February 2004

Formal Specifications and Analysis of the Computer Assisted Resuscitation Algorithm (CARA) Infusion Pump Control System

Rajeev Alur
University of Pennsylvania, alur@cis.upenn.edu

David Arney
University of Pennsylvania

Elsa L. Gunter
New Jersey Institute of Technology

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Jaime Lee
Walter Reed Army Institute of Research

See next page for additional authors

Formal Specifications and Analysis of the Computer Assisted Resuscitation Algorithm (CARA) Infusion Pump Control System

Abstract
Reliability of medical devices such as the CARA Infusion Pump Control System is of extreme importance given that these devices are being used on patients in critical condition. The Infusion Pump Control System includes embedded processors and accompanying embedded software for monitoring as well as controlling sensors and actuators that allow the embedded systems to interact with their environments. This nature of the Infusion Pump Control System adds to the complexity of assuring the reliability of the total system. The traditional methods of developing embedded systems are inadequate for such safety-critical devices. In this paper, we study the application of formal methods to the requirements capture and analysis for the Infusion Pump Control System. Our approach consists of two phases. The first phase is to convert the informal design requirements into a set of reference specifications using a formal system, in this case EFSMs (Extended Finite State Machines). The second phase is to translate the reference specifications to the tools supporting formal analysis, such as SCR and Hermes. This allows us to conclude properties of the reference specifications. Our research goal is to develop a framework and methodology for the integrated use of formal methods in the development of embedded medical systems that require high assurance and confidence.

Keywords
CARA system, requirements formalization, safety-critical systems, formal methods, software verification

Comments
Postprint version.

Author(s)
Rajeev Alur, David Arney, Elsa L. Gunter, Insup Lee, Jaime Lee, Wonhong Nam, Frederick Pearce, Steve Van Albert, and Jiaxiang Zhou

Formal Specifications and Analysis of the Computer Assisted Resuscitation Algorithm (CARA) Infusion Pump Control System *

Rajeev Alur † David Arney † Elsa L. Gunter ‡
Insup Lee † Jaime Lee § Wonhong Nam †
Frederick Pearce § Steve Van Albert § Jiaxiang Zhou †

October 2003

Abstract

Reliability of medical devices such as the CARA Infusion Pump Control System is of extreme importance given that these devices are being used on patients in critical condition. The Infusion Pump Control System includes embedded processors and accompanying embedded software for monitoring as well as control sensors and actuators that allow the embedded systems to interact with their environments. This nature of the Infusion Pump Control System adds to the complexity of assuring the reliability of the total system. The traditional methods of developing embedded systems are inadequate for such safety-critical devices. In this paper, we study the application of formal methods to the requirements capture and analysis for the Infusion Pump Control System. Our approach consists of two phases. The first phase is to convert the informal design requirements into a set of reference specifications using a formal system, in this case EFSMs (Extended Finite State Machines). The second phase is to translate the reference specifications to the tools supporting formal analysis, such as SCR, CHARON, and Hermes. This allows us to conclude properties of the reference specifications. Our research goal is to develop a framework and methodology for the integrated use of formal methods in the development of embedded medical systems that require high assurance and confidence.

*†University of Pennsylvania, ‡New Jersey Institute of Technology, §Walter Reed Army Institute of Research

*This research was supported in part by ARO DAAD19-01-1-0473, NSF CCR-9988409, NSF CCR-0086147, NSF CISE-9703220, and SRC 99-TJ-688. POC: Insup Lee (lee@cis.upenn.edu)

1 Introduction

Medical devices with embedded processors are part of safety-critical systems which must be highly reliable and built correctly with respect to the requirements. Such embedded systems consist of a collection of components that interact with each other and with their environment through sensors and actuators. Embedded software is used to control these sensors and actuators and to provide application-dependent functionality. Demands on embedded systems have been increasing as microprocessors become more powerful. This in turn requires more complicated supporting software which must also be verified. Embedded systems have been developed traditionally in an ad-hoc manner by practicing engineers and programmers. The existing technology for embedded systems does not effectively support the development of reliable and robust embedded systems. The effective development of embedded systems requires a collection of tools to capture requirements, to construct, analyze and simulate specifications, to generate and test implementations, and to monitor and check implementations at run-time.

The research goal of the UPenn/NJIT team is to develop a framework and methodology for the integrated use of formal methods in the development of embedded medical systems that require high assurance and confidence. Our approach is to apply the formal techniques to different stages of system development, ranging from requirements capture to design specification and analysis to code generation and validation. This paper describes our case study based on an infusion pump system that has been developed at WRAIR (Walter Reed Army Institute of Research). Figure 1 shows our approach, which consists of two phases. The first phase is to manually convert the informal design requirements into the reference specifications in EFSM (Extended Finite State Machines). The informal design requirements were supplied to us by WRAIR in the form of a tagged requirements document written in informal English, together with a listing of 133 questions and answers. The questions-and-answers document amended and overrode the original tagged requirements document. This first phase resulted in many new questions and answers as well as refinements to the requirements to remove inconsistencies. The second phase is to manually translate the reference specifications to the tools based on formal methods, such as SCR [5], CHARON [2], and Hermes [3]. These tools have different strengths and weaknesses, and they are used together to complement each other. For example, SCR can provide us with coverage checks, CHARON supports the modeling of hybrid systems involving both discrete and continuous behaviors (as is needed to include a model of the patient in the system), and Hermes supports model checking of safety properties.

The rest of the paper is organized as follows: Section 2 gives an overview CARA and its components, Section 3 shows the details of our EFSM translation of the English requirements, Section 4 discusses our SCR model, Section 5 explains the Hermes model, and finally, Section 6 contains our conclusions.

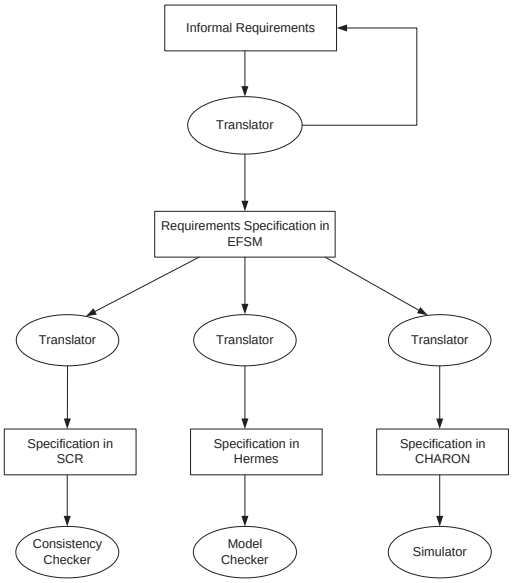

Figure 1: Two Phase Approach

2 Overview of CARA

The Computer Assisted Resuscitation Algorithm (CARA) is a system which determines the rate at which an infusion pump should infuse fluid such as saline into a patient suffering from severe hypotension due to blood loss. The CARA determines the rate based on the patient's current blood pressure; the more severe the hypotension the more rapidly the fluid is to be infused. CARA controls a system which includes a M100 infusion pump from Infusion Dynamics and a physiological monitoring device. The physiological monitoring device is the conveyor of blood pressure information for the CARA, including data from potentially several different sources such as a non-invasive cuff, a pulse wave velocity sensor, or an arterial line pressure sensor. The eventual use for the CARA system will be combat casualties, and it seems probable that a device such as this will eventually become part of emergency medical equipment deployed for automobile and industrial accidents.

