EMBEDDING SIMULATION MODELING IN DEVELOPMENT OF HIGH AUTONOMY SYSTEMS

Tag Gon Kim
Minsun Chung

Department of Electrical and Computer Engineering
University of Kansas
Lawrence, KS 66045

Abstract

One of the key problems in developing a high autonomy system stems from the embedded nature of such a system where software and hardware components must be integrated to achieve the autonomy. Not all of the desired hardware and/or software components may be directly applicable for developing such an embedded system. This paper proposes a simulation embedded methodology that can remedy the potential difficulties in such circumstances. In this methodology hardware and/or software components that are not applicable during the development time are replaced by simulated models of the components. These simulated models are embedded into the real-time installation of the software unit of the system. The proposed methodology is demonstrated by developing an intelligent real-time control system.

1. Introduction

A high autonomy system is an intelligent, real-time system with self-determination capability for carrying out predefined objectives over an extended period of time in an uncertain environment. In order to achieve the desired level of autonomy, such a system must be capable of making intelligent decisions in response to the changing environment. This requires the system to be equipped with the abilities for sensing the changes in the environment, reasoning about the changes, planning and carrying out actions to be taken in response to the changes. The prospects for building such sophisticated systems have been greatly enhanced by the recent advancements in both software and hardware technologies as the availability of the robust hardware and intelligent software components is vital in realizing the autonomy of the system [1].

A design of a high autonomy system requires the designer to specify the perception, decision, and action components as an integrated unit capable of achieving the desired level of autonomy in the system. Decisions for selecting the hardware and software components for realizing the integrated system are usually postponed until the implementation stage.

Due to the embedded nature, the development of high autonomy systems involves problems not found in the development of pure software systems. These include unavailability of up-to-spec off-the-shelf hardware components, the vulnerability of critical hardware components to software faults, and concerns for potentially hazardous conditions arising from hardware malfunctions, to list a few. Such conditions may lead to the situation where not all of the required components may be applicable at the development time.

This paper describes a simulation based solution for resolving this type of problems. The methodology applies simulation models to imitate the behaviors of the components that are not directly applicable at the development time. Such models are embedded into real-time software modules under development and simulated in real time when such modules are tested. The methodology is demonstrated by developing a real time control software system called the Real-Time Control and Data Acquisition System (RTCDAS) which is a part of an intelligent environmental control system, AIDECS (AI-Based, Distributed Environmental Control System) [2].

Section 2 briefly describes the architecture for the AIDECS. A methodology for embedding simulation models in real-time systems development is presented in section 3. Section 4 gives informal specifications of components within the AIDECS that are directly connected to the RTCDAS under development. Simulation models for the components described in section 4 are given in section 5. Design, implementation, and testing of the RTCDAS are described in section 6 and section 7. Concluding remarks are given in section 8.

2. Overview of AIDECS

The AIDECS is designed as an intelligent control system for a Bioregenerative, Closed Ecological Life Support System (BCELSS) [2]. The AIDECS consists of the Consultation Expert Systems (CES), the Natural Language User Interface (NLUI), the Constraints Checker (CC), the Schedule Manager (SM), the Schedule Executor (SE), the Control Data Base (CDB), the Schedule Decision Expert System (SDES), and the Real-Time CTRL & DAQ System (RTCDAS), as shown in Fig. 1.

The CES consists of a set of expert systems for identifying and controlling pest such as aphids in the BCELSS. The User Interface System enables the user to specify a time-based schedule to the environmental control system. The Constraint Checker sends the schedule to the Schedule Manager after deciding the acceptability of the schedule by testing it against a set of constraints imposed. The Schedule Manager generates a specification for the schedule, translates the specification into a schedule object, and then sends the object to the Schedule Evaluator. The Schedule Evaluator transforms the object into a set of activities, each of which has slots containing a pair of condition and action, and other relevant information. The Schedule Evaluator continuously evaluates each activity in the set; if the condition of an activity is satisfied, the associated action in the activity is fired. The firing of the actions result in sending a micro-level control signal(s) to the RTCDAS. This signal contains the information on the set point(s) such as temperature and location within the BCELSS. More detailed descriptions on the AIDECS can be found in [2]

