.ENABLE;
     end if;
     USER_INTERFACE.CHECK_FOR_TERMINATION(READY_TO_STOP);
```

```
   end loop;

   USER_INTERFACE.DISABLE;
   RECEIVE.DISABLE;            -- stop receiving messages
   TRANSMIT.DISABLE;           -- stop transmitting messages
   STARTUP_OPERATIONS.DISABLE;
   SHUTDOWN_OPERATIONS.DISABLE;
   NORMAL_OPERATIONS.DISABLE;
   DEGRADED_OPERATIONS.DISABLE;
   EMERGENCY_OPERATIONS.DISABLE;
   SELFTEST_OPERATIONS.DISABLE;
   tty.clear_screen;

   text_io.new_line;
   text_io.put_line( "Shutting down system tasks..." );
   USER_INTERFACE.KILL;
   TRANSMIT.KILL;
   RECEIVE.KILL;
   STARTUP_OPERATIONS.KILL;
   SHUTDOWN_OPERATIONS.KILL;
   NORMAL_OPERATIONS.KILL;
   DEGRADED_OPERATIONS.KILL;
   EMERGENCY_OPERATIONS.KILL;
   SELFTEST_OPERATIONS.KILL;
end;  -- of BOILER program
```

```
--
*****************************************************************************
--
--                    OPERATIONAL MODES Package
--
--
--     written by: Jerry D. Cavin            date: April 3rd, 1993
--
--
*****************************************************************************

with iio;                      -- integer Input/Output package
with text_io;                    -- text Input/Output package
with tty;                      -- DOS Video package
with interrupt;                  -- DOS call package
with time;                     -- DOS time function

with common_display_types;       -- color constants
use common_display_types;

with bit_ops;                    -- bit manipulation package
use bit_ops;

with TX;                       use TX;
with UI;                       use UI;
with SYSTEM_PARAMETERS;          use SYSTEM_PARAMETERS;

package OPERATIONAL_MODES is

  task STARTUP_OPERATIONS is
    entry ENABLE;
    entry DISABLE;
    entry UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE);
    entry KILL;
  end STARTUP_OPERATIONS;

  task SHUTDOWN_OPERATIONS is
    entry ENABLE;
    entry DISABLE;
    entry UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE);
    entry KILL;
  end SHUTDOWN_OPERATIONS;

  task DEGRADED_OPERATIONS is
    entry ENABLE;
```

```
    entry DISABLE;
    entry UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE);
    entry KILL;
  end DEGRADED_OPERATIONS;

  task EMERGENCY_OPERATIONS is
    entry ENABLE;
    entry DISABLE;
    entry UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE);
    entry KILL;
  end EMERGENCY_OPERATIONS;

  task NORMAL_OPERATIONS is
    entry ENABLE;
    entry DISABLE;
    entry UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE);
    entry KILL;
  end NORMAL_OPERATIONS;

  task SELFTEST_OPERATIONS is
    entry ENABLE;
    entry DISABLE;
    entry UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE);
    entry KILL;
  end SELFTEST_OPERATIONS;

end OPERATIONAL_MODES;

-------------------------------------------------------------------------

package body OPERATIONAL_MODES is

-------------------------------------------------------------------------

task body STARTUP_OPERATIONS is
--
--     During the boiler STARTUP_OPERATIONS there are a number of operations
-- to complete. First, after the boiler is initially powered up the boiler
-- sends a signal that it is on standby. After this signal arrives the
-- boiler content device is checked for normal operation. Next the steam
-- rate device is checked to make sure it is reading zero. If the boiler
-- content device is reading over 60,000 pounds the boiler is too high and
-- the dump valve is turned on and it waits until the water content has been
-- adjusted to 60,000 pounds. If the boiler content is below 40,000 pounds
-- then turn on any one of the water feed pumps until the water rises to
```

```
-- 40,000 pounds. Next all of the feed pumps are turned on for at least 30
-- seconds and no more than 40 seconds. The water rate increase is then
-- measured to ensure all of the feed pumps are working properly. If the
-- water rate increase does not match the assumed calculation the pumps are
-- then turned on and off one at a time in order to find the faulty pump.
--
-- ENTRIES:
-- It is only possible to go into STARTUP_OPERATIONS from the
-- SELFTEST_OPERATIONS.
--
-- EXITS:
-- If the water content measuring device or the steaming rate device is
-- found to be faulty then the boiler will go into SHUTDOWN_OPERATIONS.
-- It will also go into SHUTDOWN_OPERATIONS if there are less than three
-- pumps working. If all of the devices are found to be in working order
-- then the boiler is ready and goes into the NORMAL_OPERATIONS. If
-- three or more feed pumps or feed pump monitors are in working order
-- the boiler goes into the DEGRADED_OPERATIONS.
--
  ACTIVE : boolean := false;
  TASK_IS_RUNNING : boolean := true;
  LOCAL_STATE_TABLE : BOILER_STATE;
begin
  while TASK_IS_RUNNING loop
    select

      -- when starting this task set the active flag true and
        -- highlight the "SELFTEST" mode on the user interface.
        accept ENABLE do
        ACTIVE := true;
          USER_INTERFACE.HIGHLIGHT(SELFTEST);
        end ENABLE;

    or

        accept DISABLE do
          ACTIVE := false;
        end DISABLE;

    or

      -- kill the task by exiting the select loop
        accept KILL do
          TASK_IS_RUNNING := false;
        end KILL;
```

```
    or

      -- update the local state table with the new global state table
      accept UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE) do
        LOCAL_STATE_TABLE := GLOBAL_STATE_TABLE;
      end UPDATE;

    else

      if ACTIVE then
        if (LOCAL_STATE_TABLE.LEVEL_DEVICE /= WORKING) or
          (LOCAL_STATE_TABLE.RATE_DEVICE /= WORKING) then
            USER_INTERFACE.UNHIGHLIGHT(SELFTEST);
            SHUTDOWN_OPERATIONS.ENABLE;
          end if;

        if (LOCAL_STATE_TABLE.LEVEL > MAXIMUM_OPERATING_LEVEL)
then
          null;
          -- *** turn on dump valve ***
          -- *** wait till boiler content < 60_000 ***
        end if;

        if (LOCAL_STATE_TABLE.LEVEL < MINIMUM_OPERATING_LEVEL)
then
          TRANSMIT.TURN_ON_ALL_PUMPS;
          -- *** wait till boiler content > 40_000 ***
        end if;

        -- *** turn on all feed pump between 30-40 seconds ***
        -- *** if calculated water rise = boiler content then ***
          NORMAL_OPERATIONS.ENABLE; -- all pumps working
        -- *** else ***
        -- ***    find faulty pump ***
        -- *** end if; ***
         end if;

    end select;
  end loop;
  text_io.put_line( "... STARTUP_OPERATIONS Task is now terminating." );

  exception
    when others =>
        text_io.put_line( "*** Exception occured in STARTUP_OPERATIONS." );
```

```
end STARTUP_OPERATIONS;

-------------------------------------------------------------------------

task body SHUTDOWN_OPERATIONS is
--
--      When SHUTDOWN_OPERATIONS is entered a message must be displayed
-- to inform the operator why the boiler is being shut down.  The
-- operator can confirm shut down  and if desired enter into the boiler
-- self test mode.
--
-- ENTRIES:
-- SHUTDOWN_OPERATIONS can be entered from any mode.
--
-- EXITS:
-- After the SHUTDOWN_OPERATIONS the boiler can only go into
-- SELFTEST_OPERATIONS.
--
  ACTIVE : boolean := false;
  TASK_IS_RUNNING : boolean := true;
  LOCAL_STATE_TABLE : BOILER_STATE;
begin
  while TASK_IS_RUNNING loop
    select

      -- when starting this task set the active flag true and
      -- highlight the "SHUTDOWN" mode on the user interface.
        accept ENABLE do
        ACTIVE := true;
        USER_INTERFACE.HIGHLIGHT(SHUTDOWN);
        end ENABLE;

    or

        accept DISABLE do
          ACTIVE := false;
        end DISABLE;

    or

      -- kill the task by exiting the select loop
        accept KILL do
          TASK_IS_RUNNING := false;
        end KILL;
```

```
    or

      -- update the local state table with the new global state table
      accept UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE) do
        LOCAL_STATE_TABLE := GLOBAL_STATE_TABLE;
      end UPDATE;

    else

        if ACTIVE then
          -- *** display reason for shutdown ***
          -- *** confirm shutdown or goto selftest operation ***
          USER_INTERFACE.UNHIGHLIGHT(SHUTDOWN);
          SELFTEST_OPERATIONS.ENABLE;
        end if;

    end select;
  end loop;
  text_io.put_line( "... SHUTDOWN_OPERATIONS Task is now terminating." );

  exception
    when others =>
        text_io.put_line( "*** Exception occured in SHUTDOWN_OPERATIONS." );
end SHUTDOWN_OPERATIONS;

--------------------------------------------------------------------------

task body DEGRADED_OPERATIONS is
--
-- ENTRIES:
-- The DEGRADED_OPERATIONS can be entered from NORMAL_OPERATIONS,
-- EMERGENCY_OPERATIONS, and STARTUP_OPERATIONS.
--
-- EXITS:
-- When the boiler is operating in the DEGRADED_OPERATIONS if the
-- water content measuring device fails the boiler goes into
-- EMERGENCY_OPERATIONS. If all of the failed devices are reported
-- repaired the boiler goes into NORMAL_OPERATIONS. If the water content
-- measuring device reports that the water level has exceeded the limits,
-- the boiler goes into SHUTDOWN_OPERATIONS.
--
  ACTIVE : boolean := false;
  TASK_IS_RUNNING : boolean := true;
  LOCAL_STATE_TABLE : BOILER_STATE;
begin
```

```
while TASK_IS_RUNNING loop
  select

    -- when starting this task set the active flag true and
    -- highlight the "DEGRADED" mode on the user interface.
       accept ENABLE do
       ACTIVE := true;
       USER_INTERFACE.HIGHLIGHT(DEGRADED);
       end ENABLE;

  or

       accept DISABLE do
         ACTIVE := false;
       end DISABLE;

  or

    -- kill the task by exiting the select loop
       accept KILL do
         TASK_IS_RUNNING := false;
       end KILL;

  or

    -- update the local state table with the new global state table
    accept UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE) do
      LOCAL_STATE_TABLE := GLOBAL_STATE_TABLE;
    end UPDATE;

  else

    if ACTIVE and (LOCAL_STATE_TABLE.LEVEL_DEVICE /= WORKING) then
      USER_INTERFACE.UNHIGHLIGHT(DEGRADED);
        EMERGENCY_OPERATIONS.ENABLE;
    end if;

    if ACTIVE and (LOCAL_STATE_TABLE.LEVEL >
MAXIMUM_OPERATING_LEVEL) then
      USER_INTERFACE.UNHIGHLIGHT(DEGRADED);
        SHUTDOWN_OPERATIONS.ENABLE;
    end if;

  end select;
end loop;
```

```
  text_io.put_line( "... DEGRADED_OPERATIONS Task is now terminating." );

  exception
    when others =>
        text_io.put_line( "*** Exception occured in DEGRADED_OPERATIONS." );
end DEGRADED_OPERATIONS;

-------------------------------------------------------------------------

procedure CALCULATE_WATER_LEVEL(STATE_TABLE : in out BOILER_STATE)
is
begin
   null;
end CALCULATE_WATER_LEVEL;

-------------------------------------------------------------------------

task body EMERGENCY_OPERATIONS is
--
--      During the EMERGENCY_OPERATIONS calculations are made to predict
-- the worst possible behavior from all system devices.  Using this
-- information it attempts to operate the boiler within the safe limits
-- specified in the boiler specification.
--
-- ENTRIES:
-- EMERGENCY_OPERATIONS can be entered from the NORMAL_OPERATIONS
or
-- DEGRADED_OPERATIONS.
--
-- EXITS:
-- The boiler can exit EMERGENCY_OPERATIONS and enter the
-- DEGRADED_OPERATIONS when the water content measuring device becomes
-- available.  If the water level start to exceed the limits of safe
-- opertion, or the water content calculations are not possible, the
-- boiler then moves into the SHUTDOWN_OPERATIONS.
SHUTDOWN_OPERATIONS
-- is also entered if the steam rate device fails.
--
--
  ACTIVE : boolean := false;
  TASK_IS_RUNNING : boolean := true;
  LOCAL_STATE_TABLE : BOILER_STATE;
begin
  while TASK_IS_RUNNING loop
    select
```

