2002-09

SEA Environment for CARA Software

Luqi; Shing, M.; Berzins, V.; Puett, J.; Guan, Z.; Qiao, Y.; Zhang, L.; Chaki, N.; Liang, X.; Ray, W....

Monterey, California. Naval Postgraduate School

NPS-SW-02-004

NAVAL POSTGRADUATE SCHOOL
Monterey, California

SEA Environment for CARA Software

Technical Report (7/1/2002 – 09/30/2002)

By

Luqi, M. Shing, V. Berzins, J. Puett, Z. Guan, Y. Qiao, L.Zhang,
N. Chaki, X. Liang, W. Ray, M. Brown, D. Floodeen, D. Drusinsky

September 2002

Prepared for: U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

RADM David R. Ellison
Superintendent

Richard S. Elster
Provost

This report was prepared for and funded in part by the U.S. Army Research Office.

This report was prepared by:

Luqi
Professor, Computer Science

Reviewed by:

Luqi
Director, Software Engineering
Automation Center

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	9/30/2002	Technical Report 7/1/2002 – 09/30/2002

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
SEA Environment for CARA Software	40473-MA-SP
6. AUTHOR(S) Luqi, M. Shing, V. Berzins, J. Puett, Z. Guan, Y. Qiao, L.Zhang, N. Chaki, X. Liang, W. Ray, M. Brown, D. Floodeen, D. Drusinsky	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Automation Center, Naval Postgraduate School, Monterey, CA 93943	8. PERFORMING ORGANIZATION REPORT NUMBER NPS-SW-02-004
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U. S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12 a. DISTRIBUTION / AVAILABILITY STATEMENT	12 b. DISTRIBUTION CODE
Approved for public release; distribution unlimited.	

13. ABSTRACT (Maximum 200 words)

This report summarizes our prototyping effort for the Infusion Pump Computer Assisted Resuscitation Algorithm (CARA) software. Computer aided prototyping (CAP) shows promise that one system under development frees designers from implementation details by executing specifications via reusable components. In constructing distributed embedded real-time systems, the formal specifications can facilitate recording and enforcing timing constraints by providing underlying compositional architecture, restricted real-time scheduling and reusable execution base. This effort takes the computer assisted resuscitation algorithm (CARA) software for the infusion pump as the studying case, and presents different versions that model the CARA with specific focuses, such as simplicity of the design, safety-aspects, requirements coverage, enabling architecture, and so forth. Finally the evaluation on CARA requirements and on the language for prototyping complex systems and its associated tools is also provided.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Rapid Prototyping, Embedded Real-time Systems, Safety Critical Software, Infusion Pump Control	30
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION ON THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	**UNCLASSIFIED**	**UNCLASSIFIED**	**UL**

NSN 7540-01-280-5500

Standard Form 298 (Rev.2-89)
Prescribed by ANSI Std. 239-18
298-102

List of Contents

SEA[1] Environment for CARA[2] Software

9/30/2002

Luqi, M. Shing, V. Berzins, J. Puett, Z.Guan, Y. Qiao, L. Zhang,
N. Chaki, X.Liang, W.Ray, M. Brown, D. Floodeen

Abstract

Computer aided prototyping (CAP) shows promise in system development. It frees designers from implementation details by executing specifications via reusable components. In constructing distributed embedded real-time systems, the formal specifications can facilitate recording and enforcing timing constraints by providing underlying compositional architecture, restricted real-time scheduling and reusable execution base. This research takes the computer assisted resuscitation algorithm (CARA) software for the infusion pump as the studying case, and presents different versions that model the CARA with specific focuses, such as simplicity of the design, safety-aspects, requirements coverage, enabling architecture, and so forth. Finally the evaluation on CARA requirements and on the language for prototyping complex systems and its associated tools is also provided.

Key word: Rapid prototyping, System modeling and design, Specification and software tools support

1. Introduction

This report summarizes our prototyping effort for the Infusion Pump Computer Assisted Resuscitation Algorithm (CARA) software. The purposes of this effort were to

- Study the characteristics of high confidence embedded systems;
- Explore design alternatives using our tools;
- Validate the feasibility and effectiveness of our models, languages and tools;
- Demonstrate the flexibility of the tools in generating and comparing variations of software design given the same requirement set, and
- To set the focus for our FY2003 efforts for improving our models, languages and tools to better address high confidence embedded systems.

The CARA software automates the delivery of intravenous (IV) fluids to a trauma patient by monitoring the patient's blood pressure and controlling the output rate of an IV infusion pump. This is a typical example of a high confidence embedded system. Currently high confidence is a subjective concept. We would like to refine this concept to relate it to objective measurement, especially for safety-related aspects.

Our prototyping effort explored several design alternatives using our tools. It is well known that requirements faults account for a majority of software failures, particularly for first-of-a-kind applications. We explored the effectiveness of parallel conceptualization efforts to expose potential requirements issues.

We tested the feasibility of our tools and new interface, and explored the degree to which our design notations and models express the range of issues typical of high confidence embedded systems. A particular focus was to identify issues difficult to express in our current models and representations.

In FY2003 we plan to refine our languages, models, and tools to respond to specific deficiencies identified in the initial CARA prototyping effort, and to use the improved tools to establish a measurable basis for high confidence embedded systems.

---

[1] A set of Computer Aided Software Engineering Automation Tools

[2] Computer Assisted Resuscitation Algorithm

2. Language for Analysis, Modeling and Prototyping of Complex Systems (LAMPS)

Current and future DoD software systems fall somewhere within a continuous spectrum between pure information systems and pure control systems. All of these systems support the warfighter in one way or another, whether they are CONUS warehouse inventory tracking systems or the embedded software on a smart projectile in a theater of war. These systems can be distributed, heterogeneous and network-based, consisting of a set of components running on different platforms and working together via multiple communication links and protocols. More and more of these systems have critical safety aspects and associated needs for high confidence. These systems have many safety aspects and associated needs for high confidence embedded systems. Hence, we must develop models and languages to capture the requirements and attributes. We built upon our experience with specification[3] and prototyping[4] languages and developed a language (LAMPS) for modeling and prototyping complex systems. This section briefly summarizes our notations for high confidence embedded systems, which were used for the CARA prototyping effort.

LAMPS models a system as a set of concurrent state machines communicating via shared data streams. Mathematically, a system S can be represented as a 4-tuple $S = [S, E, C, D]$ where

$S = \{s_i, 1 \le i \le |S|\}$, where s_i is a component system

$E = \{e_{ij}, 1 \le i, j \le |S|\}$, where e_{ij} is the set interactions from s_i to s_j

$C = \{c(s_i), 1 \le i \le |S|\}$, where $c(s_i)$ is the set of constraints on s_i

$D = \{d(e_{ij}), 1 \le i, j \le |S|\}$, where $d(e_{ij})$ is the set constraints on the interactions from s_i to s_j

2.1 Basic Computation Graph Model

Pictorially, we can represent each s_i as a vertex of a computation graph and each e_{ij} as a set of directed edges from s_i to s_j in the graph. For example, the system shown in Figure 1 corresponds to a system with

S = {*monitor_environment, temperature_control, valve_control*} , and
E = {{*temperature, humidity*}, {*valve_adjustment*}, {*fuel*}}.

The component *monitor_environment* is associated with the set of constraints {*Maximum Execution Time* = 100 ms, *PERIOD* = 500 ms, *Mean_Time_Between_Failure* = 48 hours}, and the edge *temperature* is associated with the constraints {*LATENCY* = 500 ms, security = medium}.

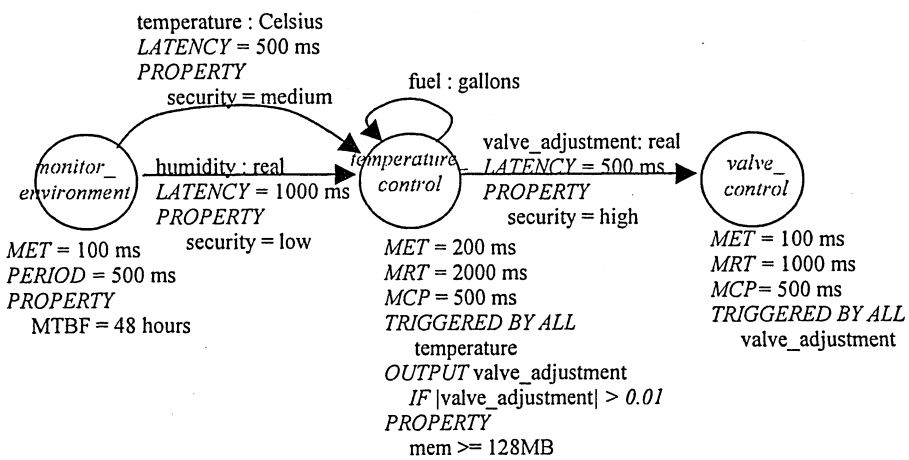

Figure 1. The LAMPS model for System Requirements

---

[3] V. Berzins and Luqi, "An introduction to the specification language SPEC", *IEEE Software*, 7(2), March 1990, pp. 74 -84

[4] Luqi, V. Berzins and R. Yeh, "A prototyping language for real-time software", *IEEE Trans. Software Engineering*, 14(10), Oct. 1988, pp. 1409 -1423

To support the automated generation of glue and wrapper code, LAMPS also provides the capability to capture the attributes of target network systems. For example, Figure 2 shows a target network connecting 3 host machines.

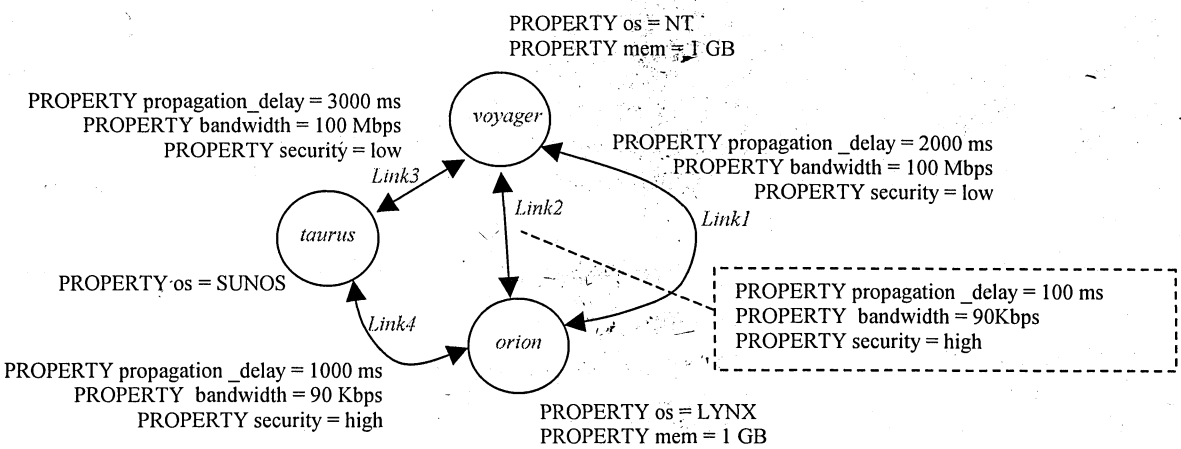

Figure 2. The LAMPS model for target network

2.2 Hierarchical Computation Graph Model

The hierarchical graph model is introduced to support abstraction. A vertex s_i in a hierarchical computation graph may in turn be modeled as a computation graph $[S_i, E_i, C_i, D_i]$ resulting in a hierarchical structure of nested computation graphs. For example, the *monitor_environment* vertex in Figure 1 may be modeled as the graph shown in Figure 3. Note that the children vertices (*monitor_temperature* and *monitor_humidity*) inherit the constraints associated with parent (*monitor_environment*) vertex. One can always represent a given static hierarchical computation graph by an equivalent basic static graph, however, care must be taken to ensure that such transformation does not introduce inconsistencies to the resultant constraints.

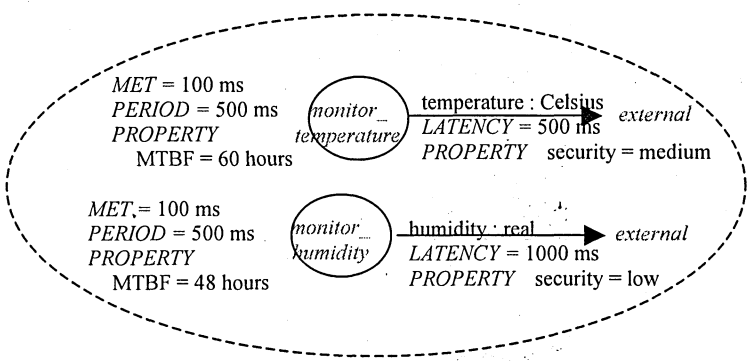

Figure 3. Decomposition for the *monitor_environment* vertex

3. Tools for Rapid Automated Prototyping of Complex Systems (SEATools)

We developed a set of tools to support the modeling, analysis and prototyping of the systems under development (Figure 4). The tools set provides a system model editor for the users to create and modify their system models, a translator to check the syntax/semantics of the system model and to generate glue and wrapper codes to realize the design for the target system architecture, and a scheduler to analyze the timing constraints and to generate code to realize these constraints in the target architecture. The tool interface also provides menus for users to manage their projects and compile the source codes into an executable prototype.

