APPLICATION OF THE MULTIGRAPH SOFTWARE ARCHITECTURE TO INTELLIGENT PATIENT MONITORING

BY

PIERRE JAMALI

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF DOCTOR OF PHILOSOPHY IN MEASUREMENT AND INFORMATION IN MEDICINE

CITY UNIVERSITY

LONDON EC1V 0HB

DEPARTMENT OF SYSTEMS SCIENCE

CENTRE FOR MEASUREMENT & INFORMATION IN MEDICINE

SEPTEMBER 1995

To my parents

Contents

List Of Figures

List Of Tables

Acknowledgements

I would like to thank my supervisors Prof. Ewart Carson and Dr. Ron Summers for giving me the opportunity to work on this exciting and challenging project. I am also grateful for their contributions throughout the project and suggestions for improving this dissertation. Equally I need to express my gratitude to our collaborators, Prof. Janos Sztipanovits and his team at Vanderbilt University, especially Dr. Gabor Karsai and Dr. Csaba Biegl for their support in using the Multigraph architecture and Dr. Ben Abbot for installing the Intelligent Process Control System software. Many thanks also to Dr. Cliff Morgan at the Adult Intensive Care Unit (AICU) of The Royal Brompton Hospital for his enthusiasm towards the project and also for taking on the unenvied task of educating an engineer in the intricacies of pulmonary physiology. I am also indebted to Dr. Michael Lunn, Senior House Officer at the AICU, without whose efforts, the clinical data collection would have been impossible. I wish to thank Dr. Roman Hovorka for his help with the use of the HUGIN software; and Massoud Pirjamali for proof reading the first draft.

Finally, my loving thanks to Faye for her patience and support especially during the final year of the project.

Abstract

This project is concerned with the development of intelligent agents for use in the Intensive Care Unit (ICU), based on the Multigraph Architecture (MA). The objectives of the work are twofold:

i.) To consider the design issues of intelligent systems, using a model-based approach of adding intelligence to existing instrumentation and information systems, irrespective of their implementation.

ii.) More specifically, to investigate the role of MA in the implementation of such systems.

The first objective will be achieved by writing a specification for an Integrated Model-based Development Environment (IMDE), identifying its functionality, user requirements, *etc.* The second objective is to implement a prototype of one such development environment. This will borrow concepts from a framework for intelligent process control for chemical plants, previously implemented in the MA. Finally an ICU application will be developed, using this IMDE, to monitor the respiratory system.

The benefits associated with using such a model-based framework include modular design, reusability of software components, automatic synthesis of run-time system from the models and simpler consistency and validation procedures. Another benefit that is of particular interest in patient monitoring systems is the integration of different software components. Because the MA is generic, it can be extended to support any modelling paradigm. This means that the development framework can handle all aspects in design of the system, *e.g.* instrument interface, user interface, signal processing and knowledge-based signal interpretation. To demonstrate this, the IMDE has been extended to include the Causal Probabilistic Network (CPN) modelling paradigm as an integral component. This facilitates the modelling of uncertainty.

Using the techniques described in summary above, the benefits gained from applying the multigraph architecture to intelligent patient monitoring can be ascertained. This forms the bulk of the work to be described in this thesis.

CHAPTER 1

INTRODUCTION

1.1 Background

This dissertation describes an intelligent monitoring system for use in the High Dependency Environment (HDE). The aim of the work, involving co-operation between the Research Centre for Measurement and Information in Medicine (MIM) at City University and the Royal Brompton National Heart and Lung Hospital (refered to throughout as RBH), is to devise novel forms of intelligent instrumentation for application in intensive care medicine. The research is based on a specific approach to design and implementation of intelligent systems, called the Multigraph Architecture (MA), developed at Vanderbilt University, Tennessee. Using this approach, techniques of model-based monitoring and artificial intelligence are merged to add intelligence to instrumentation available. The clinical application domain is adult intensive care, in particular the management of artificially ventilated post-operative cardiac patients. Software development took place in the MIM centre and the clinical aspect of work was carried out at the RBH.