The CARA software system is the aim of our formal reference specification analysis. The CARA software functions within a context that is described in Figure 2. The total Infusion Pump System consists of five components: CARA, Pump, Blood Pressure Sensor, Caregiver, and Patient. CARA must interact with each of these components and clarifying that interface is an important aspect of the reference specification. For the most part, in this work, we have focused on formalizing and analyzing just the reference specification of CARA. However, by modeling (in Hermes and CHARON, for example) the Caregiver and Patient as part of the total infusion pump we are able to capture require-

ments about their behavior upon which the correctness of CARA depends. This section briefly describes the functionality of individual components and their interfaces.

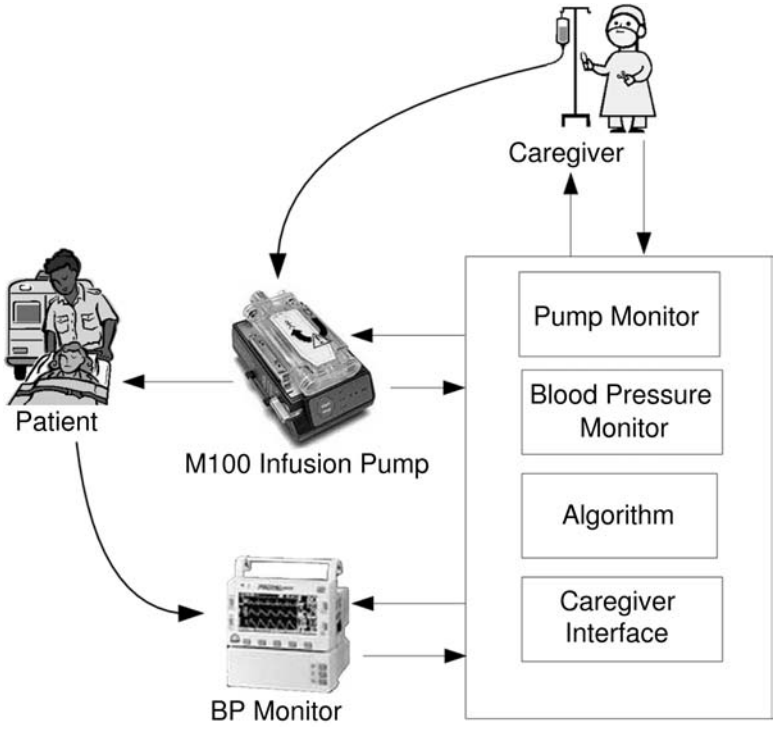

Figure 2: Overall Structure of the Infusion Pump System

CARA. CARA will drive the infusion pump with the aim of raising the patient's blood pressure and maintaining it at an acceptable level. The pump's infusion speed will be calculated by CARA based on the patient's current blood pressure and the desired average blood pressure set by the medical staff. A physiological device attached to the patient will provide the algorithm with the patient's current blood pressure information. The desired average blood pressure will be set by an attending caregiver or left at the default of 70 mmHg. The reason for such a minimal blood pressure set-point is that CARA is expected to be used, at least initially, in situations where the availability of resuscitation fluid is quite limited and there may be several patients to be treated. While 70 mmHg is a low blood pressure, it is sufficient to ensure the survival of the patients until they can be transported to better equipped care facilities. The caregiver has the option of increasing the set-point if it is judged necessary.

CARA communicates with the caregiver via alarm signals, messages that appear on a display screen, and user input via soft buttons. The display screen

shows the the driving voltage (and hence the flow rate) as determined by the algorithm. Whenever a pump fault occurs (e.g., occlusion in the fluid tubing), appropriate alarm signals are activated. The caregiver is responsible for removing the pump's faults, and if they happen when the pump is being controlled by CARA, the software releases its control. We have tried to abstract the interface so that it can specify what a replacement pump would have to satisfy in order to be used instead of the M100.

Blood Pressure Sensor. The Blood Pressure Sensor is the monitoring device which is the source of physiological signals such as arterial blood pressure and cuff blood pressure. It will be attached to the patient and communicate the signals to CARA. However, the design does not allow for direct interface with the algorithm. Instead, all data from the sensor is assumed to be stored in a shared buffer that the software will have access to. This allows for a more modular design which could enable greater flexibility in using a variety of monitoring devices.

Pump. The infusion pump is the device which moves fluid into the patient. It has a variety of sensors and can trigger alarms if fault conditions occur. The pump has two modes, manual and autocontrol. In manual mode, the pump speed is set with a switch built into the pump. In autocontrol mode, the pumping speed is set by a control voltage from an external source.

Caregiver. The caregiver represents the person that will be in charge of the infusion, usually a doctor, nurse, or medic. He or she can interact with the actual pumping device via hardware buttons on the pump and with the CARA software via messages and soft buttons on a display screen. He or she sets the desired blood pressure via soft buttons on the display, and sets a default flow rate directly on the pump for use by the pump when it is operating in manual mode (not under the control of CARA). The caregiver also takes care of potential faults and failures that may occur during infusion.

Patient. The patient is the object in this system. He or she is connected to the pump and the blood pressure sensor. Infusion is carried out depending on the patient's current state, with the aim to rapidly restore his or her intravenous volume and blood pressure.

3 Reference Model Specification of CARA in EFSM

As mentioned in the introduction, the first phase was to convert the informal requirements to the reference formal specifications.

Researchers at Walter Reed Army Institute for Research (WRAIR) provided us with two documents upon which our specification is based [12] [11]. These were the requirements list and a list of questions and answers. The requirements

document listed 228 requirements which the software had to meet. For example, typical requirements are [12]:

> **Requirement 8.1:** If the Air OK signal remains low for 10 seconds
> **Requirement 8.1.1:** An appropriate alarm message should be issued.

The questions document listed 133 questions and answers, which amended the requirements document. This format led to some initial confusion as the questions document sometimes overruled the requirements. In order to write our specification, it was necessary to use both documents in concert. This problem was exacerbated when we started adding our own questions to the list.

The requirements documents and appended questions developed at WRAIR provide a fairly thorough description of the desired behavior of the CARA system. However, they do not quite fit the standard, for example the Software Engineering Standard of the European Space Agency [9], for a user requirements document in a variety of ways. One of the problems we have in attempting to derive a system requirements document from the requirements documents from WRAIR is that the requirements documents from WRAIR contain statements about the operations CARA should perform internally to achieve certain external behaviors. An example of the specification of such internal behavior can be seen in Requirement 9:

> **Requirement 9:** CARA should monitor the back EMF line from the pump to keep track of infused fluids by polling immediately when the pump is plugged and then on every even 5-second clock interval while the pump remains plugged in.