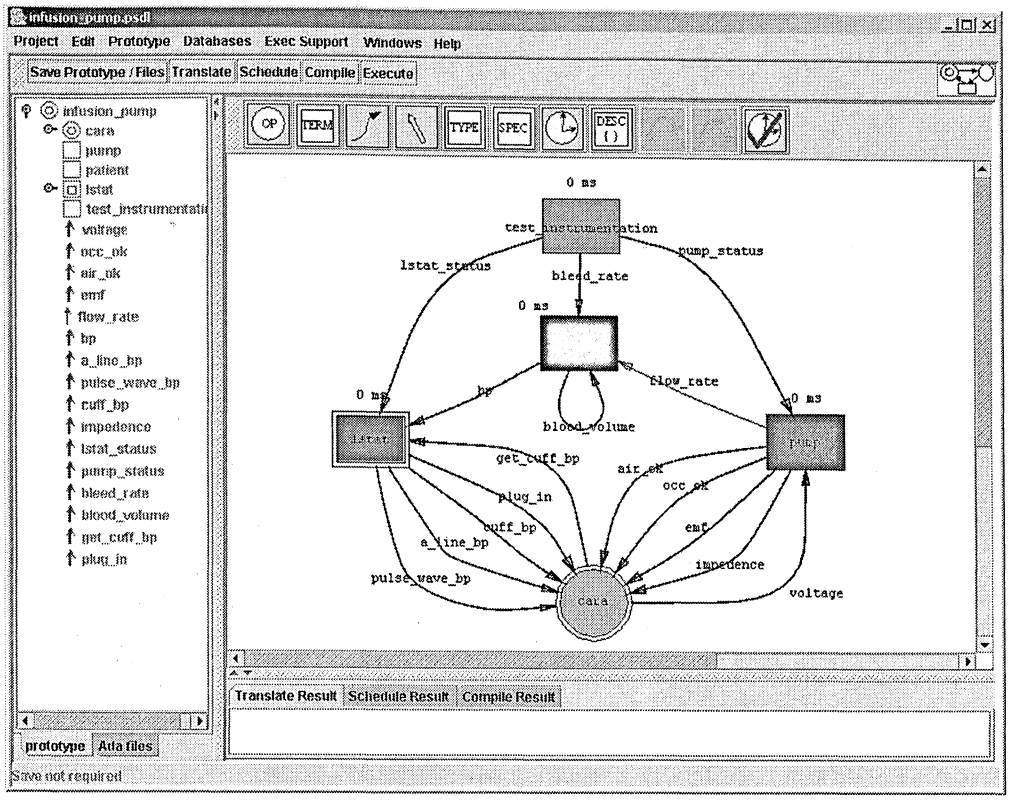

Figure 4. The SEATools Interface

4. Designs of the Infusion Pump Control Software

To evaluate the effectiveness of the language and tools, we formed five design teams and used LAMPS and SEATools to analyze the requirements of the Infusion Pump Computer Assisted Resuscitation Algorithm (CARA) software. The goal of the CARA Infusion Pump prototyping effort was develop an executable model to facilitate the analysis and understanding of the requirements (especially with respect to timing and safety) of the Computer Assisted Resuscitation Algorithm (CARA) software. This CARA software automates the delivery of intravenous (IV) fluids to a patient by monitoring the patient's blood pressure and controls the IV output rate of an infusion pump. Expected interactions between the CARA software and its environment are described informally in the following documents from WRAIR Dept. of Resuscitative Medicine:

"Hazard Analysis and Standard Operating Procedure", July 1999.
"Narrative Description of the CARA software", Jan 2001.
"CARA Pump Control Software Questions", Version 6.1, Jan 2001.
"CARA Tagged Requirements", Increment 3, Version 1.2, March 2001.

4.1 Design Model #1

The overall system environment consists of just three main components: The LSTAT stretcher is assumed to provide the majority of patient related information (e.g. blood pressures), the infusion pump is the main item for control, and the CARA Software System is the system driving the infusion pump based on data received from the LSTAT (Figure 5).

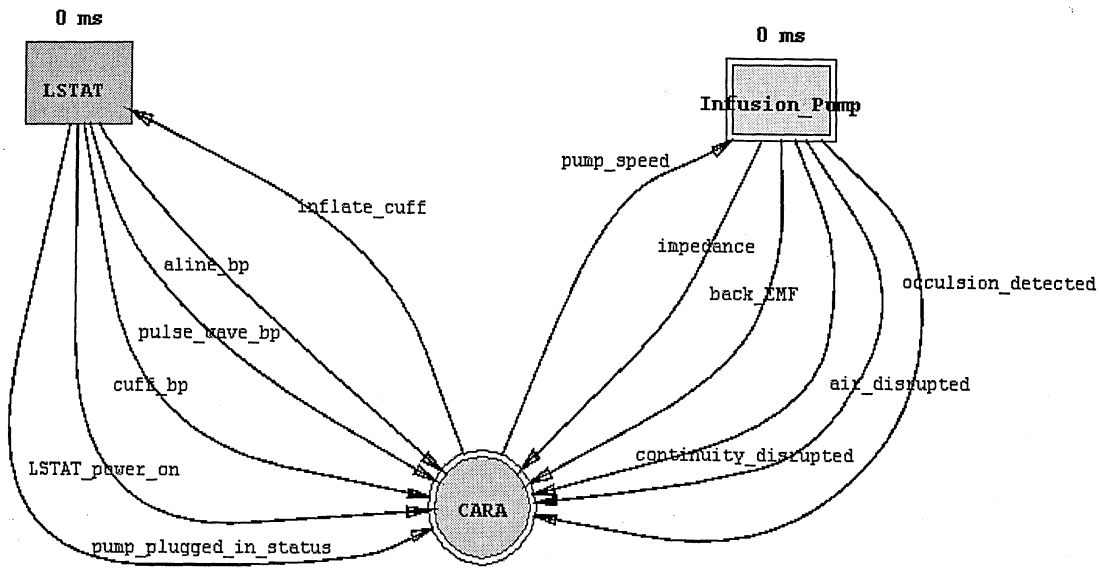

Figure 5. The top level of the CARA Software Model #1

Figure 6 shows the decomposition of the Infusion Pump. There is an eight-pin ribbon cable between the pump and the CARA. Eight of the terminators in this level correspond to the pins of the ribbon cable -- only 5 of which actually interact with the CARA. The requirements describe the need for continuity checking of all the pins via a hardware interlock -- this is modeled as an additional terminator (continuity_interlock).

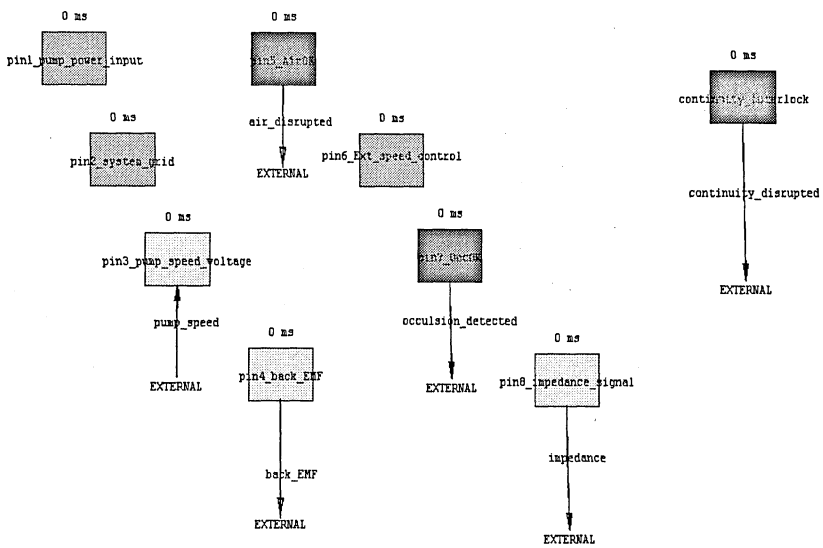

Figure 6. The Infusion Pump module of the CARA Software Model #1

Figure 7 shows the three main modules of the CARA software: a Pump_Control_Module, an IO_Module and a Management_Module. The Pump_Contol Module resolves an accurate blood pressure from various sources and then determines the appropriate flow rate for the pump. The IO_Module sends and receives inputs to the CARA

operator via the display, and the Management_Module monitors the status of the pump, the lines, and the system for logging data into the historical resuscitation file.

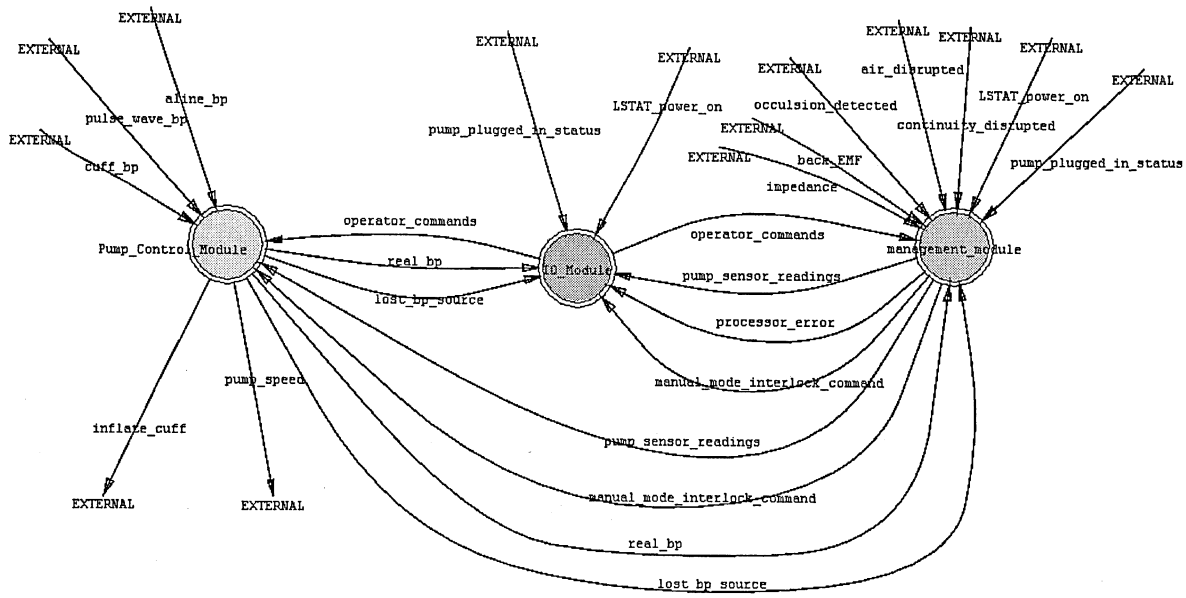

Figure 7. The CARA software module of Model #1

Figure 8 shows the Management_Module inside the CARA software. Central to this module is a manual_mode_interlock operator that is primarily responsible for returning the system to a manual mode in case of any component failure. Feeding this operator is data from the line_monitor which monitors the sensor readings from the pump and LSTAT. Also feeding the manual_mode_interlock is a processor watchdog. This watchdog is implemented on a separate processor and will sense any failure of the main processor and alert the operator (via the display). One final element of the management_module is the Resuscitation_File where all information about the changing state of the CARA system is recorded.

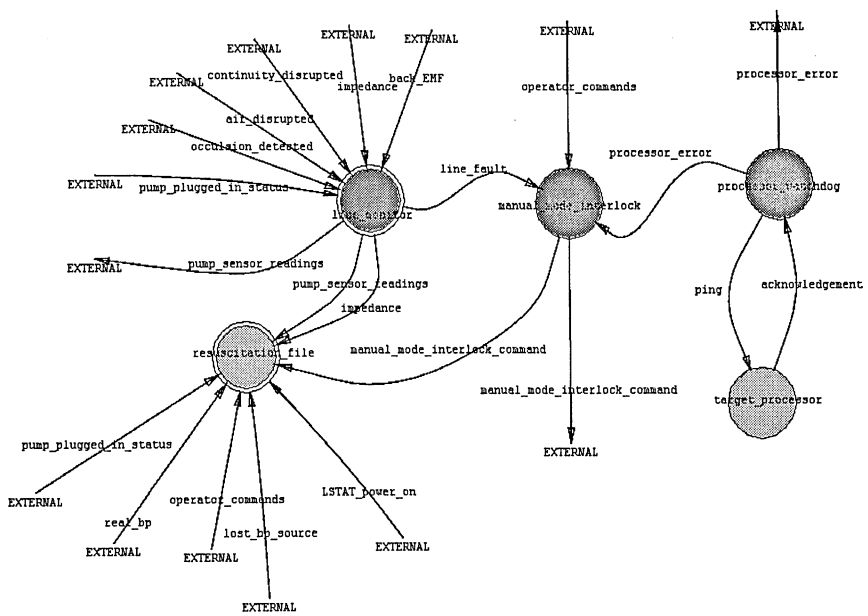

Figure 8. The Management module of Model #1

The resuscitation_file module consists of a series of operators that accept data of a particular type and convert it into data_for_file format. The resuscitation file itself is modeled as a looping data stream where additional entries are appended onto the end of the file. The line_monitor module (Figure 9) monitors sensor readings coming from the pump. In case of any problems, a line_fault is generated and immediately sent to the manual_mode_interlock. Determining the proper values of impedance, air, and EMF readings requires further decomposition of these operators.