```
  -- when starting this task set the active flag true and
  -- highlight the "EMERGENCY" mode on the user interface.
     accept ENABLE do
     ACTIVE := true;
     USER_INTERFACE.HIGHLIGHT(EMERGENCY);
     end ENABLE;

or

     accept DISABLE do
       ACTIVE := false;
     end DISABLE;

or

  -- kill the task by exiting the select loop
     accept KILL do
       TASK_IS_RUNNING := false;
     end KILL;

or

  -- update the local state table with the new global state table
  accept UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE) do
    LOCAL_STATE_TABLE := GLOBAL_STATE_TABLE;
  end UPDATE;

else

  -- if the water content device has failed calculate the level
  if ACTIVE and (LOCAL_STATE_TABLE.LEVEL_DEVICE /= WORKING) then
    CALCULATE_WATER_LEVEL(LOCAL_STATE_TABLE);

    -- if calculating not possible start SHUTDOWN_OPERATIONS
    if (LOCAL_STATE_TABLE.LEVEL = 0) then
      USER_INTERFACE.UNHIGHLIGHT(EMERGENCY);
        SHUTDOWN_OPERATIONS.ENABLE;
    end if;
  end if;

     if ACTIVE and (LOCAL_STATE_TABLE.RATE_DEVICE /= WORKING) then
    USER_INTERFACE.UNHIGHLIGHT(EMERGENCY);
      SHUTDOWN_OPERATIONS.ENABLE;
  end if;
```

```
    end select;
  end loop;
  text_io.put_line( "... EMERGENCY_OPERATIONS Task is now terminating." );

  exception
    when others =>
        text_io.put_line( "*** Exception occured in EMERGENCY_OPERATIONS." );
end EMERGENCY_OPERATIONS;

-------------------------------------------------------------------------

task body SELFTEST_OPERATIONS is
--
--     The SELFTEST_OPERATIONS first ensures that the communication links
-- are all in proper working order. It tests all of the hardware connected
-- to the boiler to ensure all of the devices are in working order. If
-- any of the hardware fails a message is displayed to the operator
-- describing the device and how it failed. Control is then given to the
-- operator.
--
-- ENTRIES:
-- The SELFTEST_OPERATIONS can be entered from the
SHUTDOWN_OPERATIONS, or
-- by operator command from the keyboard.
--
-- EXITS:
-- If all of the hardware devices passes the test, the boiler enters the
-- STARTUP_OPERATIONS.
--
  ACTIVE : boolean := false;
  TASK_IS_RUNNING : boolean := true;
  LOCAL_STATE_TABLE : BOILER_STATE;
begin
  while TASK_IS_RUNNING loop
    select

      -- when starting this task set the active flag true and
      -- highlight the "SELFTEST" mode on the user interface.
        accept ENABLE do
        ACTIVE := true;
        USER_INTERFACE.HIGHLIGHT(SELFTEST);
        end ENABLE;

    or
```

```
        accept DISABLE do
          ACTIVE := false;
        end DISABLE;

    or

      -- kill the task by exiting the select loop
        accept KILL do
          TASK_IS_RUNNING := false;
        end KILL;

    or

      -- update the local state table with the new global state table
      accept UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE) do
        LOCAL_STATE_TABLE := GLOBAL_STATE_TABLE;
      end UPDATE;

    else

      if ACTIVE then
        -- *** test the monitor ***
        -- *** test the keyboard ***
        -- *** test the comm port ***
        USER_INTERFACE.UNHIGHLIGHT(SELFTEST);
          STARTUP_OPERATIONS.ENABLE;
      end if;

    end select;
  end loop;
  text_io.put_line( "... SELFTEST_OPERATIONS Task is now terminating." );

  exception
    when others =>
        text_io.put_line( "*** Exception occured in SELFTEST_OPERATIONS." );
end SELFTEST_OPERATIONS;

-------------------------------------------------------------------------

task body NORMAL_OPERATIONS is
--
--      In NORMAL_OPERATIONS the water level is kept between 65,000 and
-- 85,000 pounds.  It is maintained at this level by switching on and off
-- the feed pumps.
```

```
--
-- ENTRIES:
-- The NORMAL_OPERATIONS can be entered from the STARTUP_OPERATIONS
or
-- the DEGRADED_OPERATIONS.
--
-- EXITS:
-- If the water content measuring device fails the boiler goes into
-- EMERGENCY_OPERATIONS. If any other device fails the boiler goes into
-- DEGRADED_OPERATIONS. If the water exceeds the limits of the safe
-- operation of the boiler, the boiler goes into the SHUTDOWN_OPERATIONS.
--
  ACTIVE, PUMPS_ARE_ON : boolean := false;
  TASK_IS_RUNNING : boolean := true;
  LOCAL_STATE_TABLE : BOILER_STATE;
begin
  while TASK_IS_RUNNING loop
    select

      -- when starting this task set the active flag true and
      -- highlight the "NORMAL" mode on the user interface.
         accept ENABLE do
        ACTIVE := true;
        USER_INTERFACE.HIGHLIGHT(NORMAL);
         end ENABLE;

    or

         accept DISABLE do
           ACTIVE := false;
         end DISABLE;

    or

      -- update the local state table with the new global state table
         accept UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE) do
        LOCAL_STATE_TABLE := GLOBAL_STATE_TABLE;
      end UPDATE;

    or

      -- kill the task by exiting the select loop
         accept KILL do
           TASK_IS_RUNNING := false;
         end KILL;
```

```
    else

      -- when not rendezvouing with other tasks execute this part
      -- if the normal task is active.
      if ACTIVE then

        -- if the level device is not unhighlight the NORMAL mode
        -- and go into the emergency mode.
        if (LOCAL_STATE_TABLE.LEVEL_DEVICE /= WORKING) then
          USER_INTERFACE.UNHIGHLIGHT(NORMAL);
            EMERGENCY_OPERATIONS.ENABLE;
        end if;

        -- if the pumps have not already been started and the water
        -- level in the boiler is below the minimum operating level
        -- then turn all of the pumps on.
        if (PUMPS_ARE_ON = false) and
          (LOCAL_STATE_TABLE.LEVEL < MINIMUM_OPERATING_LEVEL)
then
          PUMPS_ARE_ON := true;
          TRANSMIT.TURN_ON_ALL_PUMPS;
        end if;

        -- if the pumps are on and the water level has risen above
        -- the minimum operating level then turn all of the pumps off
        if PUMPS_ARE_ON and
          (LOCAL_STATE_TABLE.LEVEL > MINIMUM_OPERATING_LEVEL)
then
          PUMPS_ARE_ON := false;
          TRANSMIT.TURN_OFF_ALL_PUMPS;
        end if;

        if (LOCAL_STATE_TABLE.LEVEL > MAXIMUM_OPERATING_LEVEL)
then
          USER_INTERFACE.UNHIGHLIGHT(NORMAL);
            SHUTDOWN_OPERATIONS.ENABLE;
        end if;

        if (LOCAL_STATE_TABLE.RATE_DEVICE /= WORKING) then
          USER_INTERFACE.UNHIGHLIGHT(NORMAL);
            SHUTDOWN_OPERATIONS.ENABLE;
        end if;

      end if;
```

```
    end select;
  end loop;
  text_io.put_line( "... NORMAL_OPERATIONS Task is now terminating." );

  exception
    when others =>
        text_io.put_line( "*** Exception occured in NORMAL_OPERATIONS." );
end NORMAL_OPERATIONS;

-------------------------------------------------------------------------

end OPERATIONAL_MODES;
```

```
-- ***************************************************************************
--
--                      GENERIC QUEUE
--
--          This packages is to be used for generic instantiations
--      of simple queues of different types.
--
--      written by: Jerry D. Cavin          date: March 18th, 1993
--
-- ***************************************************************************

generic
  MAX : positive;
  type ITEM is private;

package QUEUE is
  function FULL return boolean;
  function EMPTY return boolean;
  function GET return ITEM;
  procedure PUT(JOB : in ITEM);
  procedure INITIALIZE;
end QUEUE;

package body QUEUE is

  type ARRAY_OF_JOBS is array(1..MAX) of ITEM;

  type QUE is
    record
        DATA : ARRAY_OF_JOBS;
        FIRST, LAST : integer range 1..MAX;
        SIZE : integer range 0..MAX;
    end record;

  Q : QUE;

-------------------------------------------------------------------------

function FULL return boolean is
begin
  return Q.SIZE = MAX;
end FULL;

-------------------------------------------------------------------------
```

```
function EMPTY return boolean is
begin
  return Q.SIZE = 0;
end EMPTY;

-------------------------------------------------------------------------

procedure PUT(JOB : in ITEM) is
begin
  if FULL then
    raise CONSTRAINT_ERROR;
  end if;
  Q.DATA(Q.LAST) := JOB;
  Q.LAST := Q.LAST mod MAX + 1;
  Q.SIZE := Q.SIZE + 1;
end PUT;

-------------------------------------------------------------------------

function GET return ITEM is
  JOB : ITEM;
begin
  if EMPTY then
    raise CONSTRAINT_ERROR;
  end if;
  JOB := Q.DATA(Q.FIRST);
  Q.FIRST := Q.FIRST mod MAX + 1;
  Q.SIZE := Q.SIZE - 1;
  return JOB;
end GET;

-------------------------------------------------------------------------

procedure INITIALIZE is
begin
  Q.FIRST := 1;
  Q.LAST := 1;
  Q.SIZE := 0;
end INITIALIZE;

-------------------------------------------------------------------------

end QUEUE;
```

```
--*********************************************************************
--
--              RX (RECEIVED) COMMUNICATIONS Package
--
--
--    written by: Jerry D. Cavin          date: April 3rd, 1993
--
--*********************************************************************

with text_io;
with interrupt;              -- DOS call package
with time;                   -- DOS time function

with bit_ops;                -- bit manipulation package
use bit_ops;

with SYSTEM_PARAMETERS;      use SYSTEM_PARAMETERS;
with OPERATIONAL_MODES;      use OPERATIONAL_MODES;
with UI;                     use UI;
with TX;                     use TX;

package RX is

  task RECEIVE is
    entry ENABLE(COMM : integer);
    entry DISABLE;
    entry KILL;
  end RECEIVE;

end RX;

package body RX is

  GLOBAL_STATE_TABLE : BOILER_STATE :=

  ( true,                    -- OK_TO_TRANSMIT
    SELFTEST,                    -- MODE
    false,                   -- START
    0,                       -- INSTRUMENT_TIME
    WORKING,                     -- COMPUTER

    0.0,                     -- LEVEL_CAL
    0,                       -- LEVEL
    WORKING,                     -- LEVEL_DEVICE
```

```
    0,                          -- RATE
    WORKING,                    -- RATE_DEVICE

    (OFF, OFF, OFF, OFF),       -- PUMP
    (WORKING, WORKING, WORKING, WORKING),    -- PUMP_DEVICE

    (NO_FLOW, NO_FLOW, NO_FLOW, NO_FLOW),    -- MONITOR
    (WORKING, WORKING, WORKING, WORKING) );  -- MONITOR_DEVICE

-------------------------------------------------------------------------

procedure CLEAR_BUFFER(BUFFER : in out string) is
begin
   for INDEX in BUFFER'range loop
        BUFFER(INDEX) := ' ';
   end loop;
end CLEAR_BUFFER;


-------------------------------------------------------------------------

function BIT(I : integer; NUM : integer) return boolean is
--
--
--
begin
   case NUM is
     when 0 =>  return (I and 16#0001#) > 0;
     when 1 =>  return (I and 16#0002#) > 0;
     when 2 =>  return (I and 16#0004#) > 0;
     when 3 =>  return (I and 16#0008#) > 0;
     when 4 =>  return (I and 16#0010#) > 0;
     when 5 =>  return (I and 16#0020#) > 0;
     when 6 =>  return (I and 16#0040#) > 0;
     when 7 =>  return (I and 16#0080#) > 0;
     when 8 =>  return (I and 16#0100#) > 0;
     when 9 =>  return (I and 16#0200#) > 0;
     when 10 => return (I and 16#0400#) > 0;
     when 11 => return (I and 16#0800#) > 0;
     when 12 => return (I and 16#1000#) > 0;
     when 13 => return (I and 16#2000#) > 0;
     when 14 => return (I and 16#4000#) > 0;
     when 15 => return (I < 0);
     when others => return false;
   end case;
```

```
end BIT;

-------------------------------------------------------------------------

function RX_READY(COMM_PORT : integer) return boolean is
--
-- returns true when the serial port specified is ready to receive,
-- false, if it is not ready to receive.
--
   REGS : interrupt.registers;
begin
   REGS.AX := 16#0300#;
   REGS.DX := COMM_PORT-1;
   interrupt.vector( on => 16#14#, register_block => REGS );
   if BIT(REGS.AX, 8) then
         return true;
   end if;
   return false;
end RX_READY;

-------------------------------------------------------------------------

function RS232_READ(COMM_PORT : integer) return character is
--
--
--
REGS : interrupt.registers;
   MASK : integer := 16#0100#;
   CHAR_MASK : integer := 16#00ff#;
   CH : character;
begin
   REGS.AX := 16#0200#;
   REGS.DX := COMM_PORT-1;
   interrupt.vector( on => 16#14#, register_block => REGS );
   if BIT(REGS.AX, 15) then
         return '?';
   end if;