The "even 5-second clock interval" is referring to the value of an internal clock. In our work, we considered the requirements documents as a given, except where we required additional clarification. Therefore, our specifications also fall short of the standards for system requirements documents because they deliberately contain a reflection of all internal details described in the original requirements documents.

We started our translation by specifying the CARA system using 21 Extended Finite State Machines (EFSMs). An EFSM is a finite state machine with variables, and in our model the state machines communicate using shared variables. Specifying the system in this way allowed us to generate a standard document which could be referred to by each of the several teams working with the system. Our goal in creating EFSMs was to capture our understanding of the requirements documents in a precise and mathematically rigorous manner. We could then apply various formal analysis techniques by translating EFSMs to other formalisms. The requirements documents and questions were written in relatively non-technical English, and thus contained a number of ambiguities. In the course of creating the EFSMs, we generated 30 questions identifying 4 inconsistencies, 12 instances of incompleteness and clarifying our understanding of 14 terms.

Inconsistencies arose when the requirements defined behaviors which were contradictory or incompatible. These definitions appeared both in the tagged requirements document and the questions. For instance, there were several exchanges requesting clarification on the fact that the requirements indicate that certain actions are to be taken if a beat-to-beat source is lost for more than 3 minutes (Requirements 42 and 43), but the Q/A document says it should be 2 minutes (Question 120).

Incompleteness is when the requirements did not specify a behavior which seemed to be necessary or which was required by another behavior. Question 134 asks whether we can assume that we can detect the source of a BP reading, as it seemed to be assumed but was never stated explicitly.

Clarifications of terms were necessary as the WRAIR team used medical terminology we were unfamiliar with. It was also necessary in cases where the English was ambiguous. For instance, one requirement stated that something should be done on every even 5 second interval. We modeled the system so that this event happened at the 10, 20, 30, ... second marks – the even (not odd) 5 second intervals. It was only after we asked for clarification that we knew that they meant that it should happen every 5 seconds.

3.1 Overall Structure of CARA.

In addition to directly implying certain internal implementation details, the requirements documents strongly suggest certain internal logical structure, or components for any implementation. We now describe a logical decomposition of CARA into these modules and briefly explain their functionalities and the interfaces between them. A graphical layout of the CARA components and their interfaces is depicted in Figure 3. The components include Caregiver Interface, Blood Pressure Monitor, Algorithm, and Pump Monitors.

The purpose of the Algorithm component is to control the infusion rate of the pump and keep track of infusion related data in log files. A patient's blood pressure is used to compute the rate at which the pump will be infusing: the higher the blood pressure, the lower the flow rate. The CARA algorithm controls the flow rate as long as there are no complications in the pump's operation. In case of complications, control reverts to the manual operator or caregiver.

The Caregiver Interface unit has two functions. First, it serves as means of communication with a caregiver. It allows the caregiver to modify infusion parameters such as the target blood pressure, and also initiate and terminate the algorithm's control of the pump. Second, it displays and sounds error messages. There are two kinds of error messages: pump malfunctions and patient/infusion related error messages.

The Blood Pressure (BP) Monitor relays vital data (such as BP source, BP value and infusion related errors) from the BP sensors to the Algorithm and Caregiver Interface units. The BP Monitor is designed to work with multiple sources, such as arterial line, pulse wave velocity, or cuff sensors, and it will automatically select the 'best' source after corroborating them. It also calculates the change in the patient's blood pressure and can trigger a number of alarms.

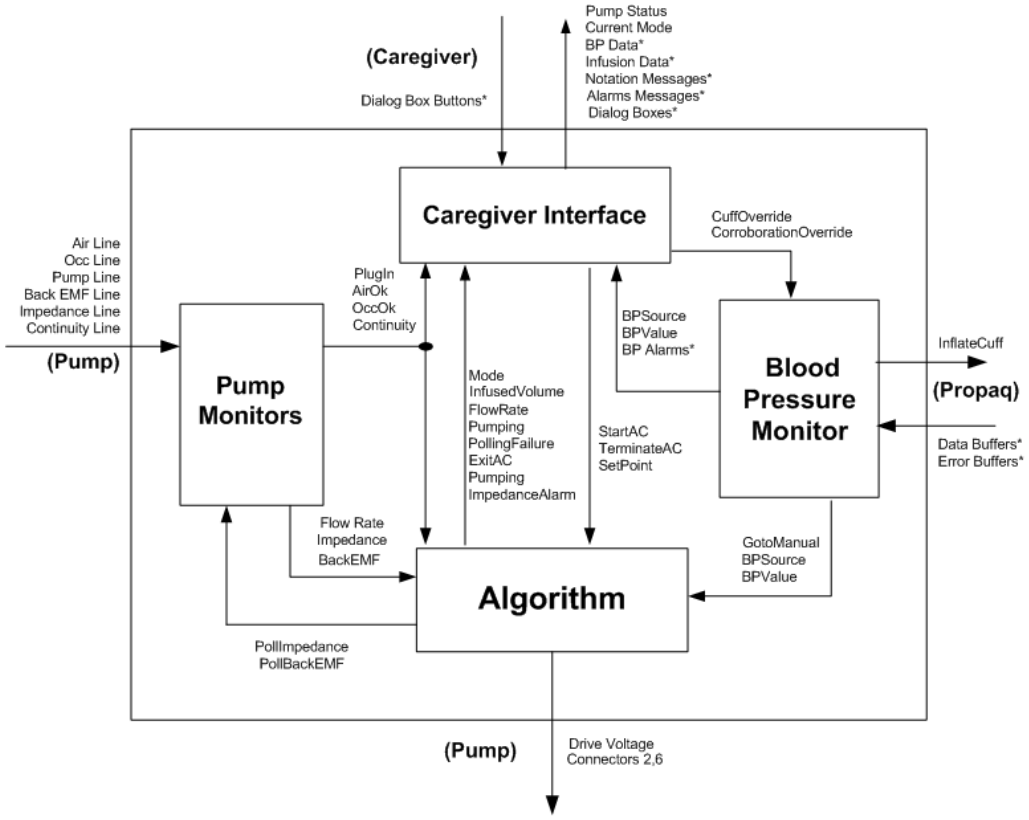

Figure 3: Components of CARA module

The Pump Monitors check that the pump is functioning correctly by checking that it is plugged in and that the infusion process is within norms. There are individual monitors for each of the key readings followed by the monitors. These include occlusion, impedance, back EMF (i.e., a measure of how fast the pump motor is pumping), continuity, and whether the pump is plugged in. The monitors report data back to the Algorithm and may also directly trigger alarms.

3.2 Modeling of CARA in EFSM.

The hierarchical structure within the four components of CARA is shown in Figure 4.

- The Pump Monitor is modeled as six EFSM's, where Monitor Plugin, Monitor AirOk, Monitor OccOk, and Monitor Continuity are used by the monitor to check if the input signal is available, and Monitor backEMF and Monitor Impedance are used to poll the flow rate and infused volume and to poll the impedance during Pumping.

- The BP Monitor is modeled as three EFSM's, which in turn include nested EFSM's: BP Detector is responsible for determining the BP sources and BP value, BP Handler is to handle the data and corroborate the BP Source, and BP Checker is used in Auto Control status to report falling BP and BP slope. In the Detector and Handler, different BP sources have different EFSM's.