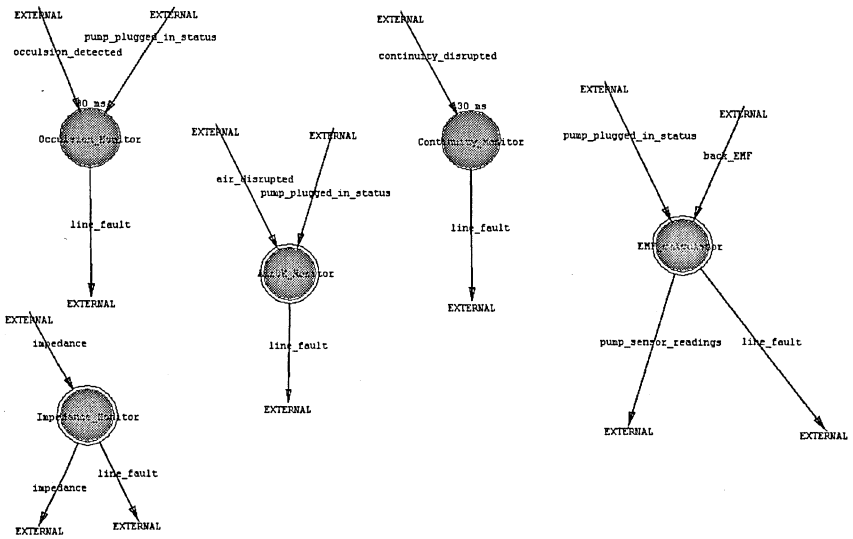

Figure 9. The Line Monitor module of Model #1

The AirOK_Monitor consists of two operators, a Start_Air_Timer and a Generate_Air_Fault_Operator. The Start_Air_Timer monitors for an air disruption in the air line. If it receives input indicating there is an air disruption, it waits an allotted time to see if the disruption clears. If disruption clears, the timer is reset. If it does not, the Generate_Air_Fault_Operator fires indicating a line_fault.

The EMF_calculator module performs two major functions. First it performs EMF polling to accumulate appropriate EMF values to calculate both the infuse rate and the total volume infused. These values are later sent to the display and the resuscitation file. It also monitors for absent or out of tolerance EMF values. If acceptable EMF values are not received within an appropriate time, it generates a line_fault that eventually sets the system back into manual mode.

The Impedance_Monitor module performs two major functions, much like the EMF calculator. First it performs impedance polling to accumulate appropriate impedance values to send to the resuscitation file. It also monitors for absent or out of tolerance impedance values. If acceptable impedance values are not received within an appropriate time, it generates a line_fault that will also set the system back into manual mode.

Figure 10 shows the Pump Control module. This is the major safety critical module of the CARA. It is responsible for resolving an accurate blood pressure reading from the various input sources and then using this blood pressure to determine the correct input to the pump when the system is in auto-control mode. Because of the safety critical nature of this module, we chose to implement this with Triple Modular Redundancy (TMR). This specific safety architecture was not called for explicitly in the requirement statement; however, the safety environment implicitly requires some form of redundancy to ensure that the proper commands are sent to the pump. The TMR architecture uses three concurrent modules performing similar functions and producing similar output, but using different internal algorithms in their calculations. A voting element is then responsible for determining which output to use. Module1 is the only module of the three that has been fully decomposed. Modules 2 & 3 could be decomposed similarly to Module 1 but would use different algorithms (inserted at the programming stage).

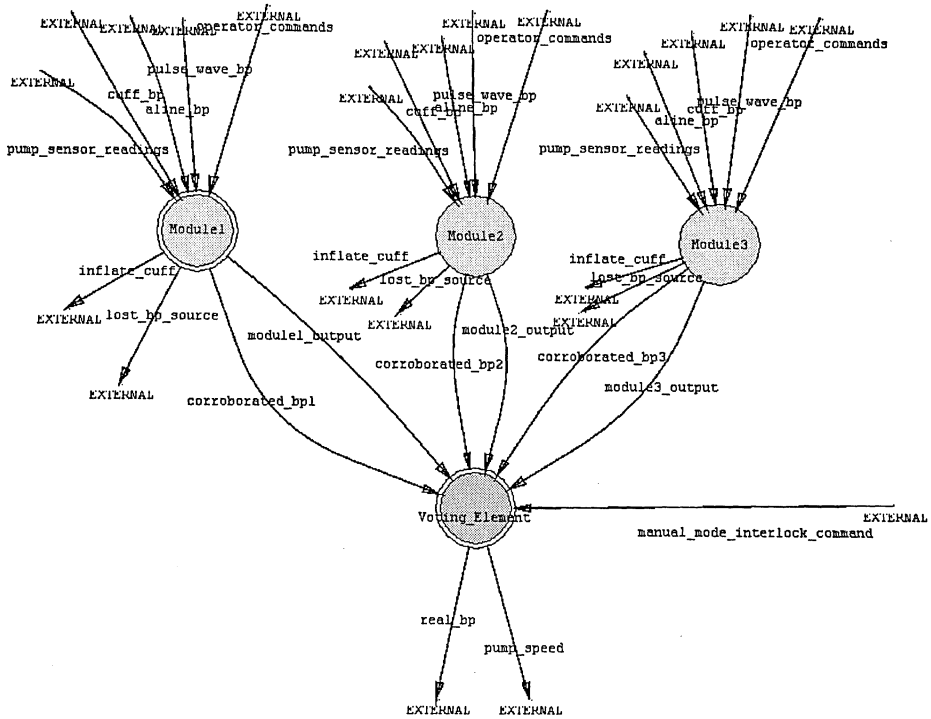

Figure 10. The Pump Control module of Model #1

Module1 is decomposed into two main functions: first, a blood pressure calculator responsible for determining which blood pressure to use during further calculations, and second, a pump speed calculator which determines the appropriate pump command to issue (Figure 11).

The BP_calculator module (Figure 12) has been decomposed into three operators: first, the Aline_Corroborator which attempts to generate a corroborated arterial line blood pressure for future calculation; second, the Pulse_Wave_Corroborator which attempts to generate a corroborated pulse wave blood pressure for future calculation; and finally, a BP_Priority_Calculator which given the blood pressures available (arterial line, pulse wave, and cuff) determines the blood pressure that will be used (corroborated_bp1) based on a blood pressure priority scheme.

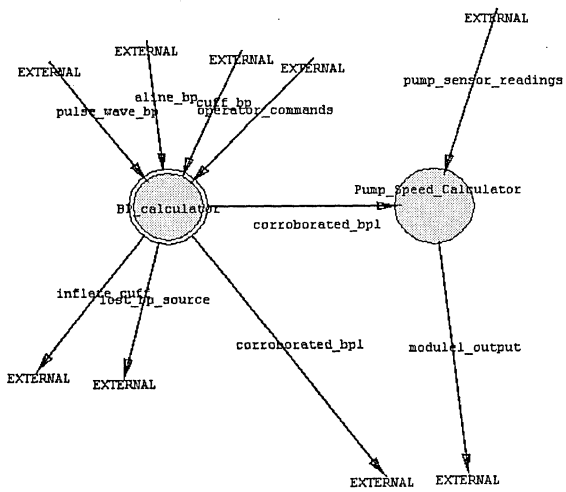

Figure 11. The Module1 module of Model #1

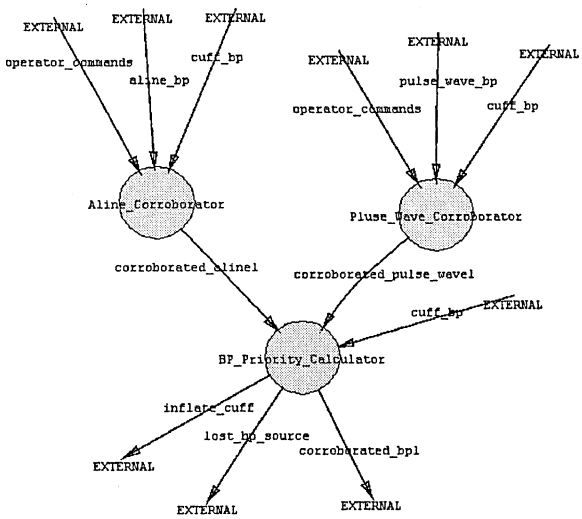

Figure 12. The BP_Calculator module of Model #1

Recall that there are two other concurrent modules (modules 2 & 3) that are performing similar tasks. The voting_element is decomposed into two main sub-operators. First, the Vote operator compares the inputs from Modules 1, 2, 3. If the all the data are within a set tolerance of each other, the operator averages the values and outputs the real_bp (for the display and the resuscitation file) and the pump_speed. If two of the inputs are within tolerance and one is outside tolerance, it disregards the value outside tolerance, averages the other two and outputs the averages. If all three values are outside tolerance, the vote operator disregards all three values and waits for satisfactory data. The Terminate_Autocontrol operator is responsible for returning the system to manual control if the manual_mode_interlock_command is invoked.

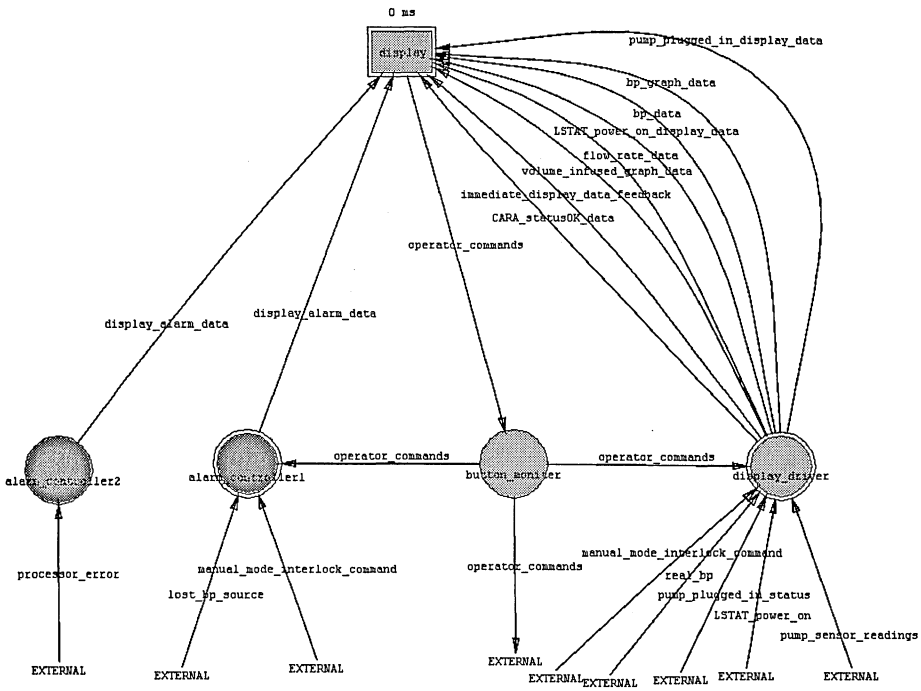

Figure 13. The IO module of Model #1

Figure 13 shows the IO_Module. This module handles the input and output to the CARA display for the benefit of the CARA operator. Note that unlike version 1, in this version the CARA Display has been modeled as a decomposable terminator within this module. The alarm functions have been separated from other display functions to help isolate the safety critical functions. Also note the duplicate alarm_controller. This additional alarm controller would be implemented on the second processor with the main processor watchdog. In case of main processor failure, alarms would be raised to the display.

The alarm_controller1 accepts alarm generating data streams and outputs alarm data (both text and audio alarm) to the CARA display. As previously mentioned, the second controller (alarm_controller2) is implemented on the second processor to enable alarm data to be transmitted to the display in case of main processor failure. The display_driver contains a series of operators that act as drivers for the information displayed on the CARA display. Some of these operators (the graph operators in particular) maintain aggregate data to send to the display. Also several of the operators function by comparing any new data to old data and only update the display in the case of changes. The display module is a decomposed terminator that simulates the functions of the CARA display. Each terminator represents separate sets data that can be displayed on the Display. This module also simulates user input by way of operator_commands (button pushes) from the Display.

4.2 Design Model #2

This design for the infusion pump prototype has four layers. The top level gives the outline of the infusion pump. It includes the CARA, pump, LSTAT, pressure_gauge and patient (Figure 14). The pressure_guage module is made up of three atomic operators, each responsible for monitoring one of the three blood pressure sources (Figure 15).