   MASK := REGS.AX and CHAR_MASK;
   return character'val( MASK );
end RS232_READ;

-------------------------------------------------------------------------

procedure RS232_INITIALIZE(COMM_PORT : integer) is
```

```
--
--
--
   REGS : interrupt.registers;
begin
   REGS.AX := 16#00e3#;    -- 9600 baud, no parity, 1 stop bit, 8 bits
   REGS.DX := COMM_PORT-1;
   interrupt.vector( on => 16#14#, register_block => REGS );
   USER_INTERFACE.DISPLAY( "Result Code="&integer'image(REGS.AX) );

   if BIT(REGS.AX, 15) then
     USER_INTERFACE.DISPLAY( "*** INIT: Timed Out." );
   end if;

   if BIT(REGS.AX, 14) then
     USER_INTERFACE.DISPLAY( "*** INIT: TX Shift Register Empty.");
   end if;

   if BIT(REGS.AX, 13) then
     USER_INTERFACE.DISPLAY("*** INIT: TX Hold Register Empty.");
   end if;

   if BIT(REGS.AX, 12) then
     USER_INTERFACE.DISPLAY("*** INIT: Break Detected.");
   end if;

   if BIT(REGS.AX, 11) then
     USER_INTERFACE.DISPLAY("*** INIT: Framing Error.");
   end if;

   if BIT(REGS.AX, 10) then
     USER_INTERFACE.DISPLAY("*** INIT: Parity Error.");
   end if;

   if BIT(REGS.AX, 9) then
     USER_INTERFACE.DISPLAY("*** INIT: Overrun Error.");
   end if;

   if BIT(REGS.AX, 8) then
     USER_INTERFACE.DISPLAY("*** INIT: Data Ready.");
   end if;

   if BIT(REGS.AX, 7) then
     USER_INTERFACE.DISPLAY("*** INIT: Receive Line Signal Detected.");
   end if;
```

```
   if BIT(REGS.AX, 6) then
     USER_INTERFACE.DISPLAY("*** INIT: Ring Indicator.");
   end if;

   if BIT(REGS.AX, 5) then
     USER_INTERFACE.DISPLAY("*** INIT: Data-Set Ready.");
   end if;

   if BIT(REGS.AX, 4) then
     USER_INTERFACE.DISPLAY("*** INIT: Clear to Send.");
   end if;

   if BIT(REGS.AX, 3) then
     USER_INTERFACE.DISPLAY("*** INIT: Change in Receive Line Signal
Detected.");
   end if;

   if BIT(REGS.AX, 2) then
     USER_INTERFACE.DISPLAY("*** INIT: Trailing Edge Ring Indicator.");
   end if;

   if BIT(REGS.AX, 1) then
     USER_INTERFACE.DISPLAY("*** INIT: Change in Data-Set Ready Status.");
   end if;

   if BIT(REGS.AX, 0) then
     USER_INTERFACE.DISPLAY("*** INIT: Change in Clear-to-Send Status.");
   end if;

end RS232_INITIALIZE;

-------------------------------------------------------------------------

function IS_IN(IS_THIS_STRING : string;
               IN_THIS_STRING : string) return boolean is
   INDEX : integer;
begin
   for OUTER_INDEX in IN_THIS_STRING'range loop
         INDEX := OUTER_INDEX;
         for INNER_LOOP in IS_THIS_STRING'range loop
            exit when IS_THIS_STRING(INNER_LOOP) /=
IN_THIS_STRING(INDEX);
            if INNER_LOOP = IS_THIS_STRING'length then
                  return true;
```

```
            end if;
            INDEX := INDEX + 1;
         end loop;
   end loop;
   return false;
end IS_IN;

--------------------------------------------------------------------------

function IDENTIFY_MESSAGE(MESSAGE : in MESSAGE_BUFFER) return
                              MESSAGES_RECEIVED is
begin
   if IS_IN("SYNC", MESSAGE) then
         return SYNC;
   end if;
   if IS_IN("SHUTNOW", MESSAGE) then
         return SHUTNOW;
   end if;
   if IS_IN("START", MESSAGE) then
         return START;
   end if;
   if IS_IN("BOILSTDBY", MESSAGE) then
         return BOILSTDBY;
   end if;
   if IS_IN("TSTMSG0", MESSAGE) then
         return TSTMSG0;
   end if;
   if IS_IN("TSTMSG1", MESSAGE) then
         return TSTMSG1;
   end if;
   if IS_IN("TSTMSG2", MESSAGE) then
         return TSTMSG2;
   end if;
   if IS_IN("TSTMSG3", MESSAGE) then
         return TSTMSG3;
   end if;
   if IS_IN("TSTMSG4", MESSAGE) then
         return TSTMSG4;
   end if;
   if IS_IN("TSTMSG5", MESSAGE) then
         return TSTMSG5;
   end if;
   if IS_IN("BOILEVADJ", MESSAGE) then
         return BOILEVADJ;
   end if;
```

```
if IS_IN("PUMPINDON", MESSAGE) then
      return PUMPINDON;
end if;
if IS_IN("PUMPINDOFF", MESSAGE) then
      return PUMPINDOFF;
end if;
if IS_IN("WATFLOWON", MESSAGE) then
      return WATFLOWON;
end if;
if IS_IN("WATFLOWOFF", MESSAGE) then
      return WATFLOWOFF;
end if;
if IS_IN("LEVCONST", MESSAGE) then
      return LEVCONST;
end if;
if IS_IN("WATERLEVEL", MESSAGE) then
      return WATERLEVEL;
end if;
if IS_IN("STEAMRATE", MESSAGE) then
      return STEAMRATE;
end if;
if IS_IN("PUMPOK", MESSAGE) then
      return PUMPOK;
end if;
if IS_IN("WATFLOWOK", MESSAGE) then
      return WATFLOWOK;
end if;
if IS_IN("WATLEVOK", MESSAGE) then
      return WATLEVOK;
end if;
if IS_IN("STMRATOK", MESSAGE) then
      return STMRATOK;
end if;
if IS_IN("PUMPUS", MESSAGE) then
      return PUMPUS;
end if;
if IS_IN("WATFLOWUS", MESSAGE) then
      return WATFLOWUS;
end if;
if IS_IN("WATLEVUS", MESSAGE) then
      return WATLEVUS;
end if;
if IS_IN("STMRATUS", MESSAGE) then
      return STMRATUS;
end if;
```

```
   return UNKNOWN;
end IDENTIFY_MESSAGE;

-------------------------------------------------------------------------

procedure STRIP(CH : character; BUFFER : in out MESSAGE_BUFFER) is
begin
   for INDEX in BUFFER'range loop
         if BUFFER(INDEX) = CH then
            BUFFER(INDEX) := ' ';
         end if;
   end loop;
end STRIP;

-------------------------------------------------------------------------

procedure FILL( FROM_INDEX, TO_INDEX : integer;
                BUFFER : in out MESSAGE_BUFFER ) is
--
--
--
begin
   for INDEX in FROM_INDEX..TO_INDEX loop
         if (INDEX > 0) and (INDEX < MESSAGE_BUFFER'length) then
            BUFFER(INDEX) := ' ';
         end if;
   end loop;
end FILL;

-------------------------------------------------------------------------

procedure PROCESS_MESSAGE(MESSAGE : in out MESSAGE_BUFFER;
                          STATE_TABLE : in out BOILER_STATE) is
--
--
--
   PUMP : integer;
begin
   STRIP(ASCII.STX, MESSAGE);
   STRIP(ASCII.ETX, MESSAGE);

   case IDENTIFY_MESSAGE(MESSAGE) is
         when SYNC =>
            FILL(1, 10, MESSAGE);  -- remove everything but the numbers
            STATE_TABLE.INSTRUMENT_TIME := long_integer'value(MESSAGE);
```

```
USER_INTERFACE.DISPLAY("Time:"&long_integer'image(STATE_TABLE.INSTRU
MENT_TIME));

        when SHUTNOW => null;
        when START => STATE_TABLE.START := true;
        when BOILSTDBY => null;
        when TSTMSG0 => null;
        when TSTMSG1 => null;
        when TSTMSG2 => null;
        when TSTMSG3 => null;
        when TSTMSG4 => null;
        when TSTMSG5 => null;
        when BOILEVADJ => null;

        when PUMPINDON =>
           FILL(1, 10, MESSAGE);  -- remove everything but the numbers
           PUMP := integer'value(MESSAGE);
           if (PUMP >= 1) and (PUMP <= NUMBER_OF_PUMPS) then
                STATE_TABLE.PUMP(PUMP) := ON;
           end if;

        when PUMPINDOFF =>
           FILL(1, 10, MESSAGE);  -- remove everything but the numbers
           PUMP := integer'value(MESSAGE);
           if (PUMP >= 1) and (PUMP <= NUMBER_OF_PUMPS) then
                STATE_TABLE.PUMP(PUMP) := OFF;
           end if;

        when WATFLOWON =>
           FILL(1, 10, MESSAGE);  -- remove everything but the numbers
           PUMP := integer'value(MESSAGE);
           if (PUMP >= 1) and (PUMP <= NUMBER_OF_PUMPS) then
                STATE_TABLE.MONITOR(PUMP) := FLOW;
           end if;

        when WATFLOWOFF =>
           FILL(1, 10, MESSAGE);  -- remove everything but the numbers
           PUMP := integer'value(MESSAGE);
           if (PUMP >= 1) and (PUMP <= NUMBER_OF_PUMPS) then
           STATE_TABLE.MONITOR(PUMP) := NO_FLOW;
           end if;

        when LEVCONST =>
           FILL(1, 10, MESSAGE);  -- remove everything but the numbers
```

```
--       STATE_TABLE.LEVEL_CAL := integer'image(MESSAGE);

         when WATERLEVEL =>
            FILL(1, 10, MESSAGE);  -- remove everything but the numbers
--       STATE_TABLE.LEVEL := integer'image(MESSAGE);

         when STEAMRATE =>
            FILL(1, 10, MESSAGE);  -- remove everything but the numbers
--       STATE_TABLE.RATE := integer'image(MESSAGE);

         when PUMPOK =>
            FILL(1, 10, MESSAGE);  -- remove everything but the numbers
            PUMP := integer'value(MESSAGE);
            if (PUMP >= 1) and (PUMP <= NUMBER_OF_PUMPS) then
                 STATE_TABLE.PUMP_DEVICE(PUMP) := WORKING;
            end if;

         when WATFLOWOK =>
            FILL(1, 10, MESSAGE);  -- remove everything but the numbers
            PUMP := integer'value(MESSAGE);
            if (PUMP >= 1) and (PUMP <= NUMBER_OF_PUMPS) then
                 STATE_TABLE.MONITOR_DEVICE(PUMP) := WORKING;
            end if;

         when WATLEVOK =>
            STATE_TABLE.LEVEL_DEVICE := WORKING;

         when STMRATOK =>
            STATE_TABLE.RATE_DEVICE := WORKING;

         when PUMPUS =>
            FILL(1, 10, MESSAGE);  -- remove everything but the numbers
            PUMP := integer'value(MESSAGE);
            if (PUMP >= 1) and (PUMP <= NUMBER_OF_PUMPS) then
                 STATE_TABLE.PUMP_DEVICE(PUMP) := REPAIR_ACK;
            end if;

         when WATFLOWUS =>
            FILL(1, 10, MESSAGE);  -- remove everything but the numbers
            PUMP := integer'value(MESSAGE);
            if (PUMP >= 1) and (PUMP <= NUMBER_OF_PUMPS) then
                 STATE_TABLE.MONITOR_DEVICE(PUMP) :=
                   REPAIR_ACK;
            end if;
```

```
            when WATLEVUS =>
               STATE_TABLE.LEVEL_DEVICE := REPAIR_ACK;

            when STMRATUS =>
               STATE_TABLE.RATE_DEVICE := REPAIR_ACK;

            when others => null;
      end case;

      -- clear the message buffer of any message remains
      for INDEX in MESSAGE'range loop
            MESSAGE(INDEX) := ' ';
      end loop;
   end PROCESS_MESSAGE;

   ------------------------------------------------------------------------

   function TIME_STAMP return float is
   --
   -- calculates a unique time stamp
   --
     HOURS   : time.hours_range;
     MINUTES : time.minutes_range;
     SECONDS : time.seconds_range;
     HUNDREDTHS : time.hundredths_range;
     STAMP : FLOAT := 0.0;
   begin
     time.get( HOURS, MINUTES, SECONDS, HUNDREDTHS);
     STAMP := float(HOURS) * 3600.0;
     STAMP := STAMP + float(MINUTES) * 60.0;
     STAMP := STAMP + float(SECONDS);
     STAMP := STAMP + float(HUNDREDTHS)/100.0;
     return STAMP;
   end TIME_STAMP;