- The Caregiver Interface has three subcomponents. Status Display shows pumping data and the current mode of CARA and also reports any exceptional situation. Button I/O allows the caregiver to reset the set point value. Alarm Display show messages associated with current alarms.

- The Algorithm is used to decide the current mode of CARA and to implement the polling algorithm. It switches from one mode to the other according to the current situation of the pump system. It also controls the pumping flow rate according to the control BP value.

Interfaces between the modules are modeled by global variables in the EFSMs. We specify which components read and write a particular variable and what the meaning of that variable is. To prevent reuse of names, we prefix each variable name with a two letter code for the module. All variables are globally readable but only writeable by a single module. Thus, if there is a condition which can be triggered by multiple modules, it is necessary to calculate the conjunction of the conditions for each module. For example, in the Algorithm EFSM, we see the statement "CA_backManual OR CB_backManual OR CP_backManual OR CC_backManual $\rightarrow$ CA_mode = Manual". This simply means that any of four EFSMs can trigger the algorithm to switch the mode to "Manual". We use similar message passing schemes for most of the alarms.

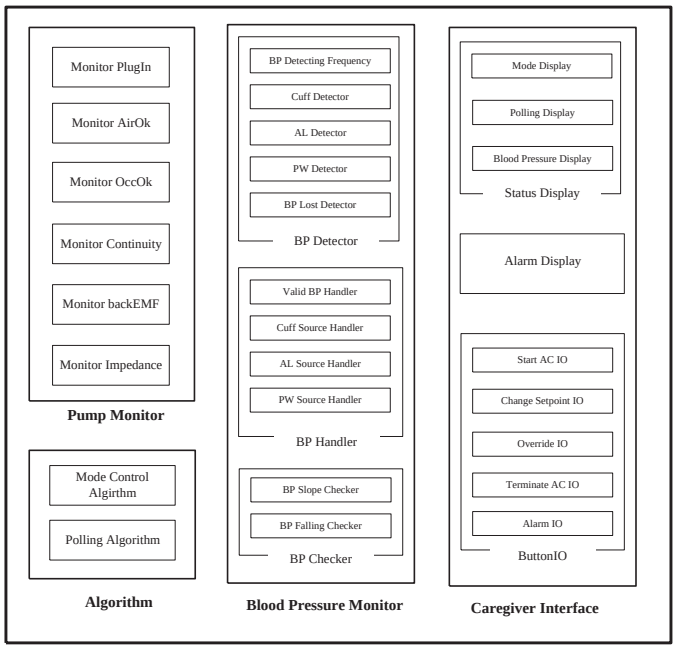

Figure 4: Detailed view of CARA components

3.3 Detailed Example

As an example of the lowest level of detail captured in the EFSM specification, we will describe the EFSM components for the Blood Pressure Handler in greater detail. The English requirements are shown in Figure 6.

Figure 5 shows the EFSM we generated from the requirements listed in Figure 6. This generation was done manually through several iterations. Table 7 shows the listing of variables used in this state machine. Variables can be written by only one EFSM in the system, but may be read by many EFSMs.

Generating the EFSM documents from the requirements list was a multistep, iterative process. The first step was reading and understanding the English language requirements. In addition to the requirements illustrated in Figure 6, there were multiple sets of questions which modified or superceded the original requirements. This arrangement necessitated multiple reading of all the documents to identify the requirements pertaining to a particular section. Once we identified the requirements we thought went together, we began constructing a state machine to implement those requirements. After several iterations of this process, we finalized the set of EFSMs presented in part in this document.

The requirements listed in Figure 6 are not implemented entirely in the Cuff Handler EFSM. There is another EFSM (not shown) which assigns the control BP and handles validation of BP sources, and there are a number of other EFSMs involved. The correspondence from requirements to the EFSM is fairly

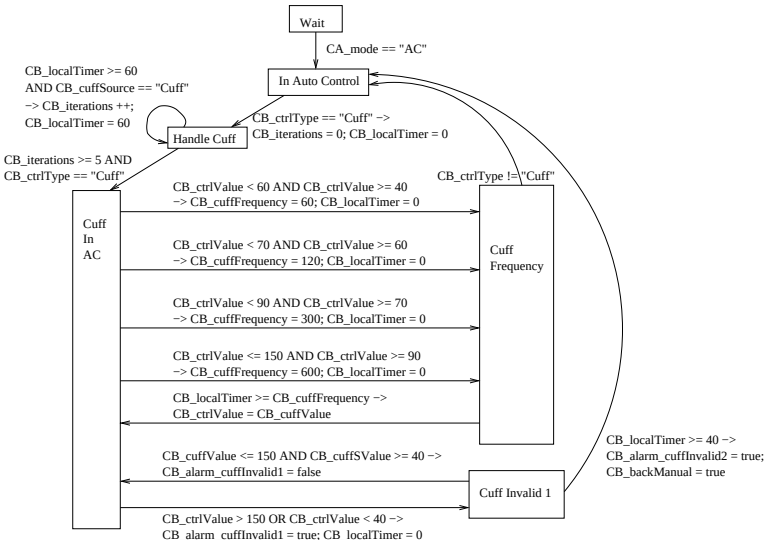

Figure 5: Cuff Handler

straightforward. Requirements 27.1 through 27.4 pertain to setting the BP sampling rate, and they match with the four transitions from "Cuff in AC" to "Cuff Frequency". Requirement 44.3.1.3 corresponds to the transition from "Cuff Invalid 1" to "In Auto Control", specifically the statement "CB_backManual = true" which triggers the Algorithm component to return to Manual mode.

As we used the EFSMs to create models in the various tools, we found a number of errors or ambiguities in our EFSM model. These were corrected and the results of the modeling were also fed back into the EFSM model. While there was a degree of overhead in creating and maintaining the EFSM documents, they were an extremely useful tool in helping to maintain conformity between the various models. They were also useful in facilitating communication between the various teams using each tool.

3.4 EFSMs as Specifications

There are some issues concerning the appropriateness of using EFSMs as system requirements. In appearance, EFSMs are highly operational in nature. They appear to specify not only what behavior is required of the system, but how the system is to achieve that desired behavior. It is indisputable that they strongly suggest a particular path of implementation. However, the extent to which they actually require a particular implementation depends greatly upon the interpretation of the EFSMs as specifications. One interpretation of an EFSM as a specification is as a set of sequences of input and output behaviors, where a system satisfies the specification provided all possible sequences of input and output behavior of the system are within that set. With this interpretation,