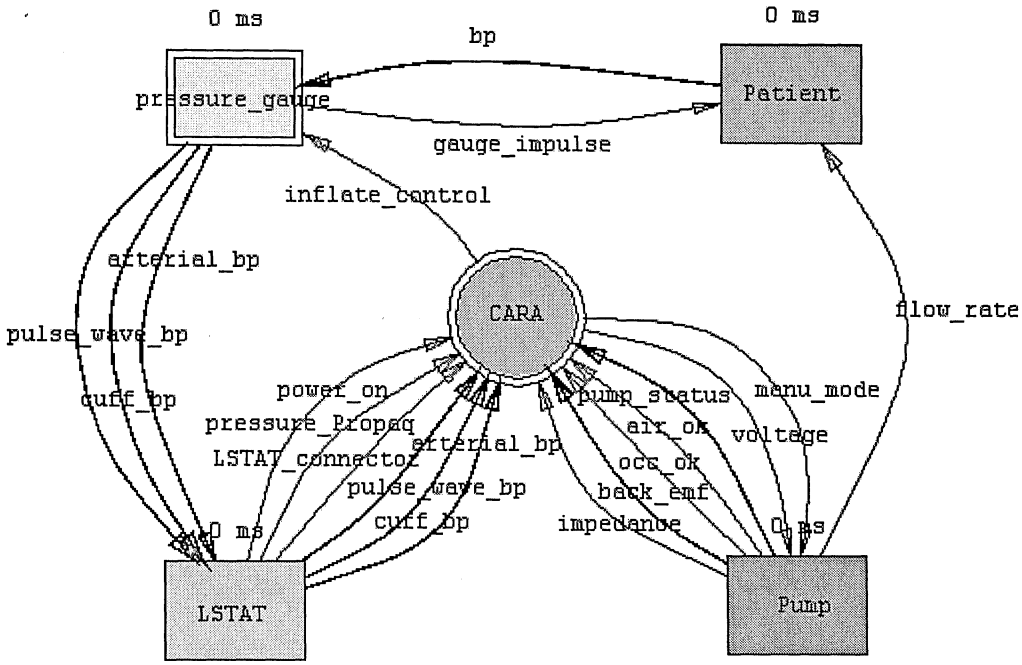

Figure 14. The top level design of Model #2

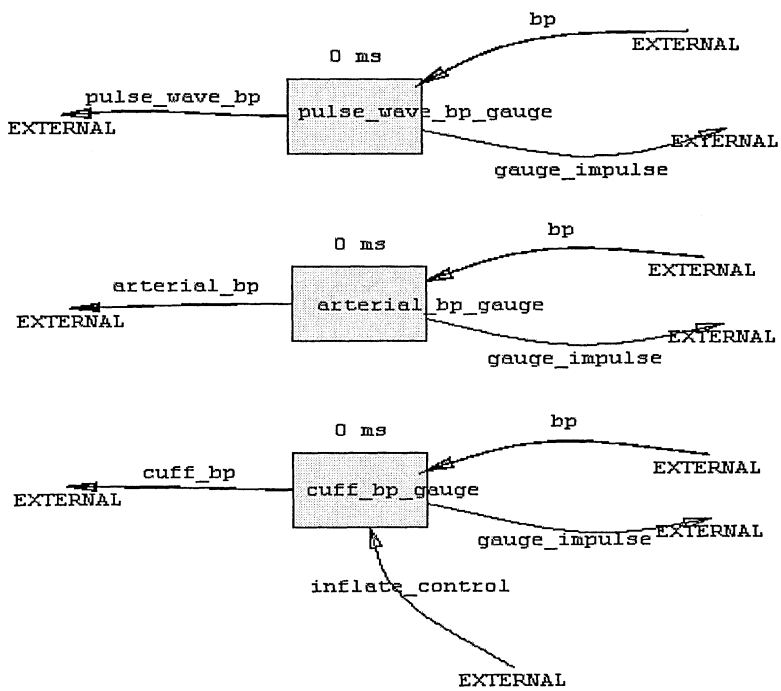

Figure 15. The Pressure_Guage module of Model #2

The CARA module is the central part of the entire system. It can be subdivided into a monitor module, a control algorithm module, a display and alarm process module and a log module (Figure 16).

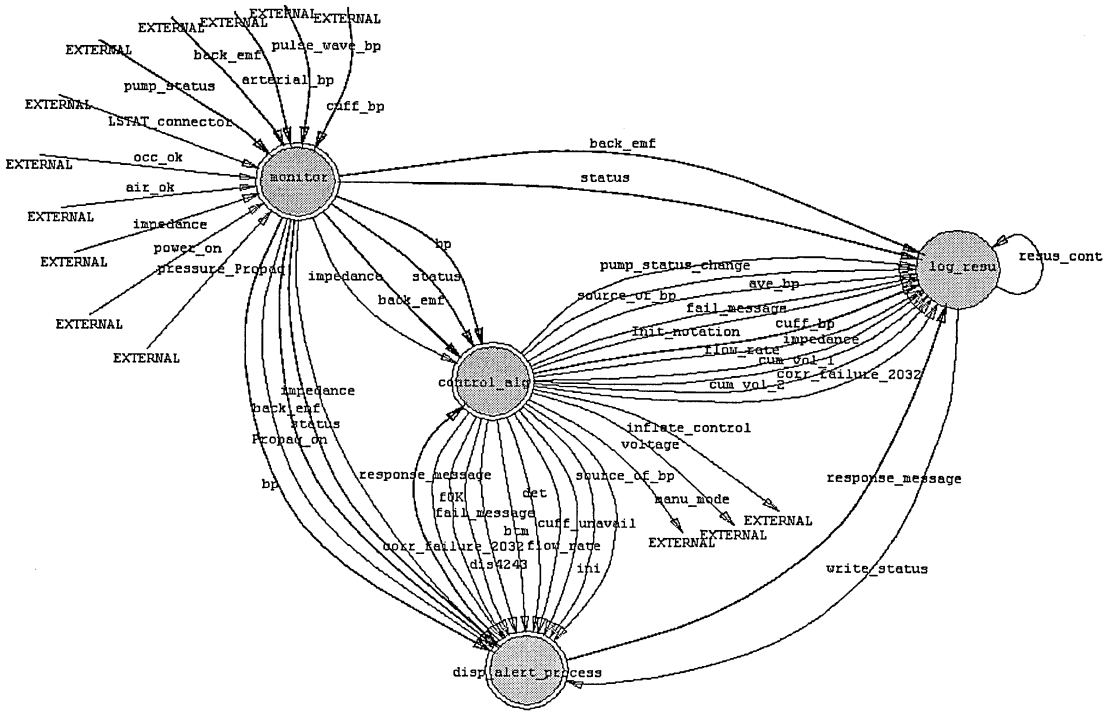

Figure 16. The CARA module of Model #2

The monitor module monitors the signal from the pump and LSTAT to provide the CARA with pump, LSTAT and blood pressure information (Figure 17).

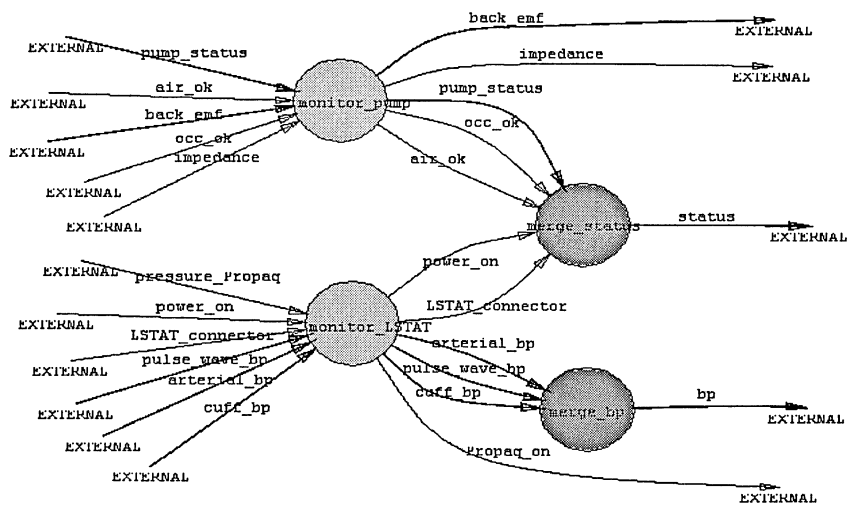

Figure 17. The Monitor module of Model #2

The display and alarm process module (Figure 18) can process an alarm and display a corresponding message according to the input data stream while the log module will manage writing the appropriate information to the resuscitation file.

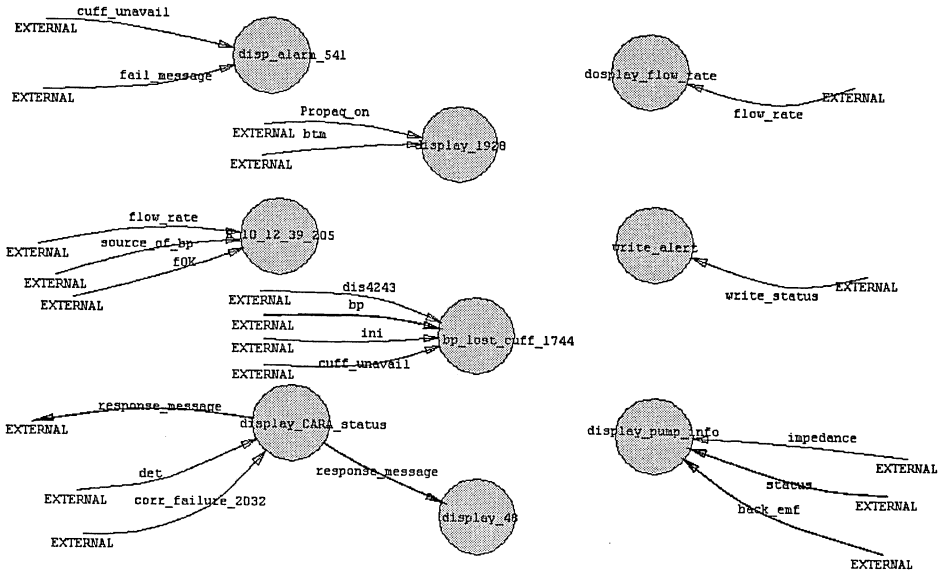

Figure 18. The Display and Alarm Process module of Model #2

The control algorithm module is the main part of the CARA (Figure 19). It is responsible for calculating the flow rate and the cumulative volume of the pump according to the signal from back EMF and providing the voltage to control the pump flow rate. The algorithm also performs several other functions on the monitored signals. The detailed process is included in the two lower layers (Figures 20, 21).