   ------------------------------------------------------------------------

   task body RECEIVE is
      MESSAGE : MESSAGE_BUFFER;
      COMM_PORT : integer := 1;
      COMMUNICATIONS_ENABLED : boolean := false;
      TASK_IS_RUNNING : boolean := true;
      START_OF_MESSAGE : float := 0.0;
      LINE : integer := 0;
   begin
```

```
LINE := 1;
while TASK_IS_RUNNING loop
  select

      accept ENABLE(COMM : integer) do
        LINE := 2;
        COMM_PORT := COMM;
        RS232_INITIALIZE(COMM_PORT);
        COMMUNICATIONS_ENABLED := true;
        LINE := 3;
      end ENABLE;

  or

      accept KILL do
        LINE := 4;
        TASK_IS_RUNNING := false;
      end KILL;

  or
      accept DISABLE do
        LINE := 5;
        COMMUNICATIONS_ENABLED := false;
      end DISABLE;

  else

      LINE := 6;
      if COMMUNICATIONS_ENABLED and RX_READY(COMM_PORT) then

        GLOBAL_STATE_TABLE.OK_TO_TRANSMIT := false;
        LINE := 7;
        TRANSMIT.UPDATE(GLOBAL_STATE_TABLE);
        LINE := 8;
        START_OF_MESSAGE := TIME_STAMP;
        LINE := 9;
        for INDEX in 1..MESSAGE_SIZE loop
          exit when RX_READY(COMM_PORT) = false;
          LINE := 10;
          MESSAGE(INDEX) := RS232_READ(COMM_PORT);
          -- loop until we get a valid printable character, or until
          -- ready becomes false
          while (MESSAGE(INDEX) < ' ') and (MESSAGE(INDEX) > '~') loop
              exit when RX_READY(COMM_PORT) = false;
              MESSAGE(INDEX) := RS232_READ(COMM_PORT);
```

```
            end loop;
          end loop;
          LINE := 11;
          USER_INTERFACE.UPDATE_RX(MESSAGE);
          LINE := 12;
          PROCESS_MESSAGE(MESSAGE, GLOBAL_STATE_TABLE);
          LINE := 13;
          while (TIME_STAMP-RECEIVE_WINDOW < START_OF_MESSAGE) loop
            null;  -- delay until reception window has expired
          end loop;
          LINE := 14;
          GLOBAL_STATE_TABLE.OK_TO_TRANSMIT := true;
        end if;

        LINE := 15;
        if COMMUNICATIONS_ENABLED then
          TRANSMIT.UPDATE(GLOBAL_STATE_TABLE);
          STARTUP_OPERATIONS.UPDATE(GLOBAL_STATE_TABLE);
          SHUTDOWN_OPERATIONS.UPDATE(GLOBAL_STATE_TABLE);
          DEGRADED_OPERATIONS.UPDATE(GLOBAL_STATE_TABLE);
          EMERGENCY_OPERATIONS.UPDATE(GLOBAL_STATE_TABLE);
          NORMAL_OPERATIONS.UPDATE(GLOBAL_STATE_TABLE);
          SELFTEST_OPERATIONS.UPDATE(GLOBAL_STATE_TABLE);
        end if;
        LINE := 16;

    end select;
  end loop;
  text_io.put_line( "... RECEIVE Task is now terminating." );

  exception
    when others =>
        text_io.put_line( "**** EXCEPTION occured in RECEIVE @ Line " &
                          integer'image(LINE) );
end RECEIVE;

-------------------------------------------------------------------------

end RX;
```

```
--*****************************************************************
--                                    **
--              SYSTEM PARAMETERS                    **
--       This package contains all of the predefined system      **
--    used inthe boiler control program.                **
--                                    **
--    written by: Jerry D. Cavin          date: April 3rd, 1993   **
--
--*****************************************************************

package SYSTEM_PARAMETERS is

   -- ************** Boiler Parameters (Superhot MK1) *****************

   BOILER_TOTAL_CAPACITY    : constant long_integer := 130_000;  -- pounds
   BOILER_MAXIMUM_SAFE      : constant long_integer := 110_000;  -- pounds
   BOILER_MINIMUM_SAFE      : constant long_integer := 23_000;   -- pounds
   MAXIMUM_OPERATING_LEVEL  : constant long_integer := 60_000;   -- pounds
   MINIMUM_OPERATING_LEVEL  : constant long_integer := 40_000;   -- pounds

   MAXIMUM_BOILER_RATE      : constant long_integer := 700_000;  -- lb/hr
   MAXIMUM_BOILER_INCREASE  : constant long_integer := 4_200_000; -- lb/hr/hr
   MAXIMUM_BOILER_DECREASE  : constant long_integer := 84_000_000; --
lb/hr/hr

   -- ************* Boiler Feed Pumps (Forcefeed Type 2) ***************

   MAXIMUM_FEED_PUMP_OUTPUT     : constant long_integer := 275_000;  -- lb/hr
   FEED_PUMP_OUTPUT_UPPER_LIMIT : constant long_integer := 0;    -- %
   FEED_PUMP_OUTPUT_LOWER_LIMIT : constant long_integer := -5;   -- %
   FEED_PUMP_START_UP_DELAY     : constant long_integer := 6;    -- seconds
   FEED_WATER_TEMPERATURE       : constant long_integer := 18;   -- deg C
   FEED_WATER_TEMPERATURE_UPPER : constant long_integer := 23;   -- deg C
   FEED_WATER_TEMPERATURE_LOWER : constant long_integer := -10;  -- deg C
   FEED_WATER_PRESSURE          : constant long_integer := 5;    -- mPa

   -- ********** Boiler Feed Pump Monitor (Everwatch Type 5a) *************

   WATER_FEED_SET_POINT         : constant long_integer := 425;  -- gal/min
   UPPER_SET_POINT_LIMIT        : constant long_integer := 1;    -- %
   LOWER_SET_POINT_LIMIT        : constant long_integer := -2;   -- %

   -- ******** Boiler Content Measuring Device (Levelmet Type 4) **********

   NOMINAL_CONTENT_CALIBRATION : constant float := 2.5;     -- per 1000 lbs
```

```
   MAXIMUM_CONTENT_CALIBRATION : constant float := 2.88;    -- per 1000 lbs
   MINIMUM_CONTENT_CALIBRATION : constant float := 1.95;    -- per 1000 lbs
   CONTENT_UPPER_LIMIT         : constant float := 0.5;     -- %
   CONTENT_LOWER_LIMIT         : constant float := -0.7;    -- %
   MAXIMUM_CONTENT_READING     : constant float := 360.00;  -- units
   MINIMUM_CONTENT_READING     : constant float := 30.00;   -- units

   -- ************* Steam Rate Device (Steamapp Mk 2, Mod3) ************

   MAXIMUM_RATE_READING   : constant long_integer := 850_000;  -- lbs/hr
   MINIMUM_RATE_READING   : constant long_integer := 0;        -- lbs/hr
   RATE_UPPER_LIMIT       : constant long_integer := 2_000;    -- lbs/hr
   RATE_LOWER_LIMIT       : constant long_integer := -3_000;   -- lbs/hr
   RATE_OUTPUT_RESOLUTION : constant long_integer := 500;      -- lbs/hr

   -- ************************ System Parameters ************************

   NUMBER_OF_PUMPS : constant := 4;       -- total number of pumps/monitors
   RECEIVE_WINDOW  : constant := 1.250;   -- # seconds allowed between
                                          -- receiving/transmission

   -- ****************** System "Type" Definitions **********************

   MESSAGE_SIZE : constant := 20;
   subtype MESSAGE_BUFFER is string(1..MESSAGE_SIZE);

   type DEVICE is (ON, OFF);
   type MONITOR is (FLOW, NO_FLOW);
   type STATE is (WORKING, REPAIR_REQ, REPAIR_ACK, FAILED);

   type COLLECTION_OF_PUMPS is array(1..NUMBER_OF_PUMPS) of DEVICE;
   type COLLECTION_OF_MONITORS is array(1..NUMBER_OF_PUMPS) of
MONITOR;
   type COLLECTION_OF_STATES is array(1..NUMBER_OF_PUMPS) of STATE;
   type OPERATING_MODES is
         (SELFTEST, COMPTEST, NORMAL, DEGRADED, EMERGENCY,
SHUTDOWN);

   --
   -- this is the state table record containing all information
   -- needed to determine the current state of the boiler.  it also
   -- contains information about the current state of the communications
   -- network.
   --
   type BOILER_STATE is
```

```
      record
        OK_TO_TRANSMIT    : boolean;
        MODE              : OPERATING_MODES;
        START             : boolean;
        INSTRUMENT_TIME   : long_integer;
        COMPUTER          : STATE;

        LEVEL_CAL         : float;
        LEVEL             : long_integer;
        LEVEL_DEVICE      : STATE;

        RATE              : long_integer;
        RATE_DEVICE       : STATE;

        PUMP              : COLLECTION_OF_PUMPS;
        PUMP_DEVICE       : COLLECTION_OF_STATES;

        MONITOR           : COLLECTION_OF_MONITORS;
        MONITOR_DEVICE    : COLLECTION_OF_STATES;
    end record;

--
-- the following is the list of messages that can be transmitted
-- by this package.
--
type MESSAGES_TRANSMITTED is
      (STX,       -- start of message tranmission character
       ETX,       -- end of transmission character

       SYNC,      -- syncronization message
       READY,     -- ready to control boiler water content

       COMPTEST,  -- computer test mode
       SELFTEST,  -- system test and initialization
       NORMAL,    -- normal operation mode
       DEGRADED,  -- degraded operation mode
       EMERGENCY, -- emergency operation mode
       SHUTDOWN,  -- shutdown mode

       LEVELCAL,  -- calibration constant query to Levelmet Type 4
       BOILHIGH,  -- boiler content high

       PUMP1ON,   -- turn on feed pump number 1
       PUMP2ON,   -- turn on feed pump number 2
       PUMP3ON,   -- turn on feed pump number 3
```

```
    PUMP4ON,    -- turn on feed pump number 4

    PUMP1OFF,   -- turn off feed pump number 1
    PUMP2OFF,   -- turn off feed pump number 2
    PUMP3OFF,   -- turn off feed pump number 3
    PUMP4OFF,   -- turn off feed pump number 4

    TSTMSG0,    -- test message number 0
    TSTMSG1,    -- test message number 1
    TSTMSG2,    -- test message number 2
    TSTMSG3,    -- test message number 3
    TSTMSG4,    -- test message number 4
    TSTMSG5,    -- test message number 5

    COMPFAIL,   -- computer failure

    PUMPUS1,    -- repair request for pump 1
    PUMPUS2,    -- repair request for pump 2
    PUMPUS3,    -- repair request for pump 3
    PUMPUS4,    -- repair request for pump 4

    WATFLOWUS1, -- feed pump monitor 1 repair request
    WATFLOWUS2, -- feed pump monitor 2 repair request
    WATFLOWUS3, -- feed pump monitor 3 repair request
    WATFLOWUS4, -- feed pump monitor 4 repair request

    WATLEVUS,   -- boiler water content measurement device repair request
    STMRATUS,   -- steaming rate measurement device repair request

    PUMPOK1,    -- pump 1 repaired acknowledgement
    PUMPOK2,    -- pump 2 repaired acknowledgement
    PUMPOK3,    -- pump 3 repaired acknowledgement
    PUMPOK4,    -- pump 4 repaired acknowledgement

    WATFLOWOK1, -- feed pump monitor 1 repaired acknowledgement
    WATFLOWOK2, -- feed pump monitor 2 repaired acknowledgement
    WATFLOWOK3, -- feed pump monitor 3 repaired acknowledgement
    WATFLOWOK4, -- feed pump monitor 4 repaired acknowledgement

    WATLEVOK,   -- boiler content measurement device repaired ack
    STMRATOK ); -- steam rate measurement device repaired ack

--
--
--
```

```
  type MESSAGES_RECEIVED is
       (SYNC,      -- syncronization message
        SHUTNOW,   -- command to shutdown boiler
        START,     -- start controlling water content
        BOILSTDBY, -- boiler on stand by

        TSTMSG0,   -- test message number 0
        TSTMSG1,   -- test message number 1
        TSTMSG2,   -- test message number 2
        TSTMSG3,   -- test message number 3
        TSTMSG4,   -- test message number 4
        TSTMSG5,   -- test message number 5

        BOILEVADJ, -- boiler content adjusted

        PUMPINDON, -- feed pump is on
        PUMPINDOFF, -- feed pump is off
        WATFLOWON, -- feed pump monitor reports water flowing
        WATFLOWOFF, -- feed pump monitor reports water not flowing

        LEVCONST,  -- calibration constant from Levelmt Type 4
        WATERLEVEL, -- water content from Levelmet type 4
        STEAMRATE, -- steaming rate from Steamapp Mk 2, Mod 3

        PUMPOK,    -- pump is repaired
        WATFLOWOK, -- feedpump monitor repaired

        WATLEVOK,  -- boiler water content measuring device repaired
        STMRATOK,  -- steaming rate measurement device repaired