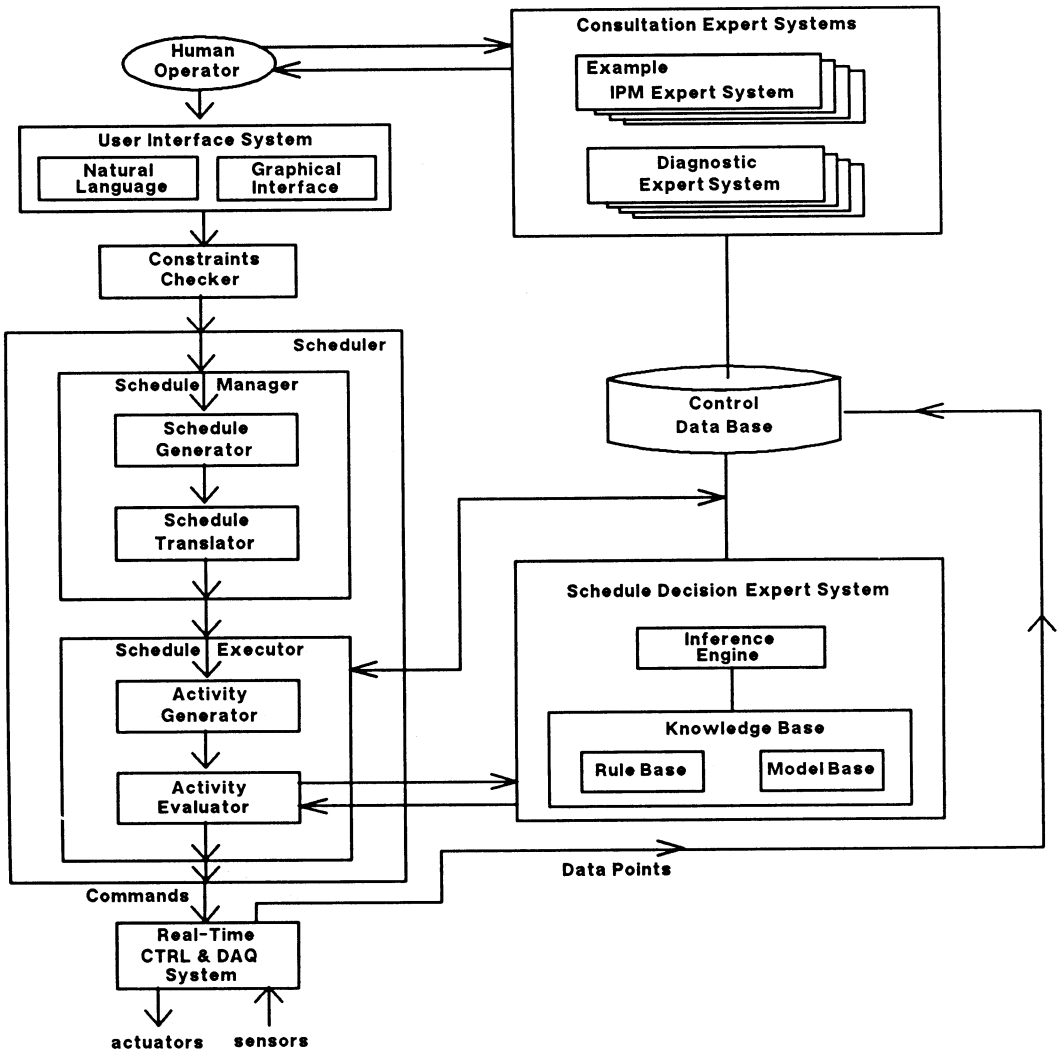

Fig. 1. Architecture of Intelligent Control System AIDCES.