The following sections describe the motivation and background of the work, both from a clinical and software design perspective, leading into a discussion on issues concerned with the need for a model-based framework for intelligent systems. The statement of the aims and objectives of this research programme and an outline plan for the rest of this dissertation complete the chapter.

1.2 Clinical Perspective

1.2.1 High Dependency Environment

The High Dependency Environment (HDE) is a collective term for any dedicated hospital unit for patients requiring specialised or intensive care. These may be general or special Operating Rooms (OR) or Intensive Care Units (ICU). The HDEs were formed for the following reasons:

- to reduce mortality and morbidity,
- to effectively manage the use of scarce, human and equipment, resources[1],
- to facilitate research.

Measurement is the cornerstone of identifying current patient state and its trajectory. Data, generated either from the measurements or through assessments made directly by clinical stuff, underlie every clinical decision. By definition patients in HDE require intensive **monitoring**, where monitoring is defined as measurement of patient physiological variables together with subjective assessments. Monitoring leads to therapeutic intervention if deemed necessary by the clinical staff. Modern HDEs are characterised by the availability of a large amount of patient data.

[1] The use of HDE in a hospital, leads to the availability of a high ratio, typically greater than one, of specialised nursing staff to patient. Also it makes it possible for the complex and expensive patient care equipment, found in the HDEs, to be utilised for a high percentage of their functional lives.

1.2.2 Data Complexity

Advances in measurement and computer technologies have resulted in an increase in the number, frequency and degree of invasiveness of parameters routinely monitored in HDE (Carson *et al.*, 1991). For instance in 1992, real-time intra-arterial blood gas monitors were introduced, such as the PB3300 manufactured by Puritan-Bennett. Previously measurements of blood gases were only possible through off-line measurements on blood samples. The PB3300 facilitates the continuous monitoring of blood gases which represents an increase in the available frequency of patient measurement. Real-time blood gas monitoring is more invasive than off-line (discontinuous) blood gas monitoring because it requires the placement of the infra-red probe in the artery. Another example of data explosion in the HDE is the ventilator. Modern ventilators with data ports can create up to 8Mb of data every day. This explosion in volume of data and their interdependent nature create problems of information management.

1.2.3 Alarms/Alerts

Another problem associated with modern HDEs is the management of alarms and alerts. These may be either equipment malfunction alarms or signal set point alarms. The problems of HDE alarms are caused by:

- the large number of alarms and their associated audio warnings, (*e.g.* the Ohmeda 5250 respiratory gas monitor has 91 alarms and warnings),
- the large number of sources of alarms,
- use of single variables to generate alarms,

- use of hard thresholds to generate alarms, hence the problem of determining alarm thresholds.

If an alarm is not generated when it should (false negative), the well being of patient will be threatened. If it is activated without proper cause (false positive), it can be annoying to the clinical staff and may lead to it being ignored, silenced or disabled. The allowable range of alarm thresholds is a compromise between the probability of false alarms and missing alarm events. This is further complicated by the presence of artifacts. Physiological signals by nature are prone to corruption by artifacts. In situations when there are too many false alarms the clinical staff will effectively disable the alarms by setting the threshold levels such that they are never triggered. This may jeopardise effectiveness of patient care. Therefore there is a need to improve the reliability, specificity and usefulness of alarms by implementing integrated, intelligent alarms (Garfinkel *et al.*, 1988; Koski *et al.*, 1990; Mäkivirta, 1989; Schecke *et al.*, 1992; Shabot *et al.*, 1990; Van der Aa, 1990). See chapter 2 for a review of the work on intelligent alarms.