        PUMPUS,    -- pump repair request acknowledgement
        WATFLOWUS, -- feed pump monitor repair request acknowledge

        WATLEVUS,  -- boiler content measurement device repair request ack
        STMRATUS,  -- steam rate measurement device repair request ack

        UNKNOWN);  -- message could not be identified

end SYSTEM_PARAMETERS;

package body SYSTEM_PARAMETERS is
end SYSTEM_PARAMETERS;
```

```
--*****************************************************************
--
--              TX (TRANSMIT)  COMMUNICATIONS Package
--
--
--   written by: Jerry D. Cavin          date: April 3rd, 1993
--
--*****************************************************************

with text_io;
with interrupt;              -- DOS call package
with time;                   -- DOS time function

with bit_ops;                -- bit manipulation package
use bit_ops;

with UI;                     use UI;
with SYSTEM_PARAMETERS;      use SYSTEM_PARAMETERS;

package TX is
  task TRANSMIT is
    entry UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE);
    entry UNTIL_READY;
    entry LEVEL_CAL;
    entry BOILER_HIGH;
    entry PUMP_OK(PUMP : integer);
    entry MONITOR_OK(MONITOR : integer);
    entry TURN_ON_PUMP(PUMP : integer);
    entry TURN_ON_ALL_PUMPS;
    entry TURN_OFF_PUMP(PUMP : integer);
    entry TURN_OFF_ALL_PUMPS;
    entry WATER_CONTENT_OK;
    entry RATE_OK;
    entry CHECK_LEVEL;
    entry CHECK_RATE;
    entry CHECK_PUMP(PUMP : integer);
    entry PUMP_STATUS(PUMP : integer);
    entry STATUS_OF_ALL_PUMPS;
    entry KILL;
    entry ENABLE(COMM : integer);
    entry DISABLE;
    entry CLEAR_TRANSMIT_QUEUE;
  end TRANSMIT;
end TX;
```

```
package body TX is

  MAX_SIZE : constant := 100;

  type TRANSMIT_JOBS is (

     TX_ALL_PUMPS_ON, TX_ALL_PUMPS_OFF, TX_ALL_PUMP_STAT,

     TX_PUMP1ON,    TX_PUMP2ON,    TX_PUMP3ON,    TX_PUMP4ON,
     TX_PUMP1OFF,   TX_PUMP2OFF,   TX_PUMP3OFF,   TX_PUMP4OFF,
     TX_PUMP1OK,    TX_PUMP2OK,    TX_PUMP3OK,    TX_PUMP4OK,
     TX_PUMP1STAT,  TX_PUMP2STAT,  TX_PUMP3STAT,
TX_PUMP4STAT,

     TX_MONITOR1OK,  TX_MONITOR2OK,  TX_MONITOR3OK,
TX_MONITOR4OK,
     TX_MONITOR1STAT, TX_MONITOR2STAT, TX_MONITOR3STAT,
TX_MONITOR4STAT,

     TX_BOILER_HIGH, TX_UNTIL_READY,  TX_LEVEL_CAL,

     TX_LEVEL_OK,   TX_RATEOK,     NO_JOB,
     TX_WATLEVOK,   TX_LEVELCAL
  );



  type ARRAY_OF_JOBS is array(1..MAX_SIZE) of TRANSMIT_JOBS;
  type QUEUE is
    record
      DATA : ARRAY_OF_JOBS;
      FIRST, LAST : integer range 1..MAX_SIZE;
      SIZE : integer range 0..MAX_SIZE;
    end record;

  COMM_PORT : integer := 1;
  SYNC_COUNT : long_integer := 0;

--------------------------------------------------------------------------

function FULL(Q : QUEUE) return boolean is
begin
  return Q.SIZE = MAX_SIZE;
end FULL;
```

```
--------------------------------------------------------------------------

function EMPTY(Q : QUEUE) return boolean is
begin
  return Q.SIZE = 0;
end EMPTY;

--------------------------------------------------------------------------

procedure ADD(Q : in out QUEUE; JOB : in TRANSMIT_JOBS) is
begin
  if FULL(Q) then
    -- **** DISPLAY TRANSMIT QUEUE IS FULL ***
    return;
  end if;
  Q.DATA(Q.LAST) := JOB;
  Q.LAST := Q.LAST mod MAX_SIZE + 1;
  Q.SIZE := Q.SIZE + 1;
end ADD;

--------------------------------------------------------------------------

procedure FIRST(Q : in out QUEUE; JOB : out TRANSMIT_JOBS) is
begin
  if EMPTY(Q) then
     JOB := NO_JOB;
     return;
  end if;
  JOB := Q.DATA(Q.FIRST);
  Q.FIRST := Q.FIRST mod MAX_SIZE + 1;
  Q.SIZE := Q.SIZE - 1;
end FIRST;

--------------------------------------------------------------------------

procedure INITIALIZE(Q : in out QUEUE) is
begin
  Q.FIRST := 1;
  Q.LAST := 1;
  Q.SIZE := 0;
end INITIALIZE;

--------------------------------------------------------------------------

function LENGTH(STR : string) return integer is
```

```
--
--
--
begin
   return STR'length;
end LENGTH;

-------------------------------------------------------------------------

procedure CLEAR_BUFFER(BUFFER : in out string) is
begin
   for INDEX in BUFFER'range loop
     BUFFER(INDEX) := ' ';
   end loop;
end CLEAR_BUFFER;


-------------------------------------------------------------------------

function BIT(I : integer; NUM : integer) return boolean is
--
--
--
begin
   case NUM is
      when 0 =>  return (I and 16#0001#) > 0;
      when 1 =>  return (I and 16#0002#) > 0;
      when 2 =>  return (I and 16#0004#) > 0;
      when 3 =>  return (I and 16#0008#) > 0;
      when 4 =>  return (I and 16#0010#) > 0;
      when 5 =>  return (I and 16#0020#) > 0;
      when 6 =>  return (I and 16#0040#) > 0;
      when 7 =>  return (I and 16#0080#) > 0;
      when 8 =>  return (I and 16#0100#) > 0;
      when 9 =>  return (I and 16#0200#) > 0;
      when 10 => return (I and 16#0400#) > 0;
      when 11 => return (I and 16#0800#) > 0;
      when 12 => return (I and 16#1000#) > 0;
      when 13 => return (I and 16#2000#) > 0;
      when 14 => return (I and 16#4000#) > 0;
      when 15 => return (I < 0);
      when others => return false;
    end case;
end BIT;
```

```
------------------------------------------------------------------------

function TX_READY(COMM_PORT : integer) return boolean is
--
-- returns true when the serial port specified is ready to transmit,
-- false, if it is not ready to transmit.
--
   REGS : interrupt.registers;
begin
   REGS.AX := 16#0300#;
   REGS.DX := COMM_PORT-1;
   interrupt.vector( on => 16#14#, register_block => REGS );
   if BIT(REGS.AX, 4) then
     return true;
   end if;
   return false;
end TX_READY;

------------------------------------------------------------------------

function RS232_WRITE(COMM_PORT : integer; CH : character) return integer is
--
--
--
   REGS : interrupt.registers;
begin
   REGS.AX := 16#0100# or character'pos(CH);
   REGS.DX := COMM_PORT-1;
   interrupt.vector( on => 16#14#, register_block => REGS );
   return REGS.AX;
end RS232_WRITE;

------------------------------------------------------------------------

procedure WRITE_MESSAGE(COMM_PORT : integer; MESSAGE : string) is
--
--
--
   RESULT_CODE : integer;
   ERROR_CODE : string(1..5) := "     ";
begin
   USER_INTERFACE.UPDATE_TX( MESSAGE );

   for INDEX in MESSAGE'range loop
     RESULT_CODE := RS232_WRITE(COMM_PORT, MESSAGE(INDEX));
```

```
      if BIT(RESULT_CODE, 15) then
          USER_INTERFACE.DISPLAY( "*** WRITE FAILED :" &
                                  integer'image(RESULT_CODE) );
          return;
        end if;
      end loop;
end WRITE_MESSAGE;

--------------------------------------------------------------------------

procedure SEND(COMM_PORT : integer; MESSAGE :
MESSAGES_TRANSMITTED) is
--
--
--
   SYNC_MESSAGE : string(1..20) := "SYNC              ";
   SYNC_LENGTH  : integer := LENGTH(long_integer'image(SYNC_COUNT));
begin
   case MESSAGE is
     when SYNC      =>       -- syncronization message
       -- the sync count (0-9999999999) in tail of message
       SYNC_MESSAGE(20-SYNC_LENGTH+1..20) :=
         long_integer'image(SYNC_COUNT);
          WRITE_MESSAGE(COMM_PORT, SYNC_MESSAGE);
       SYNC_COUNT := SYNC_COUNT + 1;

     when READY     => WRITE_MESSAGE(COMM_PORT, "READY             ");
     when COMPTEST  => WRITE_MESSAGE(COMM_PORT, "COMPTEST
");
     when SELFTEST  => WRITE_MESSAGE(COMM_PORT, "SYSTEST           ");
     when NORMAL  => WRITE_MESSAGE(COMM_PORT, "NORMALOPS
");
     when DEGRADED  => WRITE_MESSAGE(COMM_PORT, "DEGRADED
");
     when EMERGENCY  => WRITE_MESSAGE(COMM_PORT, "EMERGENCY
");
     when SHUTDOWN  => WRITE_MESSAGE(COMM_PORT, "SHUTDOWN
");
     when LEVELCAL  => WRITE_MESSAGE(COMM_PORT, "LEVELCAL
");
     when BOILHIGH  => WRITE_MESSAGE(COMM_PORT, "BOILHIGH          ");

     when PUMP1ON   => WRITE_MESSAGE(COMM_PORT, "PUMPON
1");
     when PUMP2ON   => WRITE_MESSAGE(COMM_PORT, "PUMPON
```

```
2");
    when PUMP3ON   => WRITE_MESSAGE(COMM_PORT, "PUMPON
3");
    when PUMP4ON   => WRITE_MESSAGE(COMM_PORT, "PUMPON
4");

    when PUMP1OFF  => WRITE_MESSAGE(COMM_PORT, "PUMPOFF
1");
    when PUMP2OFF  => WRITE_MESSAGE(COMM_PORT, "PUMPOFF
2");
    when PUMP3OFF  => WRITE_MESSAGE(COMM_PORT, "PUMPOFF
3");
    when PUMP4OFF  => WRITE_MESSAGE(COMM_PORT, "PUMPOFF
4");

    when TSTMSG0   => WRITE_MESSAGE(COMM_PORT, "TSTMSG0        ");
    when TSTMSG1   => WRITE_MESSAGE(COMM_PORT, "TSTMSG1        ");
    when TSTMSG2   => WRITE_MESSAGE(COMM_PORT, "TSTMSG2        ");
    when TSTMSG3   => WRITE_MESSAGE(COMM_PORT, "TSTMSG3        ");
    when TSTMSG4   => WRITE_MESSAGE(COMM_PORT, "TSTMSG4        ");
    when TSTMSG5   => WRITE_MESSAGE(COMM_PORT, "TSTMSG5        ");

    when COMPFAIL  => WRITE_MESSAGE(COMM_PORT, "COMPFAIL
");

    when PUMPUS1   => WRITE_MESSAGE(COMM_PORT, "PUMPUS         1");
    when PUMPUS2   => WRITE_MESSAGE(COMM_PORT, "PUMPUS         2");
    when PUMPUS3   => WRITE_MESSAGE(COMM_PORT, "PUMPUS         3");
    when PUMPUS4   => WRITE_MESSAGE(COMM_PORT, "PUMPUS         4");

    when WATFLOWUS1 => WRITE_MESSAGE(COMM_PORT, "WATFLOWUS
1");
    when WATFLOWUS2 => WRITE_MESSAGE(COMM_PORT, "WATFLOWUS
2");
    when WATFLOWUS3 => WRITE_MESSAGE(COMM_PORT, "WATFLOWUS
3");
    when WATFLOWUS4 => WRITE_MESSAGE(COMM_PORT, "WATFLOWUS
4");

    when WATLEVUS  => WRITE_MESSAGE(COMM_PORT, "WATLEVUS
");
    when STMRATUS  => WRITE_MESSAGE(COMM_PORT, "STMRATUS
");

    when PUMPOK1   => WRITE_MESSAGE(COMM_PORT, "PUMPOK
```

```
1");
     when PUMPOK2    => WRITE_MESSAGE(COMM_PORT, "PUMPOK
2");
     when PUMPOK3    => WRITE_MESSAGE(COMM_PORT, "PUMPOK
3");
     when PUMPOK4    => WRITE_MESSAGE(COMM_PORT, "PUMPOK
4");

     when WATFLOWOK1 => WRITE_MESSAGE(COMM_PORT, "WATFLOWOK
1");
     when WATFLOWOK2 => WRITE_MESSAGE(COMM_PORT, "WATFLOWOK
2");
     when WATFLOWOK3 => WRITE_MESSAGE(COMM_PORT, "WATFLOWOK
3");
     when WATFLOWOK4 => WRITE_MESSAGE(COMM_PORT, "WATFLOWOK
4");

     when WATLEVOK   => WRITE_MESSAGE(COMM_PORT, "WATLEVOK
");
     when STMRATOK   => WRITE_MESSAGE(COMM_PORT, "STMRATOK
");

     when STX        => WRITE_MESSAGE(COMM_PORT, ""&ASCII.STX);
     when ETX        => WRITE_MESSAGE(COMM_PORT, ""&ASCII.ETX);
   end case;
end SEND;