3. Methodology for Embedding Models

As we described earlier, a major problem in development of a real-time system that we are concerned is that not all of the required hardware/software components may be applicable at the development time. We discuss a methodology for resolving this type of problems in development of high autonomy systems. The methodology applies simulation models to imitate the behaviors of the hardware and/or software components that are not directly applicable at the development time. As the real-time requirement is critical for the viability of the autonomy system, the simulated models must be embedded into the software installment of the system in real time. As both hardware and software components in an autonomous system are basically of event driven nature, the simulation modeling based methodology fits very naturally into the system development process. The major requirement for the methodology is that the simulated models be able to emulate the desired functionalities of the unavailable components under specified timing constraints. Only then can the software installment as a whole meet the real-time requirements imposed on the high autonomy system.

To explain the methodology described above in more detail, we consider development of the RTCDAS within the AIDECS described in section 2. Assume that the RTCDAS monitors and controls two environmental parameters, namely temperature and humidity, in distributed locations. We call such locations biomes, each of which consists of plots, each of which, in turn, consists of sectors. In modelling hardware/software components in the RTCDAS development, we only consider system components within the AIDECS that are directly coupled to the RTCDAS. Such components are Scheduling System (Scheduler), Control Data Base, Actuators, Sensors, and Controlled Locations or Biomes. We develop simulation models for the components and software modules for the RTCDAS. Our simulation models are simple, event-driven ones which can be implemented as concurrent processes using Ada tasks.

4. Brief Description of Related Modules

Even though detailed descriptions for each component were given in [2] we briefly describe functional specifications of Scheduler, Control Data Base, Actuators, Sensors, and RTCDAS.

4.1. SCHEDULER

This module accepts a long-/mid-/short-term schedule for the environmental parameters setting specified by the operator. The module maintains the operator-specified schedules indicating a set of control actions to be carried out on specified locations at designated times. Execution of such control actions results in sending appropriate control commands to the RTCDAS. An example of the temperature control commands to the RTCDAS may be like:

for all plots in biome #2,
from 8:00 AM to 11:00 AM
maintain temperature at 70 degrees and humidity at 70%

4.2. DATABASE

This module maintains the history of temperature and humidity readings in biomes. Such data are collected from the distributed sensors located in different areas over an extended period of time in order to make control decision.

4.3. SENSORS and ACTUATORS

The sensors and actuators are hardware devices. There are two types of sensor and actuator pairs. A pair of a thermo-sensor and a heater is used for temperature control and a pair of a moisture-sensor and a moisturizer for humidity control. Two pairs of such devices are placed in each biome or each plot for controlling the temperature and humidity.

4.4. RTCDAS

This module collects data from sensors and issues control commands to actuators which are located in different biomes. Collected data is used to decide control actions and also sent to the DATABASE for later uses. To issue control commands, the RTCDAS compares the current values of temperature and humidity with desired values specified by the SCHEDULER.

5. Simulation Models of Hardware/Software Components

Based on the description of modules given in the previous section, we now informally specify the simulation models for SCHEDULER, DATABASE, TEMP-SENSOR, HUMID-SENSOR, and BIOME. All of these models will be implemented as concurrently runnig processes using Ada tasks for real-time simulation.

5.1. SCHEDULER Model

- Input: None
- Output: Control commands to RTCDAS
- State Variables: Ordered list of activities consisting of control actions and associated timings.

- Simulation Rule:

  Compare the scheduled time in the activity list with the current time and send appropriate control commands to the RTCDAS and update the list when the scheduled time arrives

5.2. DATABASE Model

- Input: Current values from the RTCDAS
- Output: None
- State Variables: Values of temperature and humidity
- Simulation Rule:

  When receive data from the RTCDAS, update data.