13. The CARA should be able to use a blood pressure from various sources as the input into the CARA algorithm. Blood pressure sources (artial line, cuff, other noninvasive pressures [pulse wave, etc.]) will be prioritized based on quality.
  13.1 A corroborated A-line is use priority 1
  13.2 A corroborated pulse-wave pressure is use priority 2
  13.3 A cuff pressure is use priority 3
15. During resuscitation CARA should respond to any lost blood pressure sources
  15.1 With an appropriate message
  15.2 With a level 1 alarm
  15.3 With a notation in the resuscitation file
18 Mean pressure readings from Propaq must be within 40 - 150 mmHg to be valid throughout resuscitation
27. When the cuff pressure is being used for control, CARA should set a cuff reading frequency based on a ta
In general, blood pressures will be taken more frequently while below the set point.
If the cuff is already inflating for some other reason when the time arives for another reading, an additional cuff reading does not need to be requested.
  27.1 If the mean BP is 60 or below, cuff pressures will be taken once per minute.
  27.2 If the mean BP is (60 - 70], cuff pressures will be taken once every 2 minutes.
  27.3 If the mean BP is (70 - 90], cuff pressures will be taken once every 5 minutes.
  27.4 If the mean BP is above 90, cuff pressures will be taken once every 10 minutes.
28 If CARA can not obtain a valid blood pressure in 3 minutes, it should revert back to manual mode.
  28.1 An appropriate message should be displayed.
  28.2 A level 2 alarm should be issued.
44 If only the cuff pressure is being used and an expected blood pressure reading is invalid
  44.1 An appropriate message should be displayed
  44.2 A level-1 alarm should sound
  44.3 CARA should then initiate another request for a cuff pressure
    44.3.1 If this pressure is invalid,
      44.3.1.1 An appropriate message should be displayed
      44.3.1.2 A level-2 alarm should sound
      44.3.1.3 The system will revert to manual mode
  44.4 Notations should be made to the resuscitation file

Figure 6: Requirements for the Cuff Handler in Auto Control

Variable Name	Description	Written By
CA_Mode	CARA's current mode	Algorithm
CB_alarm_cuffInvalid1	Level 1 Alarm	Cuff Handler
CB_alarm_cuffInvalid2	Level 2 Alarm	Cuff Handler
CB_backManual	Go back to manual mode	Cuff Handler
CB_ctrlType	Type of Control BP	Corroboration
CB_ctrlValue	Value of Control BP	Corroboration
CB_cuffFrequency	Frequency to read Control BP (seconds)	Cuff Handler
CB_cuffSource	Source of Cuff BP Reading?	Corroboration
CB_cuffValue	Value of Cuff BP Reading	Corroboration
CB_drivenVoltage	Voltage to drive pump	Algorithm
CB_iterations	Number of iterations	Cuff Handler
CB_localTimer	Local Timer	Cuff Handler

Figure 7: Variables of the Cuff Handler EFSM

the internal structure apparent in the EFSM is only a suggestion of the internal structure an implementation might have. The implementor is free to seek another architectural design. In this interpretation, EFSMs can be used as system requirements.

If the interpretation of the EFSM is a stronger specification where not only the sequence of values for the input and output variables are recorded, but also the sequence of values for all (global) internal variables, then the internal structure ceases to be a suggestion and in effect becomes a requirement of any system implementation of it. In this case, the EFSM has slid from the role of system requirements to that of architectural design. In our construction of the EFSM specification of CARA, it was our intention that either interpretation could be applied as was found appropriate, and thus, within the limitations of remaining faithful to the original requirements documents, that they could be used in either the capacity of total system requirements, or as a lower-level architectural design, as any team working on the CARA specification would deem appropriate.

4 CARA in SCR

SCR (Software Cost Reduction) is a formal requirements method developed at NRL [7, 6]. It is designed to be used by engineers for specifying requirements of complete systems as well as software. SCR consists of a specification editor, a simulator, and a set of analysis tools. The specification editor allows information about the system to be entered in a tabular form and there are tools for generating a number of visualizations of the design. The editor automatically generates Java code which can be run in the simulator to get immediate feedback on the effects of changing input variables. SCR represents the system as a finite state automata and it defines transitions, input and output variables, and

events in terms of that automata.

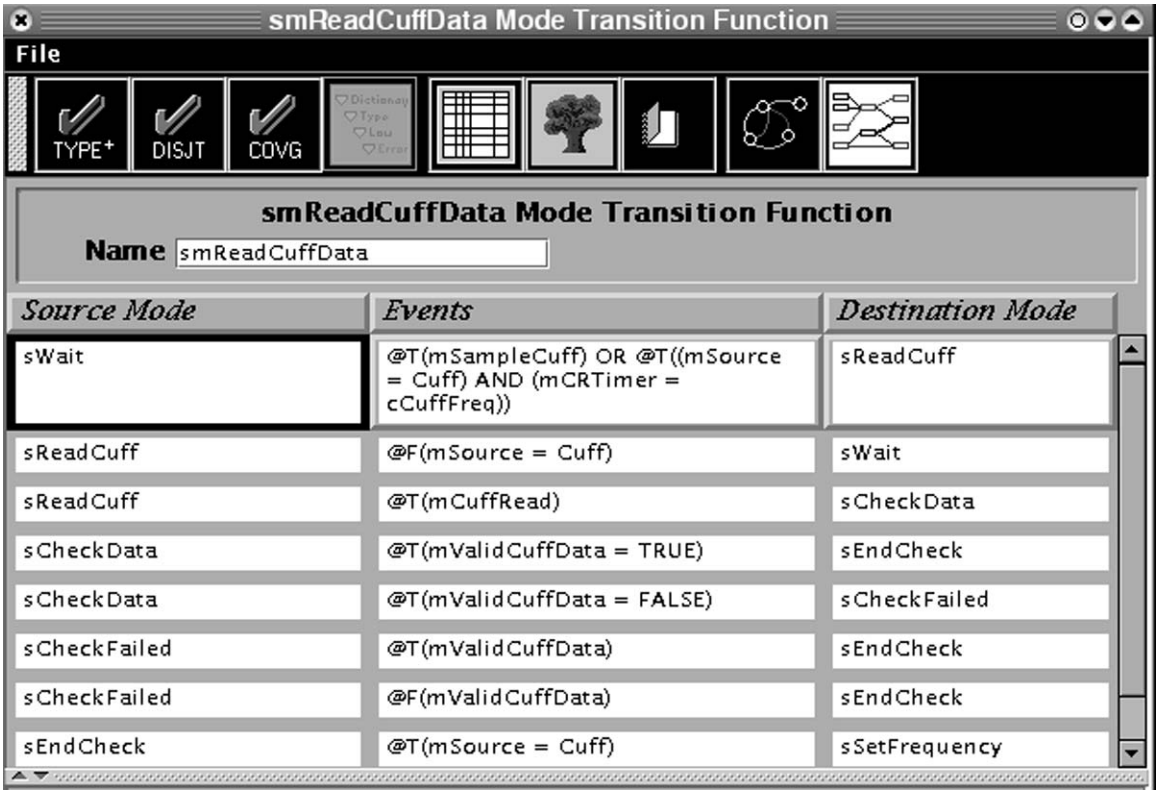

Figure 8: Tabular format of SCR

4.1 Modeling in SCR

Our SCR model of the CARA system incorporates 20 separate automata which can be viewed as working in parallel. Each automaton corresponds roughly to a machine in the EFSM model. The complete model uses 65 monitored (externally controlled) variables, 13 term variables and 38 controlled variables. Mode transition tables are represented in SCR's tabular notation with three columns; one each for starting state, transition condition and end state. This allows users to model state machines by simply putting the information in the right boxes. A state with two exit transitions would have two rows with disjoint transition conditions and destination states but the same starting state. Other tables allow definitions of variables, types and functions.