Another reason for intelligent monitoring is to avoid disasters that may be caused by human error, either due to general fatigue or boredom associated with monotonous monitoring tasks, or caused by **cognitive overload.** Cognitive overload (or cognitive indigestion) is the problem of intake of information that a human operator is faced with when required to monitor multiple channels of high rate data. Studies by instrument manufacturers, such as one carried out by Hewlett-Packard have shown that "under conditions of high mental load, clinically important data can go unnoticed" (Higgins, 1992; p. F5), and unless "clinicians specifically allocate attentional resource to the perception of a numeric display, the

only thing that they will be able to reliably report about the numeric is its colour" (Higgins, 1992; p. F5).

1.2.4 Role Of Patient Data Management Systems

Patient Data Management Systems (PDMS) for use in intensive care units and other high dependency environments are a relatively new concept. There may be many reasons for their introduction, but the most obvious one is to release the nursing staff from the clerical work associated with patient charting. Up to a third of HDE nursing time can be spent on keeping records and patient charts (Thull *et al.*, 1992). The CareVue 9000 system manufactured by Hewlett-Packard is an example of a networked PDMS designed to obviate the need for manual paper logging in patient charting. In general, PDMSs address some of the problems of integration of data from different sources. However since they were designed as a replacement to paper charts, with medical audit as a major drive, they do not offer any intelligent functionality. In other words they may help the nursing staff in patient charting but they do not offer any aid to the physicians in assimilation or interpretation of data. The emphasis is on recording of data rather than their interpretation. So much so that data visualisation (*i.e.* aiding the clinician with data interpretation through graphical presentation of data and their trend) is largely ignored, and the interface from instruments cannot be used for high rate data.

It can be concluded, therefore, that there is a clinical need for enhanced capability of converting data into information, so that as much support as possible is given to the clinician as the decision-maker.

1.3 Software Design Perspective

The term **intelligent** with regard to intelligent monitoring has taken on different meanings over the years. In the 1970s microprocessor based instruments were called intelligent because they were programmable and offered some degree of flexibility. The current generation of medical instruments is commonly referred to as **smart** instruments. The smart features include, simple limit alarms, historical trend display, reconfigurable display and some self diagnostic ability. In the control engineering community, expected features of intelligent process monitoring include: autonomous behaviour, adaptivity, robustness, signal interpretation and more recently, diagnosis.

Our understanding of intelligent monitoring, while including the above features, goes further by embracing models. Models are abstract representations of a system for a specific purpose. Recently there has been an increasing use of models in knowledge-based systems, as a structured way of representing knowledge, and with corresponding model solutions as the reasoning method. This has led to a realisation in the Artificial Intelligence (AI) community that knowledge engineering is really a modelling activity (Ford, 1993). Summers and Carson (1991) were among the first to take this view and proposed a validation methodology for knowledge based systems by drawing parallels with the model validation process. Perhaps the most important reason for using models in intelligent software is to help manage complexity in large systems (Harel, 1992).

Therefore intelligent monitoring systems should contain explicit knowledge about the system to be measured or monitored in terms of constituent models. As

software becomes more complex it increasingly reflects the complexity of its environment. Therefore an intelligent HDE agent must contain models of the patient as well as models of HDE activities.

Requirements often identified as issues for a software architecture to implement intelligent patient monitoring are:

- real-time capability,
- reusability,
- portability,
- model-based signal interpretation.

The availability of real-time data and the requirement of timely response to patient changes impose the need for real-time capability on the intelligent software for patient monitoring. The development of real-time software becomes an issue because in practice it often deviates from the software engineering methods. Also real-time software is different from non real-time software in that the correctness of the system is closely interrelated with its performance (or speed).

The reusability is important because often different applications may have many parts in common; which parts may potentially be reused. If the architecture does not promote reusability the development of new applications will become expensive and slow.

Unless the implementation is easily portable, it will be limited to a specific hardware platform. Thus applications developed under one platform will not be usable on others.

Model-based signal interpretation is important because the use of compiled knowledge such as rules is not adequate to capture the inter-dependence of physiological variables. Models allow for coherent and consistent representation of the available knowledge in a manageable way.