5.3. HEATER Model

- Input: On/Off commands from the RTCDAS
- Output: Temperature to the Biome
- State Variables: On/off status
- Parameter: Temperature incremental rate
- Simulation Rule:

  If heater is on, increase the temperature value by given incremental rate and report the new temperature value to Biome

5.4. MOISTURIZER Model

- Input: On/Off commands from RTCDAS
- Output: Humidity to the Biome
- State Variables: On/off status
- Parameter: Humidity incremental rate
- Simulation Rule:

  If moisturizer is on, increase the temperature value by given incremental rate and report the new humidity value to Biome

5.5. TEMP-SENSOR Model

- Input: Current temperature reading from Biome

  Request for the current temperature reading from the RTCDAS
- Output: The current temperature reading to the RTCDAS

  Request for the current temperature reading to Biome

  The current temperature reading to the DATABASE
- State Variable: None
- Simulation Rule:

  Periodically get current temperature from the Biome and report it to the DATABASE.

  When the the current temperature reading is requested from the RTCDAS, get the value from the Biome and send it to the RTCDAS.

5.6. HUMI-SENSOR Model

- Input: Current humidity reading from the Biome

  Request for the current humidity reading from the RTCDAS
- Output: Ccurrent humidity reading to the RTCDAS

  Request for the current humidity reading to the Biome

  Current humidity reading to the DATABASE
- State Variable: None
- Simulation Rule:

  Periodically get current humidity from the Biome and report it to the DATABASE.

  When the current humidity reading is requested from the RTCDAS, get the value from the Biome and send it to the RTCDAS.

5.7. BIOME Model

- Input: Heater status from HEATER

  Moisturizer status from MOISTURIZER

  Current temperature reading from TEMP-SENSOR

  Current humidity reading from HUMI-SENSOR
- Output: Current temperature reading to TEMP-SENSOR

  Current humidity reading to HUMI-SENSOR
- Sate Variables: Current temperature, Current humidity, Heater status, Moisturizer status
- Parameters: temperature decremental rate when heater is off

  humidity decremental rate when moisturizer is off
- Simulation Rule:

  When the current values of temperature and/or humidity are requested, send the current values to requester.

  When the status of devices are reported from devices, change the status of such devices.

  While the heater is off, decrement temperature by its decremental rate

  While the moisturizer is off, decrement humidity by its decremental rate

6. RTCDAS Design

We now describe design of the RTCDAS subsystem within the AIDECS. The following functional and non-functional requirements are imposed on the RTCDAS:

1. The RTCDAS decides control actions for temperature and humidity based on the desired values received by the

SCHEDULER and the current values acquired from TEMP-/HUMI-SENSORs.
2. The temperature and humidity in specified locations must be maintained within (desired value + 2) and (desired value -2).
3. The RTCDAS collects data in distributed locations through sensors and sends it to the DATABASE periodically.
4. The RTCDAS is a fully modular system; there must be no shared variable among any of the hardware or software components.
5. All of the simulation models described above must be embedded into the RTCDAS software unit for simulation in real time.

To meet the requirements above, we divide the RTCDAS into five components. They are CONTROLLER, HEATER_INTF, MOISTURIZER_INTF, TEMP-SENSOR_INTF, and HUMI-SENSOR_INTF as shown in Fig. 2. Also shown in Fig. 2 are simulation models connected to the RTCDAS. The CONTROLLER's function is to decide control actions and issue appropriate control commands to actuators so as to maintain desired values for control parameters in specified locations. To decide such control actions, it compares a desired value requested by the SCHEDULER with the current value reported by sensors. If control actions are needed, the CONTROLLER issues control commands through the HEATER_INTF and MOISTURIZER_INTF. The HEATER_INTF and MOISTURIZER_INTF work as interfaces for the HEATERs and MOISTURIZERs in distributed locations, respectively. Likewise, the TEMP-SENSOR_INTF and HUMI-SENSOR_INTF are interfaces for the TEMP-SENSORs and HUMI-SENSORs