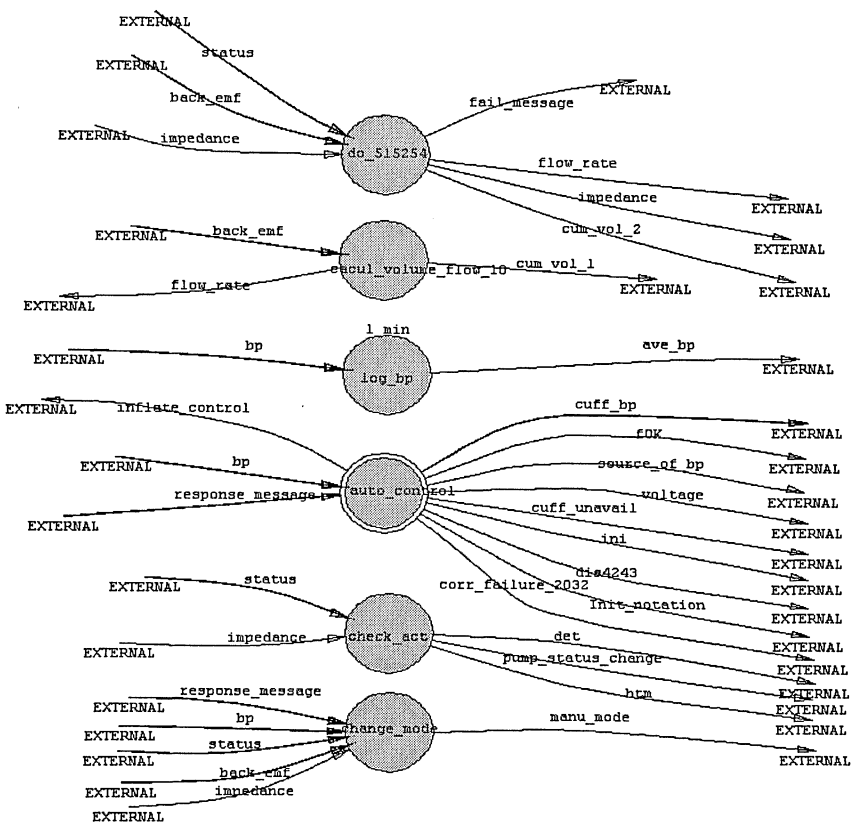

Figure 19. The Control Algorithm module of Model #2

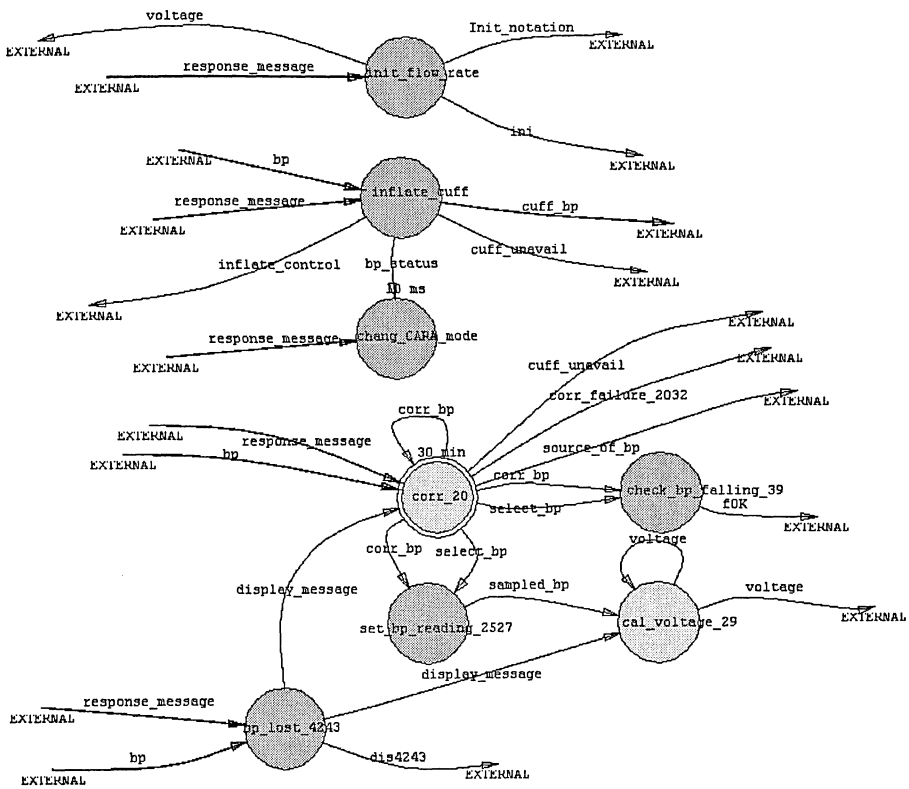

Figure 20. The Auto_Control module of Model #2

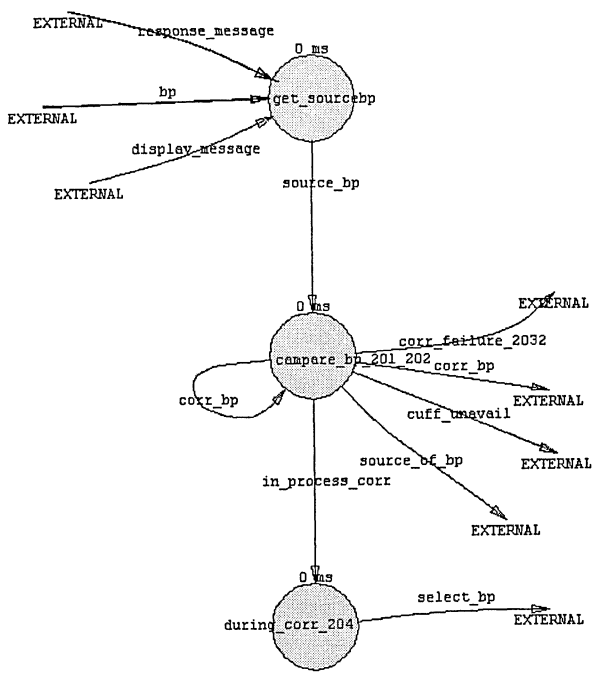

Figure 21. The Corroboration (corr_20) module of Model #2

4.3 Design Model #3

In this model, the CARA software is split into two processes (Figure 22). While most of the controlling operation and computation is encapsulated in the CARA_algorithm operator, the CARA_interaction operator is responsible for information display control, alarm signal management, recording in the resuscitation file and handling of operator over-ridings. The overall architecture assumes two external sources, pump and LSTAT_patient, which provide data to the CARA and receive control information.

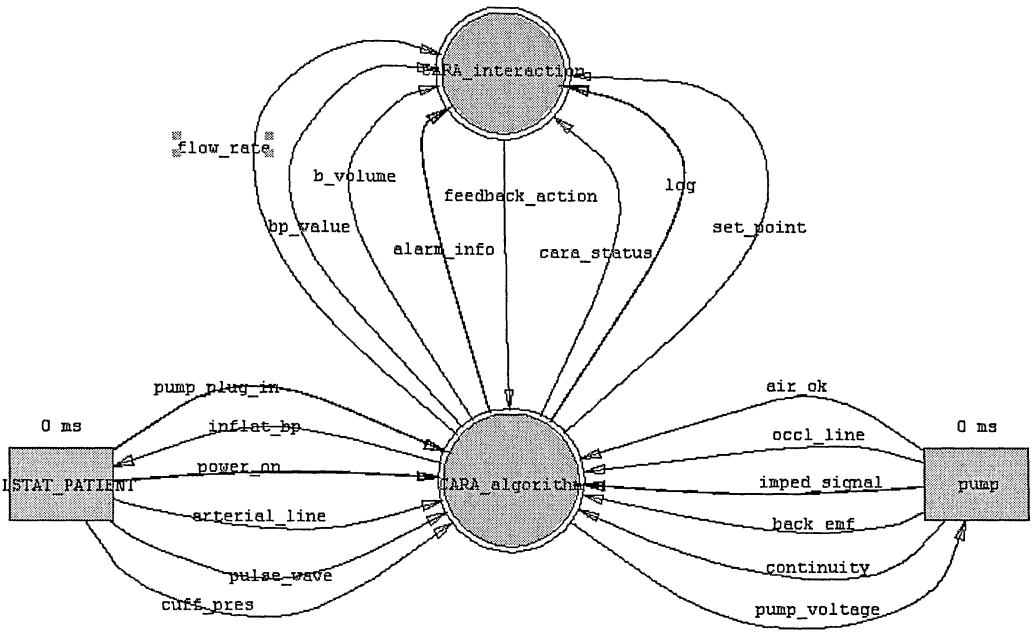

Figure 22. The top level design of Model #3

Figure 23 shows the decomposition of the CARA_algorithm module. It consists of six components. The sub-modules, power_monitor, pump_monitor, bp_monitor and bp_corroborate, are responsible for monitoring and validating the power on/off of the LSTAT, the connections and status of the signals from the pump, and the signal and status of the blood pressure from the LSTAT. Information derived from these monitors is fed into the other two modules, pump_manual_control and pump_auto_control. They are responsible for monitoring and controlling the infusion pump under two different modes of operations along with other important tasks like validating and corroborating blood pressure, switching control modes from automatic to manual and vise versa, etc. The sub-modules pump_monitor, bp_monitor, pump_auto_control, bp_corroborate and pump_manual_control are in turn made up of their own decompositions shown in Figures 24-28.