1.3.1 The Need For A Framework

When designing intelligent monitoring systems there are many issues that need to be addressed. First of all there is a variety of techniques available for interpreting the data, such as rule-based reasoning[2], fuzzy logic[3], Bayesian belief networks[4], and artificial neural networks[5]. Use of these techniques has been proposed to cope with problems created by the nature of data, such as noise, uncertainty and the complex dynamics. The difficulty associated with selecting what technique to use is also complicated by the fact that these techniques are prone to misuse. Other aspects to be considered are;

- data acquisition *i.e.* automatic data collection from a variety of sources,
- the presentation of information and visualisation of data,

---

[2] A knowledge base globally structures the rules as an acyclic tree. The logical implication paths $A \rightarrow B \rightarrow C \rightarrow D \rightarrow \ldots$ flow from the tree's root node to its leaf nodes. The inference process is defined by the enumeration of logical paths, using a search of the knowledge tree. Forward chaining inference proceeds from antecedents to consequences. Backward chaining inference proceeds from consequences to antecedents or hypotheses. In this way rule-based systems can be considered as structured symbolic estimators.

[3] Multivalued set theory, used to express inexact relations between concepts in a knowledge base. It is fundamentally a numerical framework as compared with the symbolic framework of rule-based systems.

[4] This is a numerical structured framework for probabilistic reasoning.

[5] A distributed parallel model of information processing inspired by the organisation and function of the neurones within the brain. They can generalise (learn), after training, work with incomplete or imperfect data, providing a degree of fault tolerance. In this way artificial neural networks can be considered as numeric unstructured estimators.

- the design of the user interface,
- time-critical or real-time behaviour.

Monitoring software differs from other software in that the input to the software, if not wholly at least partly, comes from the instrumentation. Therefore they need to be able to directly capture the data generated from the instrumentation. These data or a summary of them often need to be visualised for effective transfer of information to the human observer. The design of the user interface is crucial to the acceptability and usability of any software application. Finally, real-time monitoring imposes further constraints on the software. The real-time constraint, in its most relaxed form implies that the software must be capable of continuous operation, and in its most rigid form it demands the software tasks to be guaranteed to complete within specified deadlines.

Having overcome these hurdles in the design stage, there is then the software implementation to consider. Therefore implementation of intelligent real-time monitoring systems can become a formidable task.

The use of a **framework** is proposed as a means of providing a solution to the problem of design and implementation of intelligent software for patient monitoring. A framework is like a template of an application program. In Object Oriented (OO) software terminology, the term "framework" is defined as "a set of classes that embodies an abstract design for solutions to a family of related problems" (Johnson, 1988; p. 22). A framework is supported by a software architecture; it contains a set of abstractions[6], primitive objects[7] or basic software

[6] Here by abstraction is meant a set of concepts which are used to model the domain.

[7] Object is a region of memory to which a type can be applied.

modules[8] and sometimes generic reusable modules, and most importantly it has an infrastructure, *i.e.* much of the functionality already exists and the relation between abstractions is defined. Because a framework implements some common functionality much less code needs to be written and applications become smaller, and easier to debug. Another benefit comes from the use of abstractions. They help in every stage including knowledge acquisition and organisation (Pirjamali *et al.*, 1993). Other benefits include reusability of software components including both software objects and functions.

Examples of real-time frameworks for intelligent monitoring are RTworks[9] (Laffey, 1991) and Intelligent Process Control System (IPCS) (Karsai *et al.*, 1992). RTworks supports a distributed architecture based on the client-server model. It has generic modules such as interface and visualisation modules. It has an infrastructure and, for instance, the interactions between user interface module, data modules and other modules, are already implemented. Similarly IPCS is a framework, because it has an underlying architecture, some pre-defined abstractions, as well as runtime support for some algorithms. More importantly it integrates all of these aspects, in other words it has an infrastructure whereby the way that different components interact is defined.

[8] A software module implements a subset of the functionality of an application.

[9] RTworks is a commercial product developed by the Talarian Corporation.