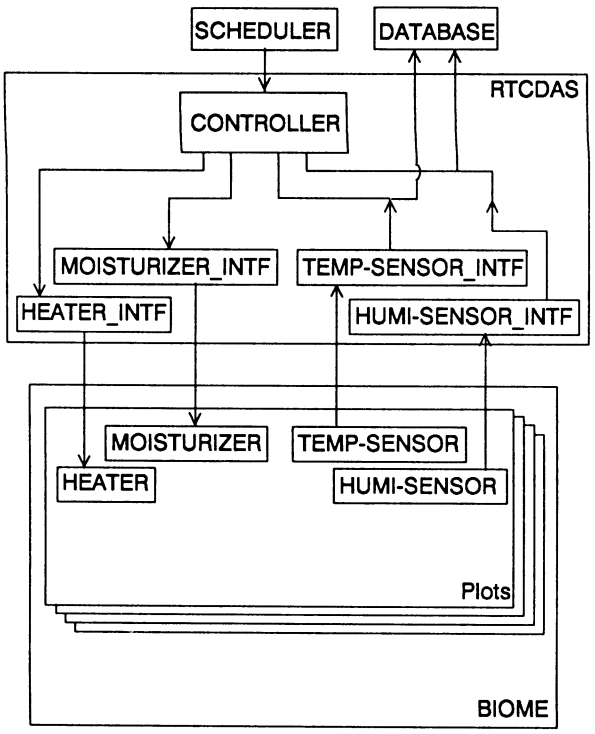

Fig. 2. Decomposition of RTCDAS.

CONTROLLER Module

- Local Variables: Desired Conditions consisting of specified locations, and set points of temperature and humidity.
- Inputs: Desired_Conditions from SCHEDULER
  Current_Temperature from TEMP-SENSOR_INTF
  Current_Humidity from HUMI-SENSOR_INTF
- Outputs: Heater_On/_Off to HEATER_INTF
  Moisturizer_On/_Off to MOISTURIZER_INTF
  Get_Temperature to TEMP-SENSOR_INTF
  Get_Humidity to HUMI-SENSOR_INTF
- Functions:

– Initialize
  Initialize the temperature and humidity for all locations with default values
– New_Setting
  When receive Desired_Condition from SCHEDULER
    Update the local variable Desired_Condition
    Send Desired_Temperature to Display
    Send Desired_Humidity to Display
– Data_Monitoring
  For every 10 seconds do the following
    Send Get_Temperature to TEMP-SENOSR_INTF to get curent temperature for all plots in a biome
    Send Get_Humidity to HUMI-SENSOR_INTF to get current humidity for all plots in a biome
– Issue_Control_Command
  When receive Current_Temperature from TEMP-SENSOR_INTF
    Compare Current_Temperature and Desired_Condition and send Heater_On or _Off commands to HEATER_INTF if needed
  When receive Current_Humidity from HUMI-SENSOR_INTF
    Compare Current_Humidity and Desired_Condition and send Moisturizer_On or _Off commands to MOISTURIZER_INTF if needed

Fig. 3. CONTROLLER Specification.

located in distributed areas. Fig. 3 gives specification of the CONTROLLER module. We include the DISPLAY Module for monitoring the operating status of the RTCDAS system in real time.

```
TASK BODY HeaterTask IS

SecondsPerUpdate : Duration := Duration(5.0);
DegreesPerUpdate : TemperatureType: Temperature-
Type(1.0);
ID : PlotNumberType;
Status : Boolean := FALSE;
LastUpdate, CurrentTime : Duration;
CurrentTemperature : TemperatureType;
DesiredTemperature : TemperatureType;

BEGIN
 ACCEPT Initialize(PlotNumber : IN PlotNumberType)
  DO
   ID := PlotNumber;
 END Initialize;
 LastUpdate := Second(Clock);
 LOOP
  SELECT
   ACCEPT HeaterOn;
   Plot(ID).HeaterOn;
   Status := True;
   LastUpdate := Second(Clock);
  OR
   ACCEPT HeaterOff;
   Plot(ID).HeaterOff;
   Status := False;
  ELSE
   Display.Synchronize;
   CurrentTime := Second(Clock);
   IF Status =True THEN
    IF CurrentTime >(LastUpdate+SecondsUpdate)
     THEN
     LastUpdate := CurrentTime;
     Plot(ID).CurrentTemperature(CurrentTempera
     ture);
     CurrentTemperature := CurrentTemperature +
     DegreesPerUpdate;
     Plot(ID).NewTemperature(CurrentTemperatur);
    END IF;
   END IF;
  END SELECT;
 END LOOP;
END HeaterTask;
```

Fig. 4. Ada Code for Heater Model.