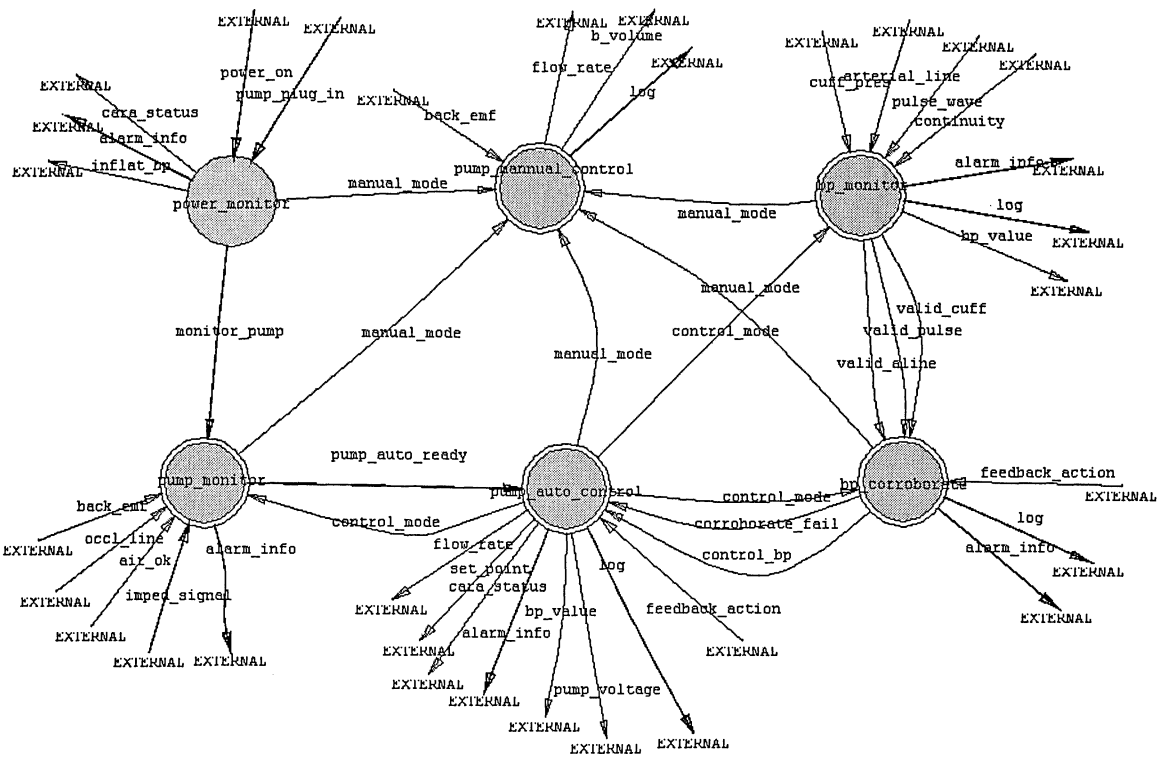

Figure 23. The CARA_Algorithm module of Model #3

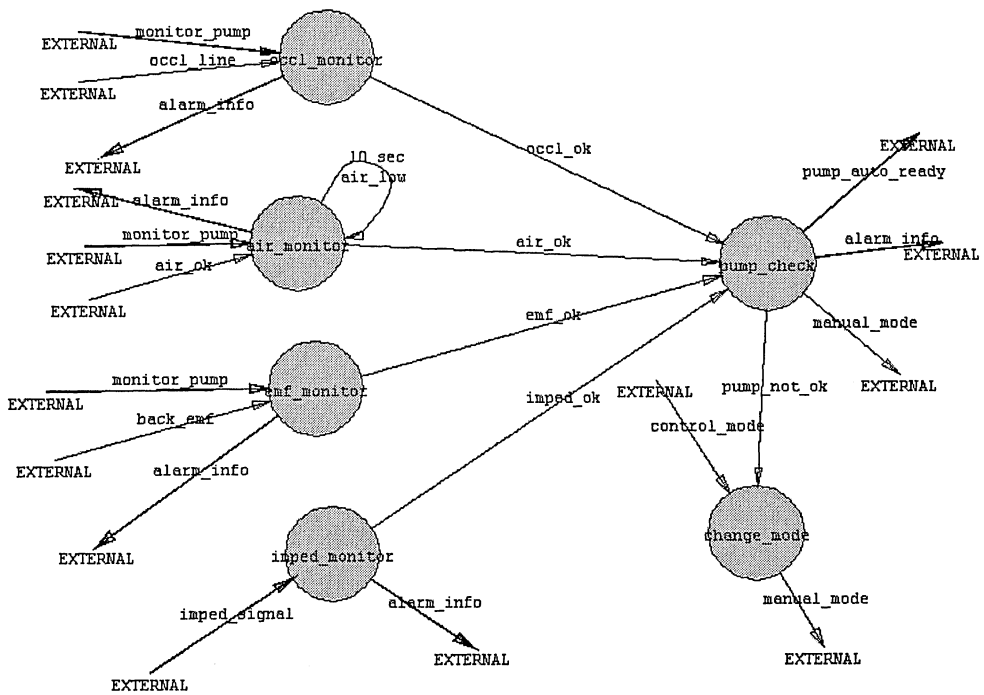

Figure 24. The pump_monitor module of Model #3

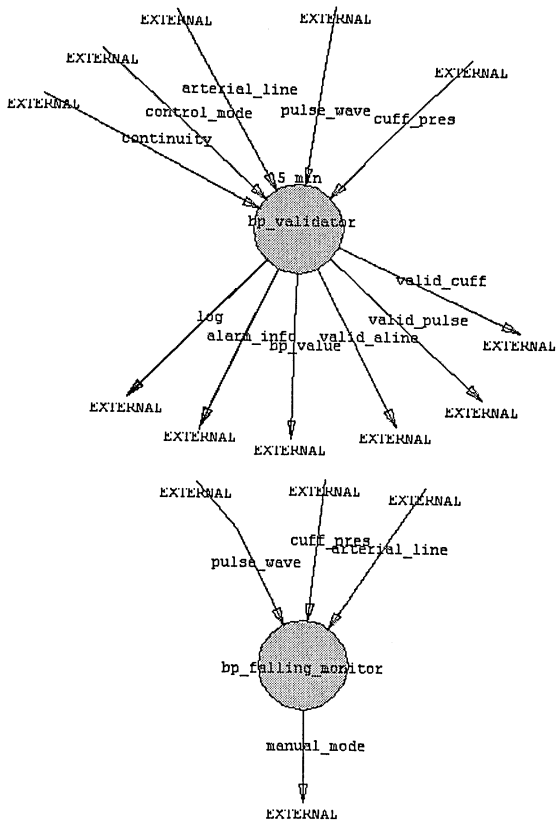

Figure 25. The bp_monitor module of Model #3

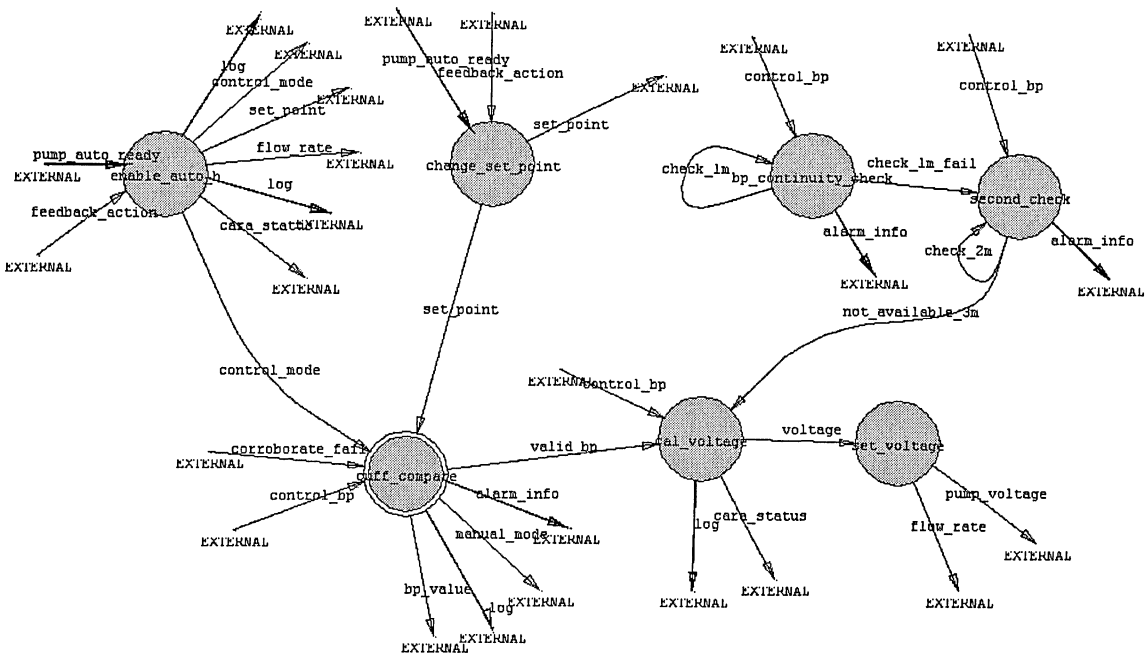

Figure 26. The pump_auto_control module of Model #3

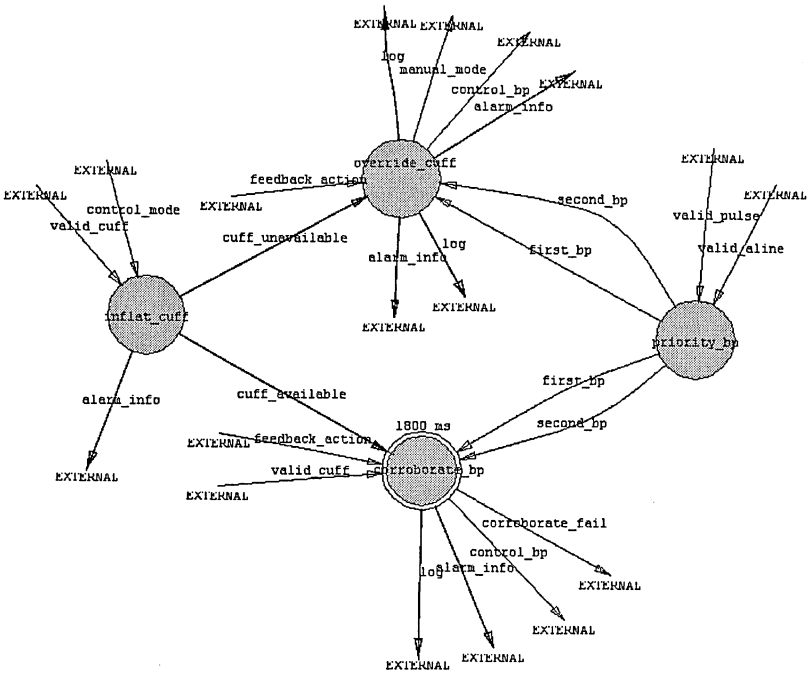

Figure 27. The bp_corroborate module of Model #3

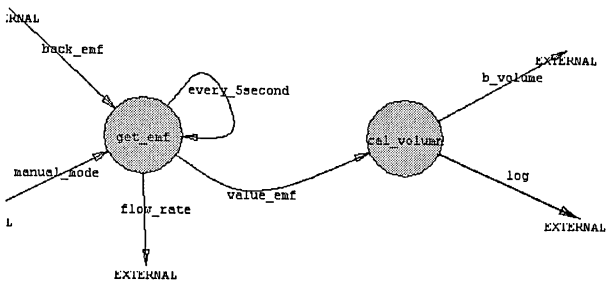

Figure 28. The pump_manual_control module of Model #3

Figure 29 shows the decomposition of the CARA_Interaction module. It consists of four sub-modules responsible for information display control, alarm signal management, resuscitation file recording and user over-ride handling.

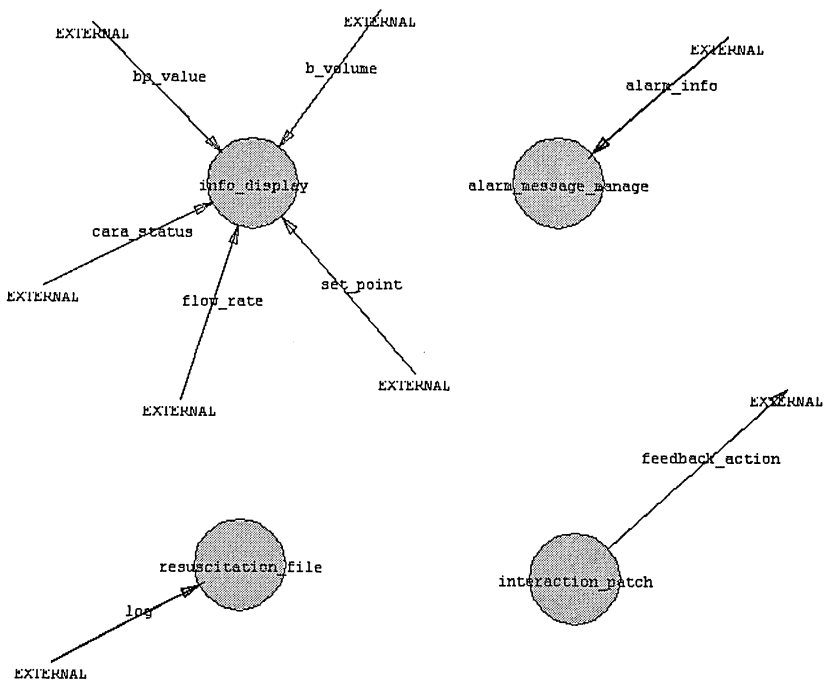

Figure 29. The CARA_Interaction module of Model #3

4.4 Design Model #4

One of the major differences between this design and the others discussed in this report is the inclusion of a test_instrumentation module in the top layer (Figure 30). The test_instrumentation module provides a GUI for the user to control the status of the pump, patient and the LSTAT simulations during prototype execution.

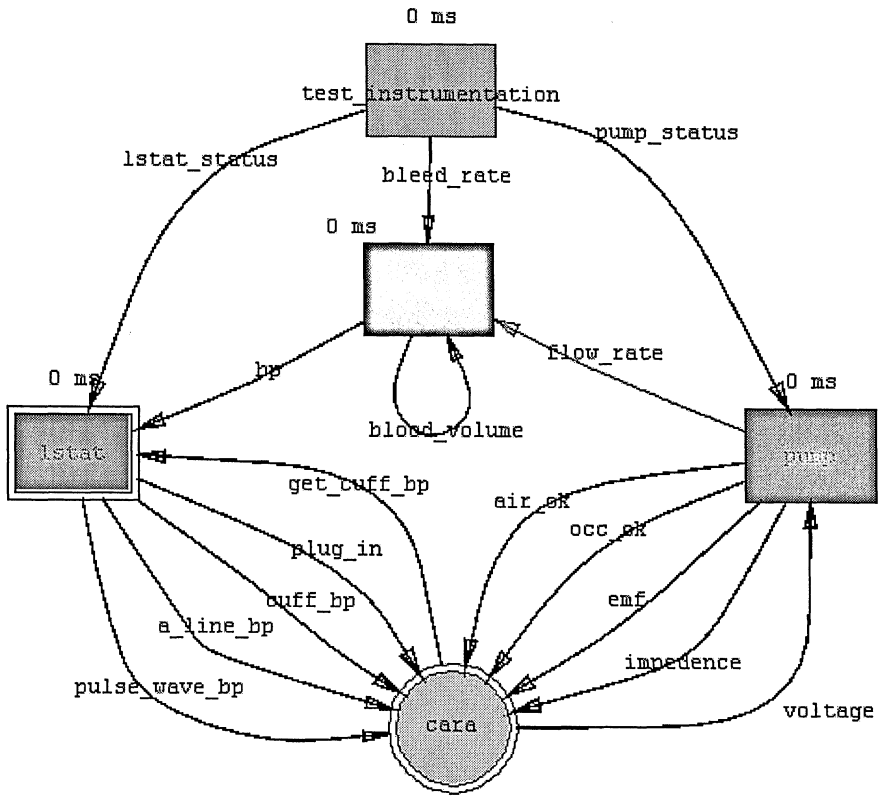

Figure 30. The top level design of Model #4

Central to this model is the CARA module, which is decomposed into six sub-modules shown in Figure 31. The monitor_bp and monitor_pump modules are responsible for monitoring and validating blood pressures from the LSTAT and monitoring signals from the infusion pump respectively. Outputs from these modules are fed to the control_pump module to determine the voltage that drives the pump rate. They also go to the manage_alarm and log_n_display_msg to alert the user as needed. Inputs from the users are processed by the manage_user_input module and the resultant events are sent to the appropriate modules for further processing.

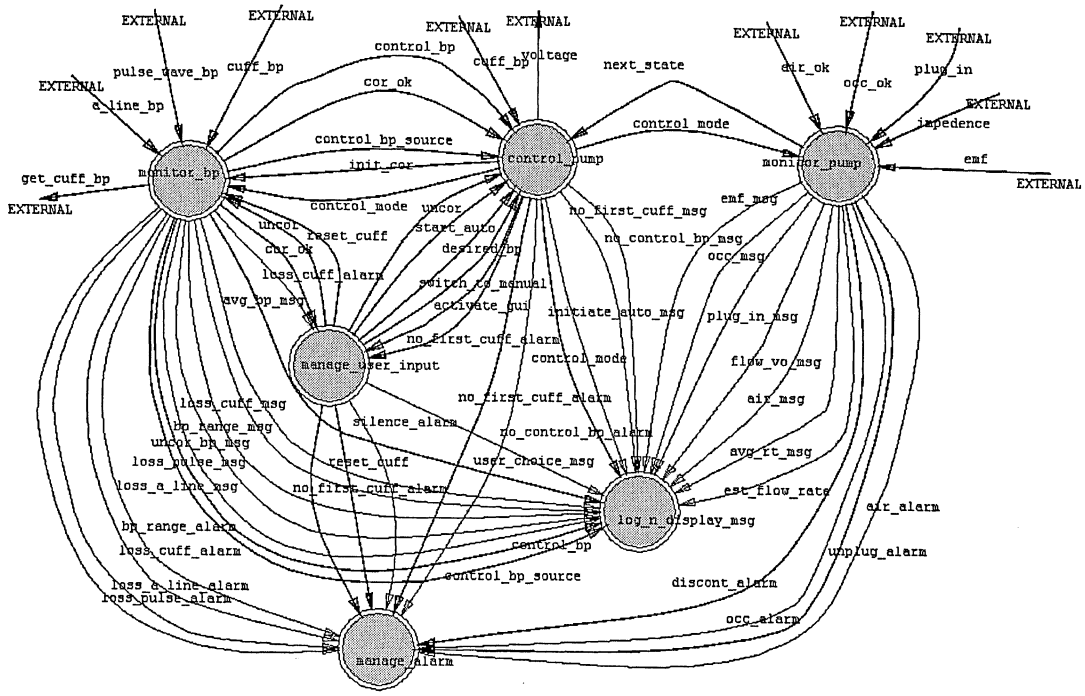

Figure 31. The CARA module of Model #4

Figure 32 shows the internal composition of the monitor_bp module. The monitor_bp module is responsible for monitoring the three different blood pressure sources (cuff, arterial line, and pulse wave). It keeps tracks of the blood pressures when the CARA control software is in manual module, and performs a blood pressure corroboration algorithm when the CARA control software is in the auto-control mode. It also alerts the user to any disruption in the blood pressure sources and signals the control_pump module to switch the CARA control software from auto-control mode back to manual mode if necessary.

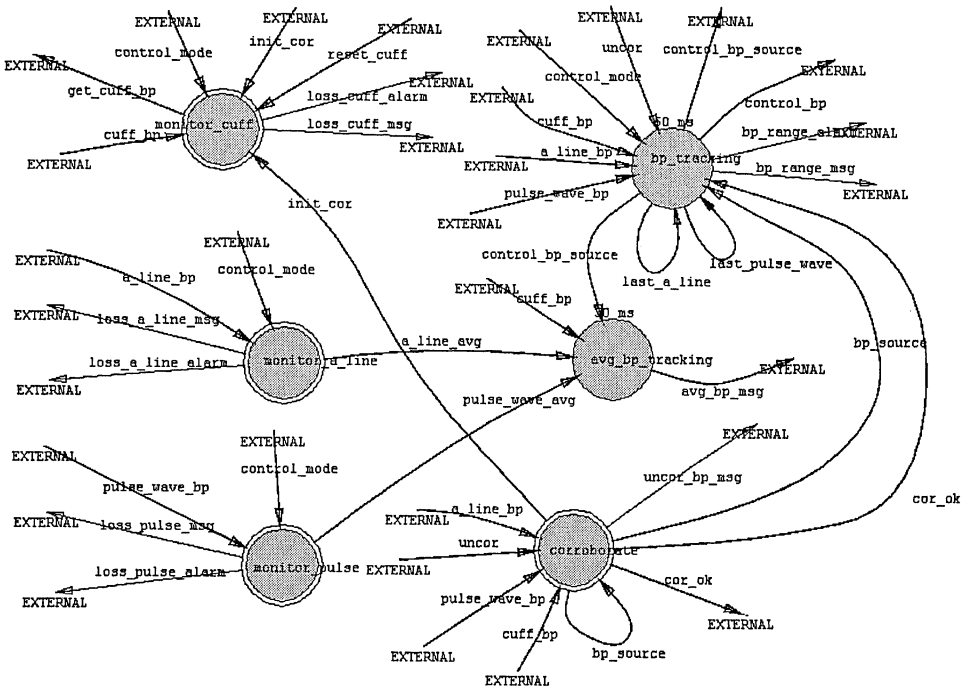

Figure 32. The bp_monitor module of Model #4

Much like the monitor_bp module, the functions of the monitor_pump module are implemented by five sub-modules: monitor_plugin, monitor_occ, monitor_air, monitor_impedence and monitor_emf (Figure 33). Outputs from these modules are used by the decide_next_state module to determine if the system is stable enough to switch into auto-control mode or if the system is so unstable that it has to switch back to or remain in manual mode. The control_pump module is then signaled to act accordingly.