1.3.2 Multigraph Architecture

The Multigraph architecture (MA) (Biegl, 1988; Sztipanovits and Bourne, 1985) is a generic framework for design and implementation of real-time intelligent systems. MA is a model-based architecture which allows direct use of models *i.e.* the computational structure is directly derived from models. The term **model-based** implies that the run-time system is derived from, and is dependent on, the models. It should not be confused with the use of first principle knowledge of physiology ("deep knowledge-based"). The models, containing abstract representations of the system of interest which embody the expert knowledge, are represented by graphical and syntactic structures.

1.4 Aim And Objectives

Given the need for intelligent patient monitoring in the HDE and the software requirements of such systems, as discussed in the previous sections, the aim of this work is to investigate the suitability of the MA for building intelligent HDE agents and to lay down the basis for design of a generic model-based patient monitoring system. The objectives are:

- to design a generic specification of a framework,
- to identify relevant modelling paradigms and software components and implement their integration into a prototype framework,
- to build and test an application prototype.

1.5 Layout Of Thesis

In this chapter the motivation and objectives of the work have been explained as well as the approach taken. Along the way definitions have been developed for the terms, **framework**, **model-based**, **intelligent** and **monitoring**. Chapter 2 provides a critical review of the literature currently available that has a similar scope and objective to those described in this thesis. Chapter 3 presents the MA, and its theoretical foundation, namely the graph computational model, as well as IPCS, as an example of application of MA concepts in intelligent monitoring. In chapter 4, a generic specification is given for a model-based framework for applications of intelligent monitoring systems in the HDE. Chapter 5 describes some extensions and modifications of the MA framework necessary for patient monitoring. Chapter 6 contains the clinical application of the current prototype, which consists of a respiratory monitoring system. Chapter 7 discusses the model-based software synthesis, summarises the design steps and discusses the application of the technology to patient monitoring, based on the experiments that have been carried out. Finally chapter 8 contains the conclusions and the recommendations which describe the advances made in this research programme.

CHAPTER 2

REVIEW OF APPLICATIONS OF INTELLIGENT MONITORING IN THE HDE

2.1 Introduction

The principal purpose of this chapter is to review the published research work related to intelligent patient monitoring in the High Dependency Environment (HDE). The aim is to show how the work described in this thesis fits in with what has already taken place and what other researchers are doing. The chapter starts with a wider historical view of Artificial Intelligence (AI) applications in HDE, and then focuses on some specific projects for a more detailed review. Finally some general conclusions are drawn about the current state of the technology.

2.2 Historical Review Of AI Applications In The HDE

In 1960 McCarthy published his work on the development of LISP (McCarthy, 1960; McCarthy, 1978). This landmark development in AI almost coincided with the first computer applications in measurement of physiological signals for patient monitoring in the early 1960s. Over the next two decades AI grew rapidly and expert systems[10] emerged. Simultaneously there was a rapid growth of the patient monitoring technology in intensive care. As the high volume of data generated by the instrumentation in HDEs began to outstrip the human operators' ability to handle the information some AI researchers turned their attention towards the problems of data management and interpretation in HDEs.

[10] Computer programs for a well defined and specific problem domain, that try to mimic domain experts' problem solving behaviour, by incorporating factual and heuristic knowledge, typically encoded as rules.

The research in applications of AI for intelligent patient monitoring in HDEs started in the late 1970s and has resulted in a large body of work; the earliest example of which is the pioneering work of Fagan (1980) on ventilation management, called the VM system. VM was designed to interpret on-line data in an ICU for the management of post-surgical patients undergoing mechanical ventilation. The implementation was based on the formalism of MYCIN (Shortliffe, 1976). The domain knowledge was represented implicitly and no abstractions were defined except to classify the rules into five groups. These rule groups were, initialising rules, status rules, therapy rules, transition rules and instrument rules. VM remained a development system.