7. Implementation and Testing

The RTCDAS has been implemented using the VAX Ada version 2.1-28 on a VAX 9000 under VAX/VMS 5.4 operating at the University of Kansas. All components shown in Fig. 2 have been implemented as tasks in Ada and communications between components was done by the Rendezvous mechanism supported by Ada. All components are implemented as event driven simulation models. They are embedded into the RTCDAS task in real time. Fig. 4 and Fig. 5 show the Ada codes for the HEATER model and the CONTROLLER module, respectively.

To test the RTCDAS, we specified various schedules for temperature and humidity in various plots through external data files. The SCHEDULER read a data file and evaluated the schedule in real time. A data file contained a list of four tuples, each of which specified location, desired temperature, desired humidity, and time when control actions take place. Since the CONTROLLER checks current values of control variables for every 10 sec, we set intervals for changing control variables to at least 30 seconds for all schedules. To observe interactions of concurrent tasks performing real-time control in different plots, all information necessary to test the RTCDAS was displayed in four windows in a screen by the DISPLAY module. A snapshot of the screen is shown in Fig. 6. The results obtained by executing various schedules indicated that the RTCDAS appeared to be working correctly. The performance of the RTCDAS in the simulated environment seemed to meet the real-time requirements we set for the software system even in a multi-user environment.

8. Concluding Remarks

This paper introduced a simulation model embedded methodology for developing a high autonomy system. We addressed the commonly encountered problem in development of a high autonomy system where we cannot apply all of the system components while the system is being developed. The proposed methodology can remedy such a problem by replacing unavailable components by real-time simulation models. We have demonstrated the viability and feasibility of our methodology by developing a real-time control system with the language, Ada. In our experiment, all of the system components excluding the control system being developed are replaced with simulated models. Simulated models include both software and hardware units.

```
TASK BODY ControllerTask IS

TYPE DesiredConditionsType IS RECORD
                                Temperature : TemperatureType;
                                Humidity : HumidityType;
END RECORD;

SecondsPerSample : Duration := Duration(10.0);
DesiredConditions : ARRAY (1..NumberOfPlots) OF DesiredConditionsType;
CurrentTemperature : TemperatureType;
CurrentHumidity : HumidityType;
LastUpdate, CurrentTime : Duration;

BEGIN
  ACCEPT Initialize;
  LastUpdate := Seconds(Clock);
  FOR i IN 1..NumberOfPlots LOOP
    DesiredConditions(i).Temperature := TemperatureType(70);
    DesiredConditions(i).Humidity := HumidityType(70);
    Display.DesiredTemperature(i,DesiredConditions(i).Temperature);
    Display.DesiredHumidity(i,DesiredConditions(i).Humidity);
  END LOOP;
  LOOP
    SELECT
      ACCEPT NewSetting(PlotNumber : IN PlotNumberType;
                        Temperature : IN TemperatureType;
                        Humidity : IN HumidityType) DO
        DesiredConditions(PlotNumber).Temperature := Temperature;
        DesiredConditions(PlotNumber).Humidity := Humidity;
        Display.DesiredTemperature(PlotNumber,Temperature);
        Display.DesiredHumidity(PlotNumber,Humidity);
      END NewSetting;
    ELSE -- Monitoring and Issuing Control Commands
      Display.Synchronize;
      CurrentTime := Seconds(Clock);
      IF CurrentTime(LastUpdate + SecondsPerSample) THEN
        LastUpdate := CurrentTime;
        FOR i IN 1..NumberOfPlots LOOP
          TemperatureSensor(i).GetTemperature(CurrentTemperature);
          HumiditySensor (i).GetHumidity(CurrentHumidity);
          Display.CurrentTemperature(i,CurrentTemperature);
          Display.CurrentHumidity(i,CurrentHumidity);
          IF CurrentTemperature(DesiredConditions(i).Temperature + 2.0) THEN
            HeaterActuator(i).HeaterOff;
            Display.HeaterOff(i);
          END IF;
          IF CurrentTemperature(DesiredConditions(i).Temperature - 2.0) THEN
            HeaterActuator(i).HeaterOn;
            Display.HeaterOn(i);
          END IF;
          IF CurrentHumidity(DesiredConditions(i).Humidity + 2.0) THEN
            MoisturizerActuator(i).MoisturizerOff;
            Display.MoisturizerOff(i);
          END IF;
          IF CurrentHumidity(DesiredConditions(i).Humidity - 2.0) THEN
            MoisturizerActuator(i).MoisturizerOn;
            Display.MoisturizerOn(i);
          END IF;
        END LOOP;
      END IF;
    END SELECT;
  END LOOP;
END ControllerTask;
```