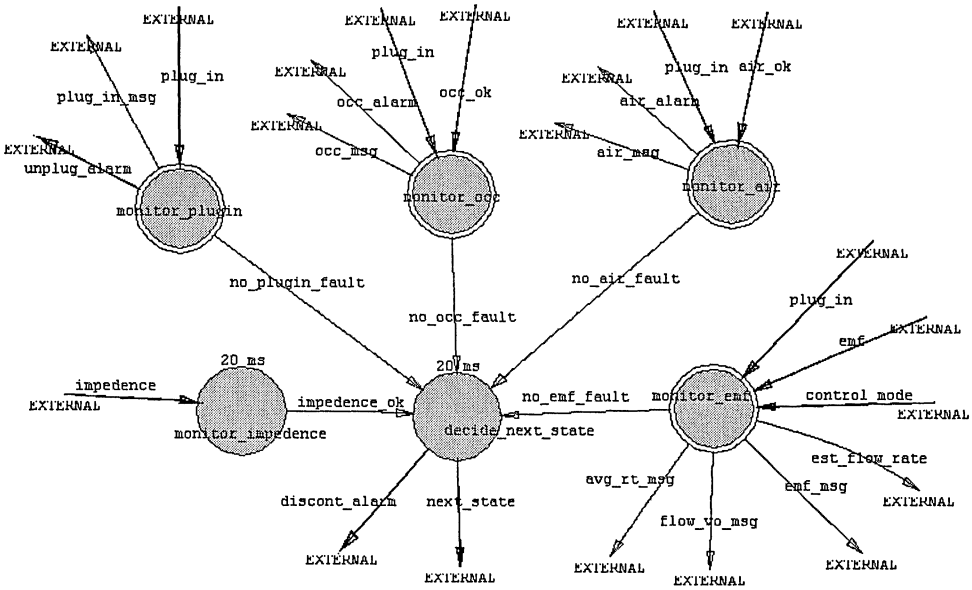

Figure 33. The monitor_pump module of Model #4

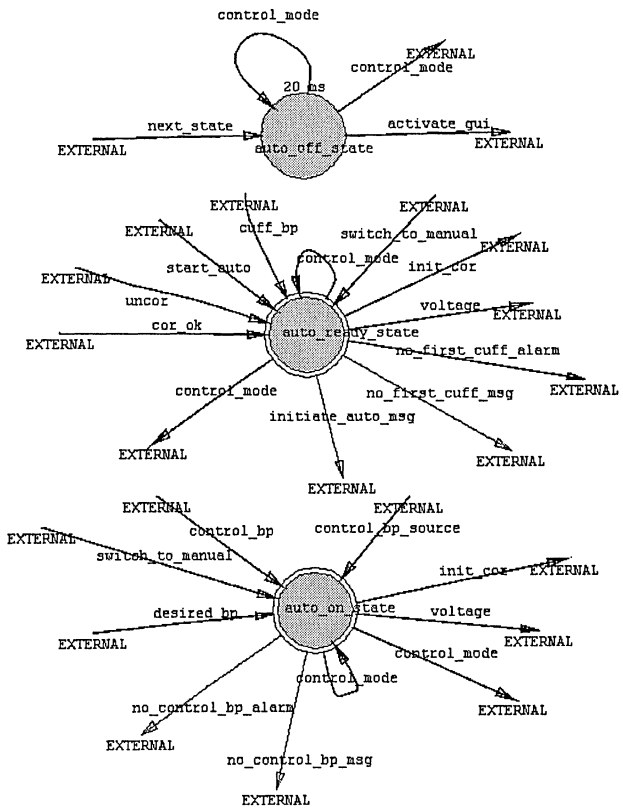

Figure 34. The pump_control module of Model #4

Figure 34 shows the decomposition of the control_pump module. It models a state machine with 3 possible states {auto_off, auto_ready, auto_on}. Depending on the value of the control_mode stream, only one of the three sub-modules can be triggered at any time.

The auto_ready_state module (Figure 35) is responsible for establishing the initial blood pressure when the user wants to start auto_control. It invokes the corresponding functions in the monitor_bp module to obtain the cuff pressure and corroborate the result with the other beat-to-beat blood pressure readings. It then uses the result to decide if the system is safe to go into the auto control mode.

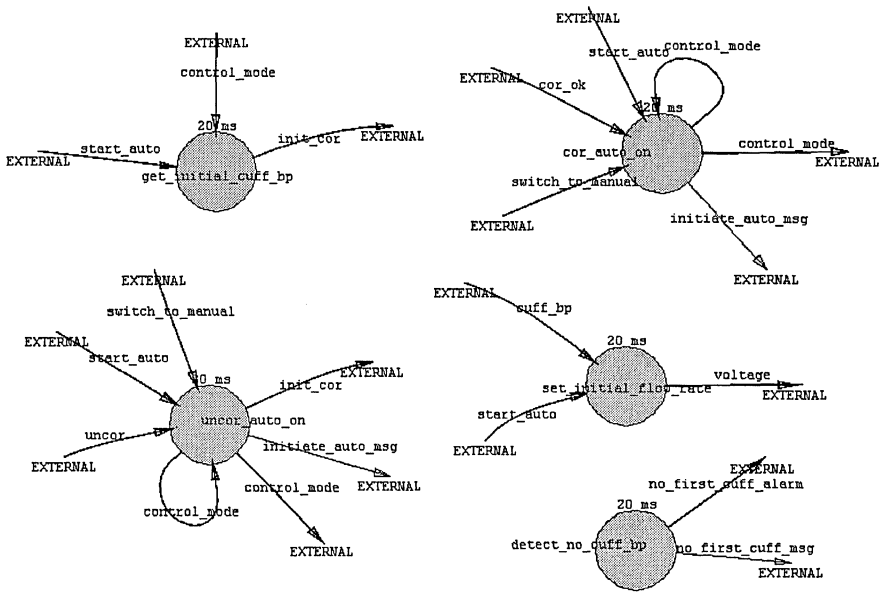

Figure 35. The auto ready state module of Model #4

Figure 36 shows the detail of the auto_on module. It establishes the cuff blood pressure according to the policy outlined in the requirements document, computes the corresponding voltage to keep the pump working at the desired flow rate, and monitors the information from the monitor_bp, monitor_pump and manage_user_input modules to determine if it is necessary to switch back to manual mode.

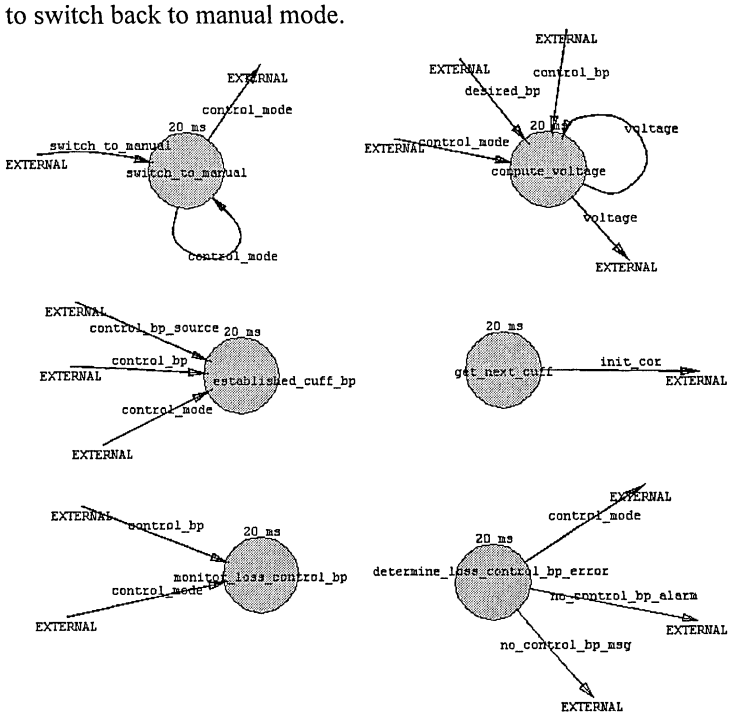

Figure 36. The auto_on_state module of Model #4

4.5 Design Model #5

This model consists of several cohesive subsystems that are created based on separation of concerns. The top layer of the model consists of the CARA software and four external sub-systems: simulated patient, LSTAT, pump and alarm (Figure 37).

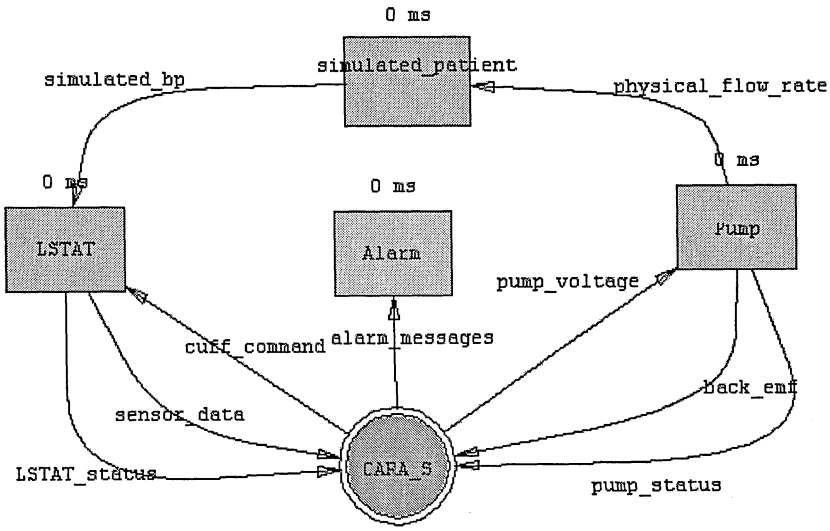

Figure 37. The top level of Model #5

Figure 38 shows the internal composition of the CARA software, which consists of six sub-modules, BP_monitor, Safety_Monitor, Pump_monitor, Pump_controller, Logger and Display. The Pump_controller module is further decomposed into a manual_pump_controller and a software_pump_controller as shown in Figure 39.

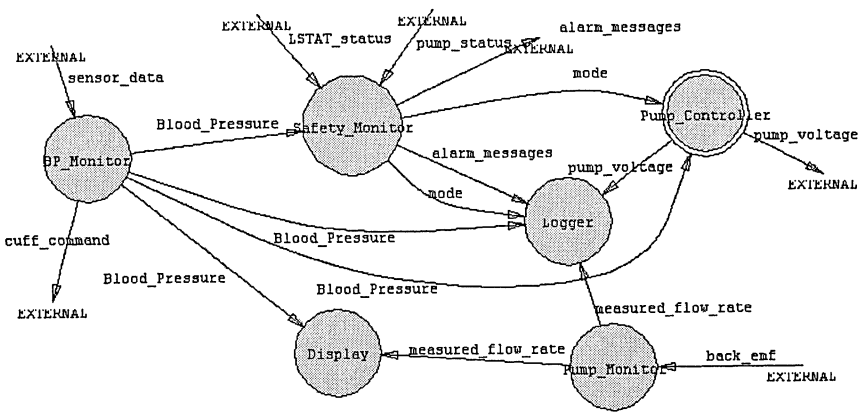

Figure 38. The CARA module of Model #5

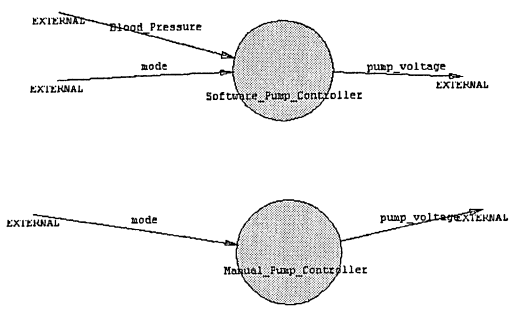

Figure 39. The Pump_controller module of Model #5

5. Comparison of the Designs

5.1 Understandability of the Model Architecture

Understandability of the model architecture is a common goal of all the designs. The design teams attempted to accomplish this goal via decomposition based on separation of concerns. Such approach resulted in very similar top-level designs. They are all made up of the CARA control software interacting with the external environment consisting of the pump and the LSTAT.

The top-level designs reflect two different assumptions of the intended prototype. Models #1 (Figure 5) and #3 (Figure 22) model the prototypes as open systems, where the pump and the LSTAT modules represent interfaces to the actual hardware. Models #2 (Figure 14), #4 (Figure 30) and #5 (Figure 37) model the prototypes as closed systems. These models all include a simulated patient to model the effect of the IV infusion on the patient's blood pressure. In addition, model #4 includes a *test_instrumentation* module to facilitate run-time testing.

Figures 7, 16, 23, 31 and 38 review the top-layer design of the CARA software. All design teams identified the following functions of the CARA software – monitoring pump, monitoring blood pressure, controlling pump, logging and displaying messages, and managing user input. However, they varied in how these functions are presented in the hierarchical decompositions. Model #1 (Figure 7) provides the simplest design. It groups these major functions into three modules – *pump_control_module* (for monitoring blood pressure and controlling pump), *io_module* (for displaying messages and managing user input), and *management_module* (for monitoring pump and logging messages). Model #2 (Figure 16) groups all monitoring functions (monitoring pump and monitoring blood pressure) into a single *monitor* module and buries the user input management deep inside the *control_alg* module. Models #3 (Figure 23), on the other hand, separates the message display and manage user input functions from the CARA control software and places them in a *CARA_interaction* module in the top level of the prototype (Figure 22). Models #4 (Figure 31) and #5 (Figure 38) have very similar designs. They both lay out all six functions explicitly in the top-layer of the CARA software.