-------------------------------------------------------------------------

procedure EXECUTE_JOB(Q : in out QUEUE; STATE_TABLE : BOILER_STATE) is
   JOB : TRANSMIT_JOBS;
begin
  if EMPTY(Q) then
    return;
  end if;
  FIRST(Q, JOB);  -- get the first job in the queue

  -- if the job is a "Transmit Until Ready" and a ready signal has
  -- not been received, then put it back into the queue.
  if (JOB = TX_UNTIL_READY) and (STATE_TABLE.START = false) then
    ADD(Q, TX_UNTIL_READY);
  elsif (JOB = TX_UNTIL_READY) then
    USER_INTERFACE.DISPLAY( "STANDBY received from Boiler." );
  end if;
```

```
if (JOB = TX_BOILER_HIGH) and
  ((STATE_TABLE.MODE /= SELFTEST) or
   (STATE_TABLE.LEVEL < MAXIMUM_OPERATING_LEVEL)) then
   return;  -- no need to send boiler high message
end if;

-- the transmission "header"
SEND(COMM_PORT, STX);
SEND(COMM_PORT, ETX);
SEND(COMM_PORT, TSTMSG0);
SEND(COMM_PORT, TSTMSG1);
SEND(COMM_PORT, TSTMSG2);
case STATE_TABLE.MODE is
  when COMPTEST => SEND(COMM_PORT, COMPTEST);
  when SELFTEST => SEND(COMM_PORT, SELFTEST);
  when NORMAL => SEND(COMM_PORT, NORMAL);
  when DEGRADED => SEND(COMM_PORT, DEGRADED);
  when EMERGENCY => SEND(COMM_PORT, EMERGENCY);
  when SHUTDOWN => SEND(COMM_PORT, SHUTDOWN);
  when others => null;
end case;

case JOB is
  when TX_ALL_PUMPS_ON =>
       SEND(COMM_PORT, PUMP1ON);   SEND(COMM_PORT, PUMP2ON);
       SEND(COMM_PORT, PUMP3ON);   SEND(COMM_PORT, PUMP4ON);
  when TX_ALL_PUMPS_OFF =>
       SEND(COMM_PORT, PUMP1OFF);  SEND(COMM_PORT, PUMP2OFF);
       SEND(COMM_PORT, PUMP3OFF);  SEND(COMM_PORT, PUMP4OFF);
  when TX_ALL_PUMP_STAT =>
       SEND(COMM_PORT, PUMPUS1);  SEND(COMM_PORT, PUMPUS2);
       SEND(COMM_PORT, PUMPUS3);  SEND(COMM_PORT, PUMPUS4);
  when TX_PUMP1ON => SEND(COMM_PORT, PUMP1ON);
  when TX_PUMP2ON => SEND(COMM_PORT, PUMP2ON);
  when TX_PUMP3ON => SEND(COMM_PORT, PUMP3ON);
  when TX_PUMP4ON => SEND(COMM_PORT, PUMP4ON);
  when TX_PUMP1OFF => SEND(COMM_PORT, PUMP1OFF);
  when TX_PUMP2OFF => SEND(COMM_PORT, PUMP2OFF);
  when TX_PUMP3OFF => SEND(COMM_PORT, PUMP3OFF);
  when TX_PUMP4OFF => SEND(COMM_PORT, PUMP4OFF);
  when TX_PUMP1OK => SEND(COMM_PORT, PUMPOK1);
  when TX_PUMP2OK => SEND(COMM_PORT, PUMPOK2);
  when TX_PUMP3OK => SEND(COMM_PORT, PUMPOK3);
  when TX_PUMP4OK => SEND(COMM_PORT, PUMPOK4);
  when TX_PUMP1STAT => SEND(COMM_PORT, PUMPUS1);
```

```
    when TX_PUMP2STAT => SEND(COMM_PORT, PUMPUS2);
    when TX_PUMP3STAT => SEND(COMM_PORT, PUMPUS3);
    when TX_PUMP4STAT => SEND(COMM_PORT, PUMPUS4);
    when TX_MONITOR1OK => SEND(COMM_PORT, WATFLOWOK1);
    when TX_MONITOR2OK => SEND(COMM_PORT, WATFLOWOK2);
    when TX_MONITOR3OK => SEND(COMM_PORT, WATFLOWOK3);
    when TX_MONITOR4OK => SEND(COMM_PORT, WATFLOWOK4);
    when TX_MONITOR1STAT => null;
    when TX_MONITOR2STAT => null;
    when TX_MONITOR3STAT => null;
    when TX_MONITOR4STAT => null;
    when TX_UNTIL_READY => SEND(COMM_PORT, READY);
    when TX_LEVEL_CAL => null;
    when TX_RATEOK =>  SEND(COMM_PORT, STMRATOK);
    when TX_LEVEL_OK => SEND(COMM_PORT, WATLEVOK);
    when TX_BOILER_HIGH => SEND(COMM_PORT, BOILHIGH);
    when others => null;
  end case;

  -- the transmission "trailer"
  SEND(COMM_PORT, TSTMSG3);
  SEND(COMM_PORT, TSTMSG4);
  SEND(COMM_PORT, ETX);
end EXECUTE_JOB;

--------------------------------------------------------------------------

task body TRANSMIT is
   MESSAGE : MESSAGE_BUFFER;
   TRANSMIT_QUEUE : QUEUE;
   LOCAL_STATE_TABLE : BOILER_STATE;
   TASK_IS_RUNNING : boolean := true;
   ACTIVE : boolean := false;
begin
  while TASK_IS_RUNNING loop
    select

      accept TURN_ON_PUMP(PUMP : integer) do
        case PUMP is
          when 1 => ADD(TRANSMIT_QUEUE, TX_PUMP1ON);
          when 2 => ADD(TRANSMIT_QUEUE, TX_PUMP2ON);
          when 3 => ADD(TRANSMIT_QUEUE, TX_PUMP3ON);
               when 4 => ADD(TRANSMIT_QUEUE, TX_PUMP4ON);
               when others => null;
        end case;
```

```
    end TURN_ON_PUMP;

or

    accept TURN_ON_ALL_PUMPS do
      ADD(TRANSMIT_QUEUE, TX_ALL_PUMPS_ON);
    end TURN_ON_ALL_PUMPS;

or

  -- define the comm port to be used and start to transmit message
  -- as they arrive in the que
    accept ENABLE(COMM : integer) do
    ACTIVE := true;
    COMM_PORT := COMM;
    INITIALIZE(TRANSMIT_QUEUE);
    end ENABLE;

or

    accept DISABLE do
      INITIALIZE(TRANSMIT_QUEUE);
    ACTIVE := false;
    end DISABLE;

or

  accept CLEAR_TRANSMIT_QUEUE do
    INITIALIZE(TRANSMIT_QUEUE);
  end CLEAR_TRANSMIT_QUEUE;

or

  accept TURN_OFF_PUMP(PUMP : integer) do
    case PUMP is
      when 1 => ADD(TRANSMIT_QUEUE, TX_PUMP1OFF);
      when 2 => ADD(TRANSMIT_QUEUE, TX_PUMP2OFF);
      when 3 => ADD(TRANSMIT_QUEUE, TX_PUMP3OFF);
          when 4 => ADD(TRANSMIT_QUEUE, TX_PUMP4OFF);
          when others => null;
    end case;
    end TURN_OFF_PUMP;

or
```

```
      accept TURN_OFF_ALL_PUMPS do
        ADD(TRANSMIT_QUEUE, TX_ALL_PUMPS_OFF);
      end TURN_OFF_ALL_PUMPS;

or

    --
    -- This message is sent when the program is ready to work and
    -- is repeated every tranmission until a valid start signal is
    -- received.
    --
    accept UNTIL_READY do
      ADD(TRANSMIT_QUEUE, TX_UNTIL_READY);
    end UNTIL_READY;

or

    --
    -- This message is sent until acknowledged, to indicate that the
    -- program has detected a problem with a pump
    --
    accept PUMP_STATUS(PUMP : integer) do
      case PUMP is
        when 1 => ADD(TRANSMIT_QUEUE, TX_PUMP1STAT);
        when 2 => ADD(TRANSMIT_QUEUE, TX_PUMP2STAT);
        when 3 => ADD(TRANSMIT_QUEUE, TX_PUMP3STAT);
              when 4 => ADD(TRANSMIT_QUEUE, TX_PUMP4STAT);
              when others => null;
      end case;
    end PUMP_STATUS;

or

      accept STATUS_OF_ALL_PUMPS do
        ADD(TRANSMIT_QUEUE, TX_ALL_PUMP_STAT);
      end STATUS_OF_ALL_PUMPS;

or

    --
    -- This message are sent as acknowledgements, each time that the
    -- instrumentation system has sent a message reporting that a
    -- particular device has been repaired.
    --
    accept PUMP_OK(PUMP : integer) do
```

```
      case PUMP is
        when 1 => ADD(TRANSMIT_QUEUE, TX_PUMP1OK);
        when 2 => ADD(TRANSMIT_QUEUE, TX_PUMP2OK);
        when 3 => ADD(TRANSMIT_QUEUE, TX_PUMP3OK);
             when 4 => ADD(TRANSMIT_QUEUE, TX_PUMP4OK);
             when others => null;
      end case;
    end PUMP_OK;

or

    accept MONITOR_OK(MONITOR : integer) do
      case MONITOR is
        when 1 => ADD(TRANSMIT_QUEUE, TX_MONITOR1OK);
        when 2 => ADD(TRANSMIT_QUEUE, TX_MONITOR2OK);
        when 3 => ADD(TRANSMIT_QUEUE, TX_MONITOR3OK);
             when 4 => ADD(TRANSMIT_QUEUE, TX_MONITOR4OK);
             when others => null;
      end case;
    end MONITOR_OK;

or

    accept WATER_CONTENT_OK do
      ADD(TRANSMIT_QUEUE, TX_WATLEVOK);
    end WATER_CONTENT_OK;

or

    accept RATE_OK do
      ADD(TRANSMIT_QUEUE, TX_RATEOK);
    end RATE_OK;

or

    --
    -- This message may be sent by the program at any time.
    --
    accept LEVEL_CAL do
      ADD(TRANSMIT_QUEUE, TX_LEVELCAL);
    end LEVEL_CAL;

or

    accept UPDATE(GLOBAL_STATE_TABLE : BOILER_STATE) do
```

```
        LOCAL_STATE_TABLE := GLOBAL_STATE_TABLE;
      end UPDATE;

    or

      --
      -- The message may only be sent in the system test and initialization
      -- mode, and only when the specified boiler water level is too high.
      --
      accept BOILER_HIGH do
          ADD(TRANSMIT_QUEUE, TX_BOILER_HIGH);
      end BOILER_HIGH;

    or

        accept KILL do
          TASK_IS_RUNNING := false;
        end KILL;

    else

        if ACTIVE then -- and LOCAL_STATE_TABLE.OK_TO_TRANSMIT then
        EXECUTE_JOB(TRANSMIT_QUEUE, LOCAL_STATE_TABLE);
        end if;

    end select;

  end loop;
  text_io.put_line( "... TRANSMIT Task is now terminating." );

  exception
    when others =>
        text_io.put_line( "**** Exception occured in TRANSMIT." );
end TRANSMIT;

-------------------------------------------------------------------------

end TX;
```

```
--  ************************************************************************
--
--                      USER INTERFACE Package
--        The User Interface packages displays fields to the CRT as
--     well as accepts input for the keyboard and set the appropriate
--     boolean variables.
--
--     written by: Jerry D. Cavin          date: April 3rd, 1993
--
--************************************************************************

with fio;                    -- float input/output package
with iio;                    -- integer Input/Output package
with text_io;                  -- text Input/Output package
with tty;                    -- DOS Video package
with interrupt;                -- DOS call package
with time;                   -- DOS time function
with box;      use box;        -- DOS routine to draw boxes

with QUEUE;                    -- the GENERIC Queue

with common_display_types;       -- color constants
use common_display_types;

with SYSTEM_PARAMETERS;          use SYSTEM_PARAMETERS;

package UI is

  task USER_INTERFACE is
    entry CLEAR_SCREEN;
    entry UPDATE( GLOBAL_STATE_TABLE : BOILER_STATE );
    entry HIGHLIGHT(MODE : OPERATING_MODES);
    entry UNHIGHLIGHT(MODE : OPERATING_MODES);
    entry UPDATE_LABELS;
    entry UPDATE_INSTRUMENT_TIME;
    entry UPDATE_LEVEL;
    entry UPDATE_LEVEL_CAL;
    entry UPDATE_LEVEL_DEVICE;
    entry UPDATE_RATE;
    entry UPDATE_RATE_DEVICE;
    entry UPDATE_PUMP( N : integer );
    entry UPDATE_PUMP_DEVICE( N : integer );
    entry UPDATE_MONITOR( N : integer );
    entry UPDATE_MONITOR_DEVICE( N : integer );
    entry CHECK_FOR_TERMINATION( READY_TO_TERMINATE : in out boolean
```