Initially, we expected that converting our EFSMs into the SCR format would be a fairly straightforward task. We recognized that this problem does not exactly fit the domain of SCR, but we felt that we could learn valuable information

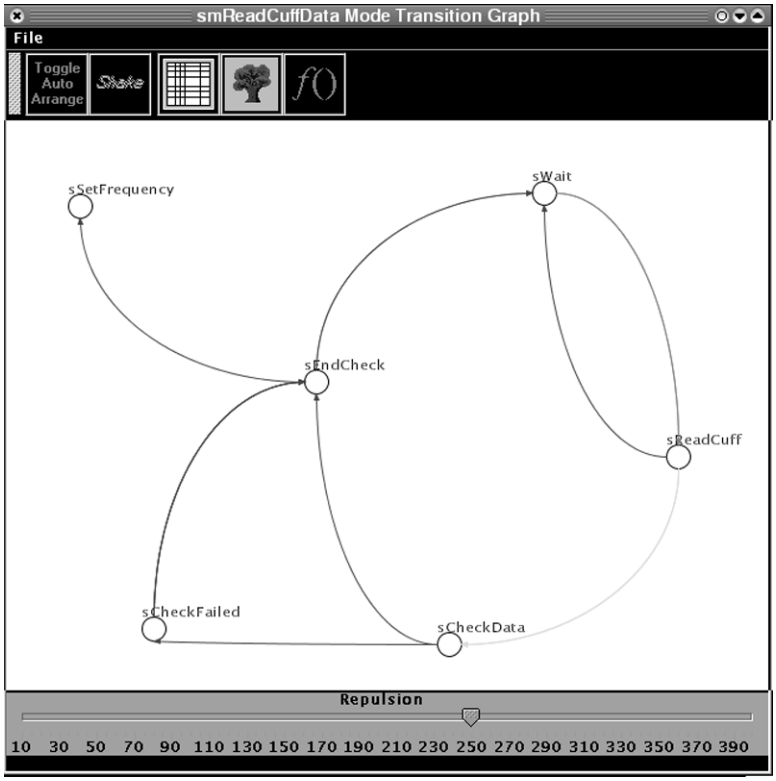

Figure 9: Graphical Output from SCR

about the system using SCR. There is an important assumption implicit to the SCR system which caused us some difficulty. This is the One Input Assumption.

> A basic assumption, called the One-Input Assumption, is that exactly one input event occurs at each state transition[5].

The One-Input Assumption requires that every state transition must be triggered by an input event – an event which arrives from the environment of the state machine. Since our EFSMs have transitions which are not triggered by the arrival of an event, we were unable to simply transfer the EFSMs into SCR.

This problem arises because SCR is designed for capturing black box requirements. We hoped to utilize SCR to capture the formal requirements intended by the set of English language informal requirements we already had. This approach would allow us to use SCR to better define the interfaces between the purely internal components – such as the algorithm for choosing a blood pressure source – and the external components like the blood pressure sensor. Since the English language requirements we were given included a large amount of information about the internal structure of the CARA system, we found it difficult to fully capture the model in SCR.

In the BP Monitor system there is an EFSM[1] which reads and validates the data from the arterial line source. Arterial line readings are more accurate than cuff readings, but if the catheter is incorrectly inserted then the value will be off by a large amount. To guard against this problem, CARA reads in the arterial line data then performs a validity check against a cuff reading. If the reading fails the first time, CARA does another cycle of reading and validation. The EFSM model has a state after the initial validation where the exit conditions are controlled by a boolean condition. If the data is valid the machine ends the check. If the cuff reading is not valid, then another reading is performed. This sort of simple conditional switching is tricky to do in SCR. Because the actual reading is performed in another machine, SCR can not use it as an input and it is necessary to define a new variable to trigger the transition. Rather than using a transition condition like "If the reading is valid then go to the end state", our model has to use "When the validation is done, if the reading is valid then go to the end state". We can not use the reading itself as a condition because its value may not change; that is, it could already be set to true when we enter the check and setting a variable to a value it already holds does not count as an event.

For this reason, instead of creating an exact duplicate of the EFSM model we created a translation which preserved the requirements from the documents while staying as close as possible to the EFSMs. In several cases, we revised the EFSMs to match the SCR translation of the requirements, and in the course of this translation we manually uncovered some more issues with the requirements. Specifically, we added 18 questions to the list. By category, we found 3 inconsistencies, 7 instances of incompleteness and 8 clarifications. An example of an

[1] Screenshots of this state machine in SCR are included in Figures 8 and 9.

important incompleteness discovered during the SCR phase was that although there were conditions for testing whether the impedance of the infusate was within range, other than logging the impedance value, the requirements failed to specify any action to be taken, such as sounding an alarm and cutting power to the pump, when the impedance was out of range. We were most successful with modeling the CARA system's interface with its environment. In particular, we found that the interface with the blood pressure sensor was not clearly defined which resulted in WRAIR clarifying that interface.

4.2 Analysis in SCR

SCR allows us to perform type, disjointness and coverage checks. Coverage checks are limited to condition tables, which can only specify term and controlled variables. In the example from the BP Monitor above, SCR allows us to check that exit conditions from the validation state are disjoint. That is, if the conditions are denoted as C1 and C2, they can not both be true at the same time ($\neg C1 \wedge \neg C2$) as this would allow an unwelcome indeterminency. In addition to this disjointness test, we would like to have a test to guarantee that there exists an action which will allow a transition out of every state ($C1 \vee C2$). This would help to ensure that the machine could not become stuck in a state with no exits. As it is now, SCR will happily pass a model where all the transition conditions are "False".

4.3 Future Work in SCR

Future work in SCR would focus on rounding out our model and completing the sections which we left simplified, such as logging. We may also work more with the SCR Simulator which would allow us to run our specification and see how it responds to input.

5 CARA in Hermes

Hermes [3] is a model checker that supports creating and manipulating hierarchical models. The Hermes input language is based on hierarchical reactive modules [1]. In hierarchical reactive modules, the notion of hierarchy is supported by an observational trace-based semantics and the semantics is used to define the notion of refinement using assume-guarantee rules. Additionally, hierarchical reactive modules support extended finite state machines where the communication is via shared variables. The Hermes implementation has a visual front-end and XML-based back-end.

The central component of the description is a mode. The attributes of a mode include global variables used to share data with its environment, local variables, well-defined entry and exit points, and submodes connected with each other by transitions. The transitions are labeled with guarded commands that access the variables by the natural scoping rules. By using Hermes' modes,

we can specify hierarchically the EFSM's modules including variables. The language distinguishes between a mode definition and a mode reference, which allows sharing and reuse. Assertions associated with control points or modes, and system-wide invariants can be used to specify desired safety properties.

We can specify the Infusion Pump System hierarchically and various properties as boolean valued expression by using Hermes. Hermes exhaustively explores the state space to determine whether the Infusion Pump System satisfies the properties. Consequently, Hermes either verifies the property or provides a counterexample.