Model #1 maintains its elegance and simplicity as we follow the decomposition to the lower levels of granularity. The whole architecture is easily followed and understood. The segregation of the identified safety critical functions from non-safety critical functions greatly enhances the safety of the design. Model #5 also attempts to segregate the safety-critical functions from the non-safety critical ones in its design, but more work is needed to help flush out the design. Models #2, #3 and #4 suffers from varying degrees of complexity at the lower levels of granularity. Model #2 (Figure 19) attempts to divide the detail activities of the control algorithms into six concurrent processes, but fails to capture the event/response relationship among these processes. Model #3 (Figures 24-28) gives a fairly complete design. Though not as elegant as model #1, its architecture is easily understood. Model #4 (Figures 32-36) gives the most detail design. It attempts to reduce the complexity of the graph through the use of triggering conditions and timer operators. For example, its *pump_control* module models a state machine consisting of three states {*auto_off_state*, *auto_ready_state* and *auto_on_state*} (Figure 34). Depending on the value of the *control_mode* state stream, exactly one of the three processes in *pump_control* module can be activated at all times. Figure 35 implements the procedure that the CARA software has to go through after the user presses the "Start Auto Control" button. The button event will trigger the *get_initial_cuff_bp* operator to request the *monitor_bp* module to corroborate the different blood pressure sources. The other four operators will remain idle awaiting the results from the *monitor_bp* module. When triggered by the arrival of the results, these four operators will decide whether it is safe to switch *into auto_control_state* and change the value of the *control_mode* state stream accordingly.

5.2 Simplicity of the Design

Simplicity of the design, in general, can be accomplished with sparse diagrams. Again, model #1 gives the simplest design. While all designs limit the number of operators in all levels to at most seven operators, they all suffer from varying degrees of data stream overcrowding. One way to solve this problem is by moving the functions around to form weaker coupling modules. For example, model #4 can greatly simplify the top-layer of the CARA software shown in Figure 31 if it follows model #1's design and place the *monitor_bp* module inside the *control_pump* module. Another way to reduce the number of data streams is through the use of composite streams. For example,

combining all alarm signals into a common stream and letting the alarm manager differentiate the different sources and priorities of the signals and process them accordingly.

5.3 Requirements coverage

All design covers most of the stated requirements in the broad sense, but they vary in detail when it comes to capturing the logic of the procedures stated in the requirements. All designs are able to identify the major functions of the CARA software and group them into different modules. With the exception of model #5, all models cover ~90% of the high level requirements and at least half of the detailed ones.

5.4 Safety Aspects of the design

The design requirements, even with the questions and answers provided, are very incomplete. The requirements' document does not provide any performance requirements – it provides primarily design requirements. In particular, the requirements documents do not identify or prioritize the functions, especially from a safety criticality perspective. As a result, the majority of the models (#2, #3 and #4) do not differentiate between safety-critical and non-critical functions.

The segregation of the identified safety critical functions from non-safety critical functions greatly enhances the safety of the design in models #1 and #5. In addition to those specified in the requirements document, model #1 includes two additional safety features in the design. First, it implements Triple Modular Redundancy (TMR) within the principle safety critical module (the *pump_control_module* in Figure 10). This TMR architecture relies on 3 different algorithms for calculating the infusion rate when in auto-control mode and a single voting mechanism to determine which rate to pass to the pump. Second, it implements a processor watchdog function (on a separate processor) to alert the operator in cases of main processor failure (Figure 8). The redundant architecture on the blood pressure corroboration promises to substantially reduce the potential for a faulty monitor to drive the infusion when in fact it should not.

6. Approach Evaluation

6.1 Evaluation of the CARA Requirements

The prototyping effort reveals a lot of omissions and discrepancies in the requirements document. First, the document does not provide any performance requirements – it provides primarily design requirements. For example:

- Req 6 explain how continuous is "continuously"? What is the maximum response time (MRT) for the system to detect and handle a discontinuity event?
- Similar to Req 6, there is no statement on how frequently the occlusion lines need to be monitored in order to satisfy Req 7.
- Neither the Back EMF units nor the algorithm in converting Back EMF to pump rate are given in Req 10.
- Do the requirements Req 11 and Req 12 concerning Back EMF computation only apply to manual control mode or to both manual and automatic control modes? Is Back EMF irrelevant in automatic mode?
- In Req 16, the term impedance is not defined. Some designers understood the term impedance as an electronic term vice a fluid mechanics term. That type of confusion could result in an implementation that could lead to hazardous conditions for the patient under treatment.
- It is unclear from the requirements document what policies are to be used for control of the blood-pressure cuff. Is it always under CARA's control, or is a hardware module in charge of taking the cuff blood pressure periodically. If the cuff is under CARA's control, then how long will it take to inflate the cuff to get a new blood pressure reading, and how long does CARA have to wait before it can inflate the cuff and take the next blood pressure reading?

- Req 20.3.2.2 is in direct contradiction of its parent requirement Req 20.3.2. The parent requirement (Req 20.3.2) states that both source pressure readings lie outside the 10% tolerance of the corresponding cuff reading, while the child requirement (Req 20.3.2.2) states that CARA should attempt to corroborate the next priority source based on the readings collected, which is a waste of time.

- In Req 20.8, the requirement does not define the maximum response time to perform the corroboration after the arrival of a higher priority blood pressure reading.

- While Req 44 requires the CARA software to check for validity of cuff pressure, the requirement document does not provide any information regarding valid range of blood pressure from different sources.

- There is no requirement related to what happens when fluid gets low. Shouldn't some kind of alarm go off? How does the pump sense low fluid level?

- There is no requirement for any kind of redundancy (with exception of redundancy implied by BP measurements). We went ahead and implemented a TMR (Triple Modular Redundancy) architecture on the key CARA algorithm module in Model #1.

- A key aspect from the requirements is that the feedback available from the system allows for many capabilities in the CARA that were not included in the requirements. Specifically, the various forms of feedback would allow the CARA to perform diagnostics of various components attached to the system on a real-time basis. While such capabilities may not be part of the intent of the CARA, they would substantially improve its safety and functionality. An example is the Back EMF from the infusion pump. The requirements simply state that the CARA should monitor the Back EMF to determine the flow rate. Coupled with the measurements of the impedance (resistance to flow of the infused solution), the CARA can calculate the amount of influent provided a patient as a function of flow rate and time. However, the Back EMF provides additional information. A low value indicates that there is little resistance to pumping the fluid, it may indicate that the IV line is not inserted (i.e., pumping free). Conversely a high Back EMF with a constant impedance may indicate occlusion before the Occlusion signal occurs or it may indicate a failing pump.

6.2 Evaluation of LAMPS and SEATools

The experiments showed that LAMPS can effectively be used to model complex embedded software. LAMPS's triggering guards and execution guards provide a very convenient means for users to specific state machines explicitly without resort to target code. The timer feature is very useful in modeling complicated timing policies. For example, Figure 40 shows a simple model with two operators and 4 timers for the policy that

"When the cuff pressure is being used for control:
If the mean BP is 60 or below, cuff pressures will be taken once per minute;
If the mean BP is (60 - 70], cuff pressures will be taken once every 2 minutes;
If the mean BP is (70 - 90], cuff pressures will be taken once every 5 minutes;
If the mean BP is above 90, cuff pressures will be taken once every 10 minutes".

The sporadic operator established_cuff_bp is triggered each time it receives a new cuff blood pressure reading and starts the appropriate timer for the next cuff-reading event. The get_next_cuff, on the other hand, polls the timers once every 30 seconds, and issues the init_cor command to the monitor_bp module in Model #4 if any of the active timer reaches its preset time triggered.

Since the above design implements the policy via LAMPS directly, users can accommodate any policy changes easily without the need to modify any source code. Moreover, since a cuff reading event may also be generated by other conditions like the loss of a beat-to-beat blood pressure source detected by the monitor_bp module in Model #4, the above design avoids any conflict with such event since it will reset its timers automatically whenever it receives a new cuff blood pressure reading.

TRIGERRED BY ALL control_bp
IF control_bp_source = cuff
and control_mode = auto_on_mode
MAXIMUM RESPONSE TIME 1 sec
MINIMUM CALLING PERIOD 50 sec
MAXIMUM EXECUTION TIME 50 ms

RESET TIMER one_min_timer
RESET TIMER three_min_timer
RESET TIMER five_min_timer
RESET TIMRE ten_min_timer
START TIMER one_min_timer IF last_cuff_bp <= 60
START TIMER three_min_timer
IF last_cuff_bp > 60 and last_cuff_bp <= 70
START TIMER five_min_timer
IF last_cuff_bp > 70 and last_cuff_bp <= 90
START TIMER three_min_timer IF last_cuff_bp > 90

PERIOD = 30 sec
FINISH WITHIN 1 sec
MAXIMUM EXECUTION TIME = 50 ms
TRIGGERED IF
one_min_timer >= 1 min or
three_min_timer >= 3 min or
five_min_timer >= 5 min or
ten_min_timer >= 10 min

RESET TIMER one_min_timer
RESET TIMER three_min_timer
RESET TIMER five_min_timer
RESET TIMER ten_min_timer

Figure 40. Model for a Cuff Blood Pressure Monitor Policy

SEATools provides the essential facilities for users to create and modify the models. It is very easy to reason about complexity using SEATools. When there are many data streams from one operator to another, it is easy to see and fix (most of the time by using a user-defined type for the stream). It is also easy to see when a particular level is too complex and needs to be further decomposed. By trying to fully implement each requirement in the model it was clear which requirements were fully/consistently specified and which were not.

The tool provides an effective means to perform requirement consistency and understandability checking. It also provides some degree of computer-aided inconsistency checking and data entry propagation at the user interface level, and complete semantic check via the translator. Figure 41 shows a simple graphical user interface of an executable prototype for Model #4, which has 20 composite operators and 89 atomic operators. The executable prototype consists of 14.7K lines of source code, 8.5K of which are generated by the translator and the scheduler of the SEATools.

The experiments also exposed some errors and future enhancements for SEATools. Future enhancements include abstraction for data streams, visual queues for the declaration and use of timers, multiple views for requirements traces, better facilities for constructing user define types, ability to designate which operators will be implemented on separate processors.

In FY03 we plan to complete the prototype to the point where measurements can be made, and to explore ways in which such measurements can be related to the degree of confidence users can put on systems. We also plan to fix deficiencies in the models and tools that were exposed by this exercise. For example, it was found that we needed to be able to decompose data streams into finer generic data streams to keep complex architecture understandable. It was also found that safety concerns required expressing new kinds of constraints. For instance, particular logical processes should be hosted on independent pieces of hardware to remove common causes for coincident failures.

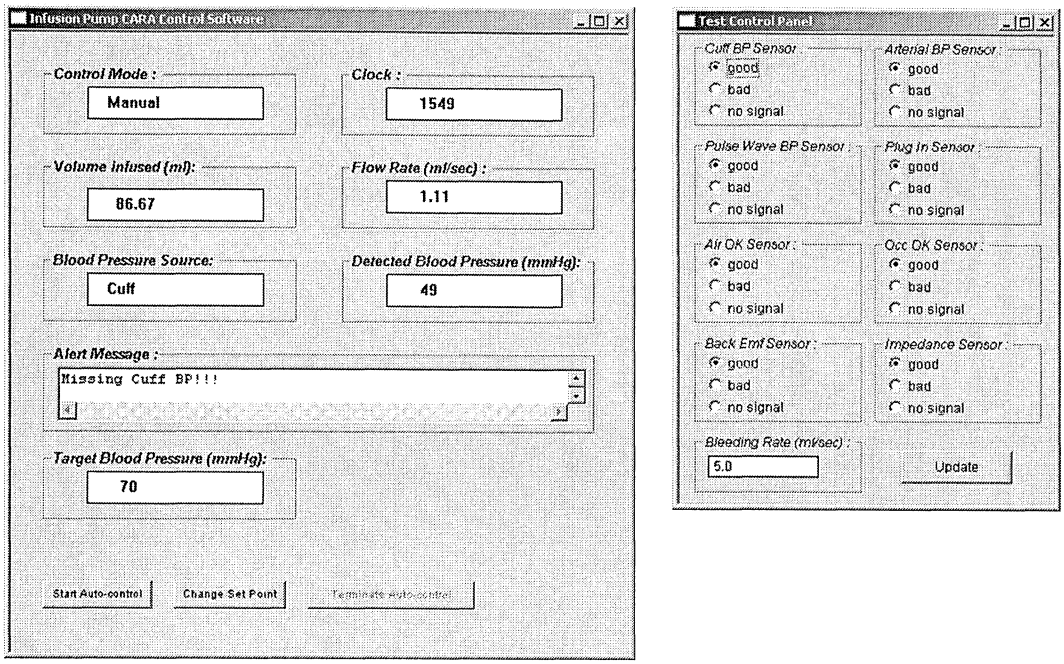

Figure 41. The Graphical User Interface of the Executable Prototype for Model #4

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5100	2
3.	Research Office, Code 09 Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Dr. David Hislop U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211	1
5.	Dr. Valdis Berzins, CS/Be Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
6.	Dr. Man-Tak Shing, CS/Sh Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
7.	Dr. Luqi, CS/Lq Computer Science Department Naval Postgraduate School Monterey, CA 93943	7