```
);
    entry CHECK_FOR_INITIATION( READY_TO_INITIATE : in out boolean );
    entry KILL;
    entry DISPLAY(TEXT : string);
    entry UPDATE_TX( MESSAGE : string );
    entry UPDATE_RX( MESSAGE : string );
    entry DISABLE;
  end USER_INTERFACE;
end UI;

package body UI is

   BLINK_ON    : constant boolean := true;
   BLINK_OFF   : constant boolean := false;
   NO_ECHO     : constant boolean := true;
   DIRECT      : constant boolean := true;
   CLEAR       : constant boolean := true;

   BLANK_LINE  : string(1..65) := (1..65 => ' ');
   NEW_LINE, LINE1, LINE2, LINE3 : string(1..65) := (1..65 => ' ');

   LOCAL_STATE_TABLE : BOILER_STATE;

   TX_MESSAGE, RX_MESSAGE : MESSAGE_BUFFER;

   --
   -- these are the list of the background chores for the user
   -- interface
   --
   type UI_JOBS is (
         DISPLAY_LABELS, CLEAR_SCREEN,

         UNHIGHLIGHT_SELFTEST, UNHIGHLIGHT_COMPTEST,
UNHIGHLIGHT_NORMAL,
         UNHIGHLIGHT_DEGRADED, UNHIGHLIGHT_EMERGENCY,
UNHIGHLIGHT_SHUTDOWN,

         HIGHLIGHT_SELFTEST,  HIGHLIGHT_COMPTEST,
HIGHLIGHT_NORMAL,
         HIGHLIGHT_DEGRADED,  HIGHLIGHT_EMERGENCY,
HIGHLIGHT_SHUTDOWN,

         DISPLAY_TIME, DISPLAY_LEVEL, DISPLAY_LEVEL_CAL,
         DISPLAY_LEVEL_DEVICE, DISPLAY_RATE, DISPLAY_RATE_DEVICE,
```

```
        DISPLAY_PUMP1, DISPLAY_PUMP2, DISPLAY_PUMP3,
DISPLAY_PUMP4,
        DISPLAY_PUMP1_DEVICE, DISPLAY_PUMP2_DEVICE,
        DISPLAY_PUMP3_DEVICE, DISPLAY_PUMP4_DEVICE,

        DISPLAY_MONITOR1, DISPLAY_MONITOR2, DISPLAY_MONITOR3,
DISPLAY_MONITOR4,
        DISPLAY_MONITOR1_DEVICE, DISPLAY_MONITOR2_DEVICE,
        DISPLAY_MONITOR3_DEVICE, DISPLAY_MONITOR4_DEVICE,

        DISPLAY_TEXT, DISPLAY_RX, DISPLAY_TX );

   --
   -- this is the queue where the background chores are kept
   --
   package UI_QUEUE is new QUEUE(100, UI_JOBS);

-------------------------------------------------------------------------

function MIN(X,Y : integer) return integer is
begin
  if X < Y then
    return X;
  end if;
  return Y;
end MIN;
-------------------------------------------------------------------------

procedure CLEAR_BUFFER(BUFFER : in out string) is
--
--
--
begin
  BUFFER(1..BUFFER'length) := (1..BUFFER'length => ' ');
end CLEAR_BUFFER;

-------------------------------------------------------------------------

procedure COPY_BUFFER(SOURCE : in string;TARGET : in out string) is
begin
  CLEAR_BUFFER(TARGET);
  TARGET(1..MIN(SOURCE'length, TARGET'length)) := SOURCE;
end COPY_BUFFER;

-------------------------------------------------------------------------
```

```
procedure DISPLAY_PUMP(PUMP : integer) is
  COLUMN : array(1..NUMBER_OF_PUMPS) of integer := (16, 30, 44, 58);
begin
  if LOCAL_STATE_TABLE.PUMP(PUMP) = ON then
    tty.put( 7, COLUMN(PUMP), "ON          ", GREEN, BLACK, BLINK_OFF );
  else
    tty.put( 7, COLUMN(PUMP), "OFF         ", RED, BLACK, BLINK_OFF );
  end if;
end DISPLAY_PUMP;

-------------------------------------------------------------------------

procedure DISPLAY_MONITOR(MONITOR : integer) is
  COLUMN : array(1..NUMBER_OF_PUMPS) of integer := (16, 30, 44, 58);
begin
  if LOCAL_STATE_TABLE.MONITOR(MONITOR) = FLOW then
    tty.put( 8, COLUMN(MONITOR), "FLOW ON      ", GREEN, BLACK,
BLINK_OFF );
  else
    tty.put( 8, COLUMN(MONITOR), "FLOW OFF     ", RED, BLACK, BLINK_OFF );
  end if;
end DISPLAY_MONITOR;

-------------------------------------------------------------------------

procedure DISPLAY_PUMP_DEVICE(DEVICE : integer) is
  COLUMN : array(1..NUMBER_OF_PUMPS) of integer := (16, 30, 44, 58);
begin
  if LOCAL_STATE_TABLE.PUMP_DEVICE(DEVICE) = WORKING then
    tty.put( 10, COLUMN(DEVICE), "SERVICEABLE  ", GREEN, BLACK,
BLINK_OFF );
  elsif LOCAL_STATE_TABLE.PUMP_DEVICE(DEVICE) = REPAIR_REQ then
    tty.put( 10, COLUMN(DEVICE), "UNVERIFIED   ", RED, BLACK, BLINK_ON );
  elsif LOCAL_STATE_TABLE.PUMP_DEVICE(DEVICE) = REPAIR_ACK then
    tty.put( 10, COLUMN(DEVICE), "UNSERVICEABLE", RED, BLACK,
BLINK_ON );
  else
    tty.put( 10, COLUMN(DEVICE), "?????????????", RED, BLACK, BLINK_ON );
  end if;
end DISPLAY_PUMP_DEVICE;

-------------------------------------------------------------------------

procedure DISPLAY_MONITOR_DEVICE(DEVICE : integer) is
```

```
  COLUMN : array(1..NUMBER_OF_PUMPS) of integer := (16, 30, 44, 58);
begin
  if LOCAL_STATE_TABLE.MONITOR_DEVICE(DEVICE) = WORKING then
    tty.put( 11, COLUMN(DEVICE), "SERVICEABLE  ", GREEN, BLACK,
BLINK_OFF );
  elsif LOCAL_STATE_TABLE.MONITOR_DEVICE(DEVICE) = REPAIR_REQ then
    tty.put( 11, COLUMN(DEVICE), "UNVERIFIED   ", RED, BLACK, BLINK_ON );
  elsif LOCAL_STATE_TABLE.MONITOR_DEVICE(DEVICE) = REPAIR_ACK then
    tty.put( 11, COLUMN(DEVICE), "UNSERVICEABLE", RED, BLACK,
BLINK_ON );
  else
    tty.put( 11, COLUMN(DEVICE), "?????????????", RED, BLACK, BLINK_ON );
  end if;
end DISPLAY_MONITOR_DEVICE;

--------------------------------------------------------------------------

procedure DISPLAY_NEW_TEXT( TEXT : string ) is
begin
  LINE1 := LINE2;   LINE2 := LINE3;
  COPY_BUFFER(TEXT, LINE3);
  LINE3(1..MIN(LINE3'length, TEXT'length)) := TEXT;
  tty.put( 20, 4, LINE1, YELLOW, BLACK );
  tty.put( 21, 4, LINE2, YELLOW, BLACK );
  tty.put( 22, 4, LINE3, YELLOW, BLACK );
end DISPLAY_NEW_TEXT;

--------------------------------------------------------------------------

procedure PROCESS_JOBS(STATE_TABLE : in out BOILER_STATE) is
begin
  if UI_QUEUE.EMPTY then
    return;
  end if;

  case UI_QUEUE.GET is
    when DISPLAY_LABELS =>
        tty.put( 2, 25, "Boiler Control Program V1.0",
                        LIGHT_BLUE, BLACK, BLINK_OFF);
        tty.put( 4,  5, "SELFTEST", GREEN, BLACK, BLINK_OFF);
        tty.put( 4, 18, "COMPTEST", GREEN, BLACK, BLINK_OFF);
        tty.put( 4, 33, "NORMAL", GREEN, BLACK, BLINK_OFF);
        tty.put( 4, 43, "DEGRADED", GREEN, BLACK, BLINK_OFF);
        tty.put( 4, 55, "EMERGENCY", GREEN, BLACK, BLINK_OFF);
        tty.put( 4, 67, "SHUTDOWN", GREEN, BLACK, BLINK_OFF);
```

```
        box.draw(  1, 1, 24, 79, box.double_top, (LIGHT_BLUE,BLACK,false) );
        tty.put( 24, 3, "B", BLUE, WHITE);
        tty.put( 24, 4, "START", WHITE, BLUE);
        tty.put( 24,10, "X", BLUE, WHITE);
        tty.put( 24,11, "EXIT", WHITE, BLUE);
        box.draw(  6, 3, 12, 15, single_top, (GREEN,BLACK,false) );
        box.draw(  6, 3, 12, 29, single_top, (GREEN,BLACK,false) );
        box.draw(  6, 3, 12, 43, single_top, (GREEN,BLACK,false) );
        box.draw(  6, 3, 12, 57, single_top, (GREEN,BLACK,false) );
        box.draw(  6, 3,  9, 71, single_top, (GREEN,BLACK,false) );
        box.draw(  6, 3, 12, 71, single_top, (GREEN,BLACK,false) );

        box.draw( 19, 3, 23, 76, single_top, (RED,BLACK,false) );
        tty.put( 19, 38, "STDIO", BLACK, RED);

        tty.put( 7, 9, "PUMP", YELLOW, BLACK);
        tty.put( 8, 6, "MONITOR", YELLOW, BLACK);
        tty.put(10, 9, "PUMP", YELLOW, BLACK);
        tty.put(11, 6, "MONITOR", YELLOW, BLACK);

        tty.put( 14, 10, "Rate State", YELLOW, BLACK);
        tty.put( 15, 10, "Rate Status", YELLOW, BLACK);
        tty.put( 14, 41, "Level State", YELLOW, BLACK);
        tty.put( 15, 41, "Level Status", YELLOW, BLACK);

        tty.put( 17, 6, "TX [", YELLOW, BLACK, BLINK_OFF);
        tty.put( 17, 31, "]", YELLOW, BLACK, BLINK_OFF);
        tty.put( 17, 41, "RX [", YELLOW, BLACK, BLINK_OFF);
        tty.put( 17, 66, "]", YELLOW, BLACK, BLINK_OFF);

        -- instrument time
        tty.put( 2, 60,
long_integer'image(LOCAL_STATE_TABLE.INSTRUMENT_TIME),
                        GREEN, BLACK, BLINK_OFF);

        -- boiler water level
        if (LOCAL_STATE_TABLE.LEVEL >= MINIMUM_OPERATING_LEVEL)
and
          (LOCAL_STATE_TABLE.LEVEL <= MAXIMUM_OPERATING_LEVEL)
then
          tty.put( 14, 56, long_integer'image(LOCAL_STATE_TABLE.LEVEL),
                GREEN, BLACK, BLINK_OFF);
        else
          tty.put( 14, 56, long_integer'image(LOCAL_STATE_TABLE.LEVEL),
                BLACK, RED, BLINK_ON);
```

```
      end if;

      for PUMP in 1..NUMBER_OF_PUMPS loop
        DISPLAY_PUMP(PUMP);
        DISPLAY_PUMP_DEVICE(PUMP);
        DISPLAY_MONITOR(PUMP);
        DISPLAY_MONITOR_DEVICE(PUMP);
      end loop;