5.1 Modeling

Our Hermes model consists of 30 modes executing concurrently and representing the five components of the Infusion Pump System; CARA, pump, caregiver, blood pressure sensor, and patient. A screenshot of the top level of our model in Hermes can be found in Figure 10. In order to derive a Hermes model from EFSM, we need to restrict variables to finite types – boolean, enumerated types, and bounded integers. States and transitions are almost the same as the EFSM's.

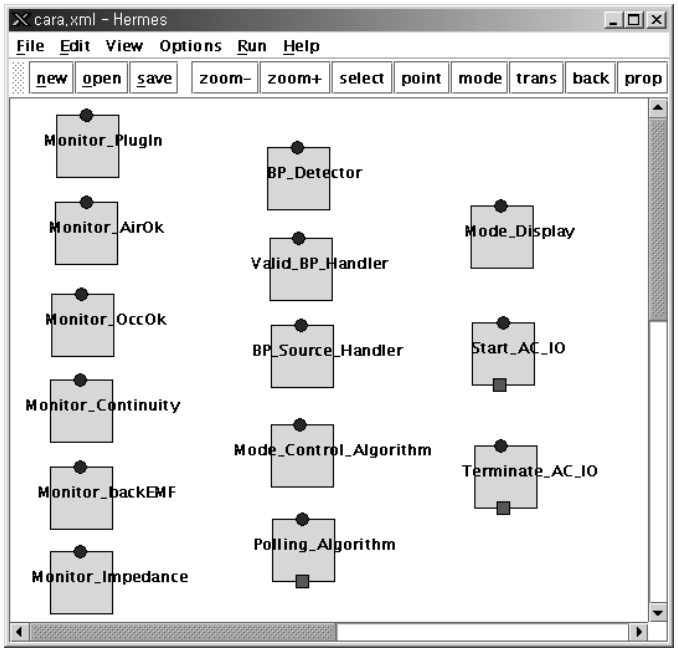

Figure 10: The top level of Hermes model

In this paper, we explain two example modes of the Hermes model: the *Mode_Control_Algorithm* mode and the *Polling_Algorithm* mode. Figure 11 describes the *Mode_Control_Algorithm* mode. In the *Mode_Control_Algorithm*

mode, we specify the four states of CARA - *wait, manual, autocontrol init*, and *autocontrol* – and the *Ask_StartAC* submode. While the transition labels are not made visible in the Hermes display, they have been translated from the transition labels of EFSM. In the *Ask_StartAC* submode, the setpoint value can be changed and *Autocontrol_Init* may be entered by pushing the *StartAC* button.

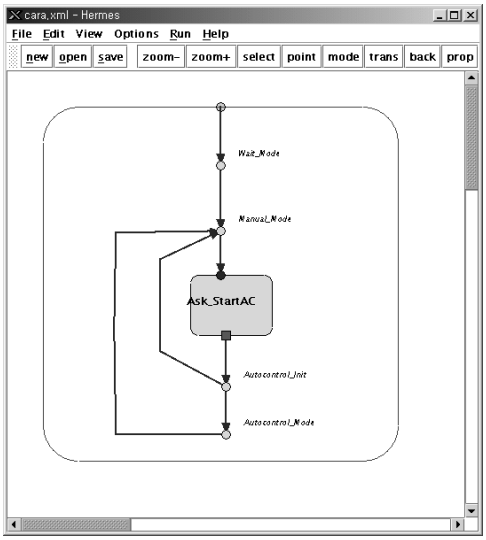

Figure 11: Mode_Control_Algorithm mode

The *Polling_Algorithm* mode is showed in Figure 12. This mode consists of two kinds of submodes, which are *Polling_Back_EMF* and *Polling_Impedance*. The polling algorithm is designed to monitor back EMF and impedance value. The impedance is polled immediately after the back EMF polling, and the back EMF polling and impedance polling are tried at most three times if they fail. If the back EMF or impedance reading cannot be obtained or is zero, the appropriate alarm must be sounded.

5.2 Analysis

Hermes has two mechanisms for describing requirements of a system. A point in a mode of a Hermes model can be given an assertion condition. This is a boolean valued expression that must be true whenever that point is active. An example of such an assertion is that at the *Manual_Mode* point of the *Mode_Control_Algorithm* mode, we always have $CA\_Mode = Manual$. A mode can also be given an assertion condition which must be true whenever any point in the mode is active. We verified that following assertions are satisfied on the corresponding points in modes.

- $(CA\_Mode = Manual)$ at *Manual_Mode* point of the *Mode_Control_Algorithm* mode

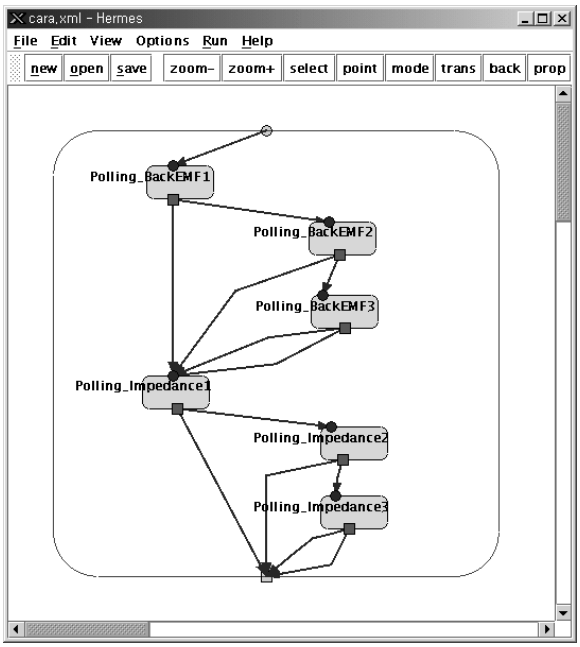

Figure 12: Polling_Algorithm mode

- $(PPu\_PlugIn = true)$ at $PlugIn$ point of the $Monitor\_PlugIn$ mode
- $(PPu\_PlugIn = true) \wedge (PPu\_AirOK = true)$ at $AirOK$ point of the $Monitor\_AirOK$ mode
- $(PPu\_PlugIn = true) \wedge (PPu\_OccOK = true)$ at $OccOK$ point of the $Monitor\_OccOK$ mode
- $(PPu\_PlugIn = true) \wedge (PPu\_ContinuityOK = true)$ at $ContinuityOK$ point of the $Monitor\_Continuity$ mode
- $(PPu\_PlugIn = true) \wedge (PPu\_backEMF\_positive = true)$ at $backEMFOK$ point of the $Monitor\_backEMF$ mode
- $(PPu\_PlugIn = true) \wedge (PPu\_Impedance\_positive = true)$ at $ImpedanceOK$ point of the $Monitor\_Impedance$ mode

Furthermore, a system also can be given an invariant which is a boolean valued expression like an assertion condition. An invariant, however, must evaluate to true for all the states of a system, no matter which modes or points are active. An example of a system-wide invariant one could test is $CB\_Alarm\_BPLow = false$. This invariant means that CB_Alarm_BPLow is always false, that is, BPLow alarm is never sounded. In this case, we do not want it to always be true; we are actually testing that it is possible for the

alarm to sound. Hermes verifies that the invariant does not hold of the Infusion Pump System and gives a counterexample which shows a path where CB_Alarm_BPLow becomes true. The path is useful for designers to make sure thier design. We checked the following false-invariants and got counterexample paths to reach the states where the invariants are false.