Fig. 5. Ada Code for CONTROLLER Module.

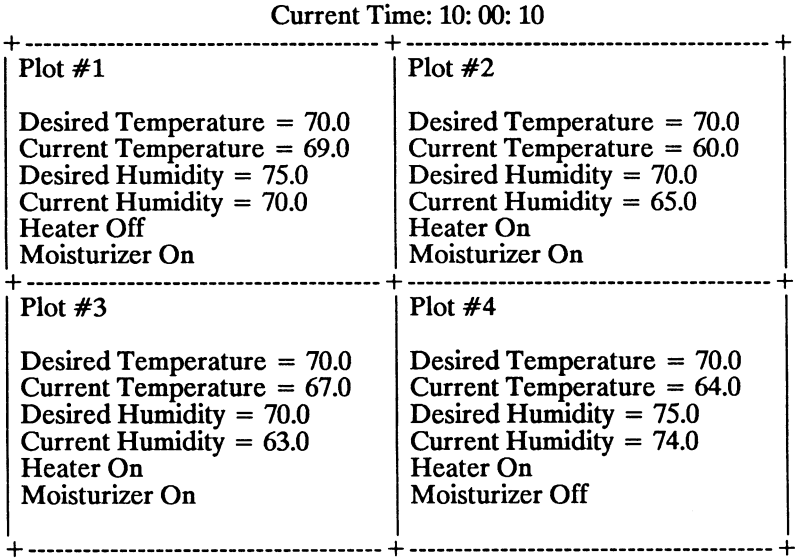

Fig. 6. Screen Snapshot for Monitroing RTCDAS.

The process-based view is very appropriate in examining real-time systems because of their event driven and inherently embedded nature. Hence the simulation based approach fits very naturally into the framework of real-time system development process. Use of simulation modeling in real-time system development is discussed in [3]. However, their approach limits the application of simulation modeling only to hardware components at the testing phase. We believe that the simulation modeling methodology can be applied in much wider scope throughout the development process. We need not view software and hardware components separately as they can all be characterized and analyzed in the uniform framework of the simulation modeling methodology. Above all, since the simulation modeling approach is backed by formal principles, we can introduce a mathematically sound, systematic methodology for real-time system development process. With the proposed methodology, a theoretical framework may be built to improve the current technology of real-time system development. Our future research will focus on such a theoretical framework for real-time system development process.

Acknowledgements

The authors thank Michael Kinney for implementation of the RTCDAS and Eric Schmitendorf for his assistance in preparing the manuscript.

References

[1] B. P. Zeigler, "High Autonomy Systems: Concepts and Models," *Proc of AI, Simulation and Planning in High Autonomy Systems*, New York: NY, IEEE Computer Society Press, 1990, pp. 2-7.

[2] T. G. Kim and B. P. Ziegler, "An AI-Based, Distributed Control System for Self-Sustaining Habitats," *Artificial Intelligence in Engineering*, Vol. 5, No. 1, Jan 1990, pp. 33-42.

[3] A. Burns and A. Wellings, *Real-Time Systems and Their Programming Languages* Addison Wesley, 1990.