      -- steam rate measurement
      if (LOCAL_STATE_TABLE.RATE >= MINIMUM_RATE_READING) and
        (LOCAL_STATE_TABLE.RATE <= MAXIMUM_RATE_READING) then
        tty.put( 14, 56, long_integer'image(LOCAL_STATE_TABLE.RATE),
                 GREEN, BLACK, BLINK_OFF);
      else
        tty.put( 14, 56, long_integer'image(LOCAL_STATE_TABLE.RATE),
                 BLACK, RED, BLINK_ON);
      end if;

  when CLEAR_SCREEN =>
      tty.clear_screen;

  when UNHIGHLIGHT_SELFTEST =>
      tty.put(4,  5, "SELFTEST", GREEN, WHITE, BLINK_OFF);

  when UNHIGHLIGHT_COMPTEST =>
      tty.put(4, 18, "COMPTEST", GREEN, WHITE, BLINK_OFF);

  when UNHIGHLIGHT_NORMAL =>
      tty.put(4, 33, "NORMAL", GREEN, WHITE, BLINK_OFF);

  when UNHIGHLIGHT_DEGRADED =>
      tty.put(4, 43, "DEGRADED", GREEN, WHITE, BLINK_OFF);

  when UNHIGHLIGHT_EMERGENCY =>
      tty.put(4, 55, "EMERGENCY", GREEN, WHITE, BLINK_OFF);

  when UNHIGHLIGHT_SHUTDOWN =>
     tty.put(4, 67, "SHUTDOWN", GREEN, WHITE, BLINK_OFF);

  when HIGHLIGHT_SELFTEST =>
      tty.put( 4,  5, "SELFTEST", WHITE, GREEN, BLINK_ON);

  when HIGHLIGHT_COMPTEST =>
      tty.put( 4, 18, "COMPTEST", WHITE, GREEN, BLINK_ON);
```

```
    when HIGHLIGHT_NORMAL =>
        tty.put( 4, 33, "NORMAL", WHITE, GREEN, BLINK_ON);

    when HIGHLIGHT_DEGRADED =>
        tty.put( 4, 43, "DEGRADED", WHITE, GREEN, BLINK_ON);

    when HIGHLIGHT_EMERGENCY =>
        tty.put( 4, 55, "EMERGENCY", WHITE, GREEN, BLINK_ON);

    when HIGHLIGHT_SHUTDOWN =>
        tty.put( 4, 67, "SHUTDOWN", WHITE, GREEN, BLINK_ON);

    when DISPLAY_TIME =>
        tty.put( 2, 60,
long_integer'image(LOCAL_STATE_TABLE.INSTRUMENT_TIME),
                GREEN, BLACK, BLINK_OFF);

    when DISPLAY_LEVEL =>
        if (LOCAL_STATE_TABLE.LEVEL >= MINIMUM_OPERATING_LEVEL)
and
          (LOCAL_STATE_TABLE.LEVEL <= MAXIMUM_OPERATING_LEVEL)
then
          tty.put( 14, 56, long_integer'image(LOCAL_STATE_TABLE.LEVEL),
                GREEN, BLACK, BLINK_OFF);
        else
          tty.put( 14, 56, long_integer'image(LOCAL_STATE_TABLE.LEVEL),
                BLACK, RED, BLINK_ON);
        end if;

    when DISPLAY_LEVEL_CAL => null;

    when DISPLAY_LEVEL_DEVICE => null;

    when DISPLAY_RATE => null;
        if (LOCAL_STATE_TABLE.RATE >= MINIMUM_RATE_READING) and
          (LOCAL_STATE_TABLE.RATE <= MAXIMUM_RATE_READING) then
          tty.put( 14, 56, long_integer'image(LOCAL_STATE_TABLE.RATE),
                GREEN, BLACK, BLINK_OFF);
        else
          tty.put( 14, 56, long_integer'image(LOCAL_STATE_TABLE.RATE),
                BLACK, RED, BLINK_ON);
        end if;

    when DISPLAY_RATE_DEVICE => null;
```

```
    when DISPLAY_PUMP1 => DISPLAY_PUMP(1);
    when DISPLAY_PUMP2 => DISPLAY_PUMP(2);
    when DISPLAY_PUMP3 => DISPLAY_PUMP(3);
    when DISPLAY_PUMP4 => DISPLAY_PUMP(4);
    when DISPLAY_PUMP1_DEVICE => DISPLAY_PUMP_DEVICE(1);
    when DISPLAY_PUMP2_DEVICE => DISPLAY_PUMP_DEVICE(2);
    when DISPLAY_PUMP3_DEVICE => DISPLAY_PUMP_DEVICE(3);
    when DISPLAY_PUMP4_DEVICE => DISPLAY_PUMP_DEVICE(4);
    when DISPLAY_MONITOR1 => DISPLAY_MONITOR(1);
    when DISPLAY_MONITOR2 => DISPLAY_MONITOR(2);
    when DISPLAY_MONITOR3 => DISPLAY_MONITOR(3);
    when DISPLAY_MONITOR4 => DISPLAY_MONITOR(4);
    when DISPLAY_MONITOR1_DEVICE => DISPLAY_MONITOR_DEVICE(1);
    when DISPLAY_MONITOR2_DEVICE => DISPLAY_MONITOR_DEVICE(2);
    when DISPLAY_MONITOR3_DEVICE => DISPLAY_MONITOR_DEVICE(3);
    when DISPLAY_MONITOR4_DEVICE => DISPLAY_MONITOR_DEVICE(4);

    when DISPLAY_TEXT => DISPLAY_NEW_TEXT( NEW_LINE );

    when DISPLAY_RX =>
          tty.put(17, 46, "                    " );
          tty.put(17, 46, RX_MESSAGE, RED, BLACK, BLINK_OFF);

    when DISPLAY_TX =>
          tty.put(17, 10, "                    " );
          tty.put(17, 10, TX_MESSAGE, RED, BLACK, BLINK_OFF);

    when others => null;

  end case;
end PROCESS_JOBS;

-------------------------------------------------------------------------

task body USER_INTERFACE is
  USER_REQUESTS_TERMINATION : boolean := false;
  USER_REQUESTS_INITIATION  : boolean := false;
  TASK_IS_RUNNING : boolean := true;
  ACTIVE : boolean := true;
  CMD : character;
begin
  UI_QUEUE.INITIALIZE;
  while TASK_IS_RUNNING loop
    select
```

```
        accept UPDATE( GLOBAL_STATE_TABLE : BOILER_STATE ) do
          LOCAL_STATE_TABLE := GLOBAL_STATE_TABLE;
        end UPDATE;

    or

        accept DISABLE do
          ACTIVE := false;
        end DISABLE;

    or

        -- find out if the user has entered the command to terminate
        -- boiler control program
        accept CHECK_FOR_TERMINATION(READY_TO_TERMINATE : in out
boolean) do
          READY_TO_TERMINATE := USER_REQUESTS_TERMINATION;
        end CHECK_FOR_TERMINATION;

    or

        -- find out if the user has entered the command to start
        -- boiler control program
        accept CHECK_FOR_INITIATION(READY_TO_INITIATE : in out boolean)
do
          READY_TO_INITIATE := USER_REQUESTS_INITIATION;
        end CHECK_FOR_INITIATION;

    or

        -- put all of the labels and boxes on the screen
        accept UPDATE_LABELS do
          UI_QUEUE.PUT(DISPLAY_LABELS);
        end UPDATE_LABELS;

    or

        accept CLEAR_SCREEN do
          UI_QUEUE.PUT(CLEAR_SCREEN);
        end CLEAR_SCREEN;

    or

        accept UNHIGHLIGHT(MODE : OPERATING_MODES) do
```

```
            case MODE is
              when SELFTEST => UI_QUEUE.PUT(UNHIGHLIGHT_SELFTEST);
              when COMPTEST => UI_QUEUE.PUT(UNHIGHLIGHT_COMPTEST);
              when NORMAL => UI_QUEUE.PUT(UNHIGHLIGHT_NORMAL);
              when DEGRADED => UI_QUEUE.PUT(UNHIGHLIGHT_DEGRADED);
              when EMERGENCY =>
UI_QUEUE.PUT(UNHIGHLIGHT_EMERGENCY);
              when SHUTDOWN => UI_QUEUE.PUT(UNHIGHLIGHT_SHUTDOWN);
            end case;
          end UNHIGHLIGHT;

      or

          accept HIGHLIGHT(MODE : OPERATING_MODES) do
            case MODE is
              when SELFTEST => UI_QUEUE.PUT(HIGHLIGHT_SELFTEST);
              when COMPTEST => UI_QUEUE.PUT(HIGHLIGHT_COMPTEST);
              when NORMAL => UI_QUEUE.PUT(HIGHLIGHT_NORMAL);
              when DEGRADED => UI_QUEUE.PUT(HIGHLIGHT_DEGRADED);
              when EMERGENCY => UI_QUEUE.PUT(HIGHLIGHT_EMERGENCY);
              when SHUTDOWN => UI_QUEUE.PUT(HIGHLIGHT_SHUTDOWN);
            end case;
          end HIGHLIGHT;

      or

          -- if the level is between the normal operating parameters
          -- then display it in non-blinking form, if it is outside
          -- of normal parameters then make it blink.
          accept UPDATE_LEVEL do
            UI_QUEUE.PUT(DISPLAY_LEVEL);
          end UPDATE_LEVEL;

      or

          accept UPDATE_LEVEL_CAL do
            UI_QUEUE.PUT(DISPLAY_LEVEL_CAL);
          end UPDATE_LEVEL_CAL;

      or

          accept UPDATE_LEVEL_DEVICE do
            UI_QUEUE.PUT(DISPLAY_LEVEL_DEVICE);
          end UPDATE_LEVEL_DEVICE;
```

```
or

    accept UPDATE_RATE do
      UI_QUEUE.PUT(DISPLAY_RATE);
    end UPDATE_RATE;

or

    accept UPDATE_RATE_DEVICE do
      UI_QUEUE.PUT(DISPLAY_RATE_DEVICE);
    end UPDATE_RATE_DEVICE;

or

    accept UPDATE_PUMP(N : integer) do
      case N is
          when 1 => UI_QUEUE.PUT(DISPLAY_PUMP1);
          when 2 => UI_QUEUE.PUT(DISPLAY_PUMP2);
          when 3 => UI_QUEUE.PUT(DISPLAY_PUMP3);
          when 4 => UI_QUEUE.PUT(DISPLAY_PUMP4);
          when others => null;
      end case;
    end UPDATE_PUMP;

or

    accept UPDATE_PUMP_DEVICE(N : integer) do
      case N is
          when 1 => UI_QUEUE.PUT(DISPLAY_PUMP1_DEVICE);
          when 2 => UI_QUEUE.PUT(DISPLAY_PUMP2_DEVICE);
          when 3 => UI_QUEUE.PUT(DISPLAY_PUMP3_DEVICE);
          when 4 => UI_QUEUE.PUT(DISPLAY_PUMP4_DEVICE);
          when others => null;
      end case;
    end UPDATE_PUMP_DEVICE;

or

    accept UPDATE_MONITOR(N : integer) do
      case N is
          when 1 => UI_QUEUE.PUT(DISPLAY_MONITOR1);
          when 2 => UI_QUEUE.PUT(DISPLAY_MONITOR2);
          when 3 => UI_QUEUE.PUT(DISPLAY_MONITOR3);
          when 4 => UI_QUEUE.PUT(DISPLAY_MONITOR4);
          when others => null;
```

```
      end case;
    end UPDATE_MONITOR;

or

    accept UPDATE_MONITOR_DEVICE(N : integer) do
      case N is
          when 1 => UI_QUEUE.PUT(DISPLAY_MONITOR1_DEVICE);
          when 2 => UI_QUEUE.PUT(DISPLAY_MONITOR2_DEVICE);
          when 3 => UI_QUEUE.PUT(DISPLAY_MONITOR3_DEVICE);
          when 4 => UI_QUEUE.PUT(DISPLAY_MONITOR4_DEVICE);
          when others => null;
      end case;
    end UPDATE_MONITOR_DEVICE;

or

    accept DISPLAY(TEXT : string) do
      COPY_BUFFER(TEXT, NEW_LINE);
      UI_QUEUE.PUT(DISPLAY_TEXT);
    end DISPLAY;

or

    accept UPDATE_RX(MESSAGE : string) do
      COPY_BUFFER(MESSAGE, RX_MESSAGE);
      UI_QUEUE.PUT(DISPLAY_RX);
    end UPDATE_RX;

or

    accept UPDATE_TX(MESSAGE : string) do
      COPY_BUFFER(MESSAGE, TX_MESSAGE);
      UI_QUEUE.PUT(DISPLAY_TX);
    end UPDATE_TX;

or

    accept KILL do
      TASK_IS_RUNNING := false;
    end KILL;

else -- check for keyboard input here

    if ACTIVE then
```

```
            if tty.char_ready then
                CMD := tty.get(NO_ECHO, DIRECT, false);
                case CMD is
                  when 'x'|'X' =>
                    USER_REQUESTS_TERMINATION := true;
                    DISPLAY_NEW_TEXT( "User requests termination." );
                  when 'b'|'B' =>
                    USER_REQUESTS_INITIATION := true;
                    DISPLAY_NEW_TEXT( "User request initiation." );
                  when others => null;
                end case;
              end if;

              PROCESS_JOBS( LOCAL_STATE_TABLE );
            end if;

        end select;

    end loop;

    exception
      when others => return;
end USER_INTERFACE;

end UI;
```