- $CA\_Mode \neq AUTO$
- $CB\_Alarm\_BPLow = false$
- $CB\_Alarm\_belowSP = false$
- $CB\_Alarm\_Slope = false$
- $CB\_Alarm\_PollEMF = false$
- $CB\_Alarm\_PollImp = false$
- $CB\_Alarm\_BPInvalid = false$
- $CB\_Alarm\_AllLost = false$
- $CB\_backManual = false$

There are limitations to the verification capabilities of Hermes. At present, Hermes is most useful for finding counter-examples to false conjectures. For verifying true assertions on a system of the size of the Infusion Pump System, with the current version of Hermes we encounter the state explosion problem and Hermes fails to successfully terminate. Future versions of Hermes may be able to remedy this problem, at least for systems of this size. Another limitation of Hermes is inherent in the system. It allows us to test various safety properties provided we can formulate the safety properties as boolean expressions of values of the system variables. However, it does not allow us to guarantee liveness properties. In general, we can not use Hermes to prove that eventually some good state will be reached. For this a system with a more expressive logic would be required.

6 Related Work

Below we describe other work that has been done to model real-world systems using finite state machine techniques, and to use that model to verify properties of those systems, and we discuss some of the differences between that work and ours.

In [4], Heitmeyer *et, al.* describe their effort to prove system invariants for the Software Requirements Specification for a Weapons Control Panel. The main emphasis of this work is to describe a set of abstraction techniques that could be applied automatically to reduce the size of the model to render it practical for purposes of model checking. Properties in the abstract model were checked using Spin through an automatic call from SRC. Once a counter example was

found in the abstract model, it was translated back to the full model and tested using simulation in SCR. One major difference with this work and ours is that we had to begin with a specification document written in English prose, with a sequence of modifying and overriding appendices, while the Software Requirements Specification with which they worked was a much more complete and precise semi-formal description needed in a "build-to" specification, including such things as listing of input and output variables and a description of output values as a function of inputs. Developing a detailed formal software requirements specification from informal English narrative was part of our endeavor. Also, in that paper, the properties that were checked were state invariants (true in every state with no reference to earlier or later states), and transition invariants (true of all pairs of pre and post states of all transitions). Here, we are building a framework to allow broader classes of properties to be checked, possibly using different tools, but always based on the same model.

In [10], an example is given of a formal specification of the mode logic of a flight guidance system. The specification utilizes a formalization based on that of SCR with several extentions such as transition buses and hierarchical organization. This formalism is esentially the same as that used in this paper. In that work, however, the formalism is incomplete in that they lack a complete formal semantics, particularly for concurrent hierarchical finite state machines. This work provides a thorough system description, but does so using informal syntax which did not admit any automated checking of properties of the specification.

[8] describes the application of the SCR* toolset to a communications security device. After converting prose requirements to SCR, the authors were able to check consistency and completeness with the SCR tools and prove properties about the system by plugging SCR into the SPIN model checker and TAME / PVS theorem prover. The CARA system is significantly larger than the comsec device (116 variables versus 39), but the details of modeling in this formalism are basically the same. A significant difference is that their work started with prose (natural language) specifications, while we were able to start with a formal EFSM specification. This sped up modeling significantly and facilitated using SCR in parallel with other tools.

7 Conclusion

We have described our on-going project using formal methods tools for the specification and analysis of the CARA based infusion pump system. Our preliminary work is based on SCR and Hermes. As mentioned in the paper, we were able to discover several inconsistencies and incompleteness in the informal requirements that were provided to us for our study.

The current work is to identify safety properties that CARA should satisfy and to model check them using Hermes. Another ongoing effort is to model the system using CHARON [2]. CHARON is a language for modular specification of interacting hybrid systems based on the notions of agent and mode. Hybrid systems exhibit behaviors that are continuous as well as discrete. The infusion

system can be naturally specified as hybrid systems. The main advantage of using CHARON is that the reference specification in EFSM can naturally be described in CHARON without any changes. In addition, it is possible to specify explicitly the model of a patient (based on the known continuous flow model of a human heart) and analyze the infusion pump system with a given particular type of patient.

While the Cleanroom process was used in developing the initial English language requirements at WRAIR, it did not employ the type of formal modeling and analyis techniques such as model checking that we have described in this paper. Ideally, such formal methods and analysis techniques should be applied from the beginning of the development process of safety-critical medical systems such as CARA.

8 Acknowledgment.

We would like to thank Alwyn Goodloe and Jitka Stribrna for their participation during the initial development of EFSM and SCR specifications. We also would like to express our appreciation to Paul Jones at FDA and David Hislop at ARO for their support and encouragement.

References

[1] R. Alur and R. Grosu. Modular refinement of hierarchic reactive machines. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 390–402, 2000.

[2] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specifications of hybrid systems in CHARON . In *Proceedings of Hybrid Systems: Computation and Control, Third International Workshop*, volume 1790 of *LNCS*, pages 6–19. Springer-Verlag, 2000.

[3] R. Alur, R. Grosu, and M. McDougall. Efficient Reachability Analysis of Hierarchical Reactive Machines. In *12th International Conference on Computer-Aided Verification, LNCS 1855*, pages 280–295. Springer Verlag, 2000.

[4] Constance Heitmeyer, James Kirby Jr., Bruce Labaw, Myla Archer, and Ramesh Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE TSE*, 24(11):927 – 948, November 1998.

[5] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, July 1996.

[6] K. Heninger, D. L. Parnas, J. E. Shore, and J. W. Kallander. Software requirements for the a-7e aircraft. Technical Report 3876, Naval Research Lab, Washington, DC, 1978.

[7] K. L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *Software Engineering*, 6(1):2–13, January 1980.

[8] James Kirby, Constance Heitmeyer, and Myla Archer. Scr: A practical approach to building a high assurance comsec system. In *15th Annual Computer Security Applications Conference (ACSAC '99)*, pages 109 – 118. IEEE Comp. Soc. Press, 1999.

[9] C. Mazza, J. Fairclough, B. Melton, D. de Pablo, A. Scheffer, and R. Stevens. *Software Engineering Standards*. Prentice Hall, 1994.

[10] Steven P. Miller. Specifying the mode logic of a flight guidance system in core and scr. In *Proceedings of the second workshop on Formal methods in software practice*, pages 44–53. ACM Press, 1998.

[11] WRAIR Dept. of Resuscitative Medicine. Cara questions. Proprietary Document, WRAIR, Silver Spring, MD, January 2001.

[12] WRAIR Dept. of Resuscitative Medicine. Cara tagged requirements. Proprietary Document, WRAIR, Silver Spring, MD, March 2001.