Perhaps the most exciting development of the 1980s in intelligent patient monitoring was the COMputerised Patient Advice System (COMPAS) project, developed at University of Utah (Sittig *et al.*, 1989). COMPAS was designed to advise clinicians in the management of patients with Adult Respiratory Distress Syndrome (ARDS). It was developed within the existing home-made hospital information system, called the Health Evaluation through Logical Processing (HELP) system (Pryor *et al.*, 1983). COMPAS was a prototype of computer-based, protocol guided, patient care. The patient management protocols were derived from skilful knowledge elicitation based on round table discussions of nine pulmonary specialists. The protocols were subsequently modified to increase their acceptability to clinical users. The rate of compliance of the physician with the system's suggestion is reported to be 92%. COMPAS is significant for a number of reasons. Firstly, it is one of the very few applications to undergo clinical trials. Secondly, it has been shown to have a direct impact on the quality of patient care, and clinical outcome. It is claimed that the increase from 10% to 40% in the survival rate of the most severe form of ARDS patients is attributable to the use of

the system. Finally, the COMPAS project is still in routine clinical use under the name of CORE (Henderson *et al.*, 1992).

COMPAS belongs to a class of applications which are aimed at adding intelligence to patient monitoring, primarily by incorporating the capability of giving advice. Other similar applications include AIRS (Summers *et al.*, 1993), ESTER (Hernández-sande *et al.*, 1989), WEANPRO (Tong, 1991). Some researchers on the other hand concentrated on the problem of intelligent alarming. PONI (Garfinkel *et al.*, 1988) was aimed at reducing the number of false positive alarms in the operating room by rule-based alarm validation. PONI was implemented within a home-made Hospital Information System (HIS) called the Hospital Operating Room NETwork (HORNET) (Garfinkel *et al.*, 1987). The intelligent alarms in anaesthesia project, detects faults in breathing circuit, using an expert system approach (Van der Aa, 1990). Koski *et al.* (1990) investigated application of median type filters for improving reliability of alarms.

2.3 Critical Review Of Real-Time Monitoring Applications In The HDE

The focus of interest in this thesis is the HDE applications of real-time monitoring, among which five have been selected for in-depth review. The five projects reviewed here have been selected because they are the closest in scope to this dissertation; *i.e.* they are all concerned with the wider problem of intelligent patient monitoring rather than expert systems solutions for restricted and isolated problems. They all take a common approach of implementing intelligence by integrating knowledge-based systems with the existing HDE instrumentation.

Finally all five projects are in some way related to the respiratory monitoring and management of ventilators.

These five projects have also been selected as being representative of the prevailing techniques, and contain some originality in their approach. The Intelligent Cardiovascular Monitor (ICM) is reviewed because it is based on a parallel architecture. GUARDIAN is considered because it interfaces to instruments for direct real-time data acquisition and because of its distributed architecture. SIMON is included because it adopts a model-based framework. PATRICIA is described because it is a typical rule-based approach. Finally, the AIM-INFORM project has been included, because it attempts to lay the foundations for a framework for Knowledge Based Systems (KBS) in HDE applications.

2.3.1 ICM

The Intelligent Cardiovascular Monitor (ICM) was developed by Factor and colleagues at the University of Yale (Factor *et al.*, 1990). The ICM was designed as a prototype application of the parallel software architecture called *process trellis*. The trellis model of parallelism is based on a flat computational graph. The so-called hierarchical approach of ICM is really a layered view of the different stages of signal transformation from raw data to qualitative data to higher level physiological states. The addition of a higher level, therapeutic layer is possible but not implemented. The ICM contains about 70 modules for processing of real-time data. The prototype was tested with patient data during open heart surgery and

was reported to produce "reasonable" results. No further information on the test results is available in the literature.

The process trellis is a parallel architecture that runs on a parallel computer. It allows for the program to be divided into sub-tasks. The sub-task can be executed simultaneously on CPUs of a parallel computer. The ICM project is specially important because it is one of the few that actually addressed the processing power requirements for the interpretation of multiple channels of real-time data, by incorporating facilities for parallel execution.

2.3.2 GUARDIAN

The GUARDIAN project is a prototype intelligent monitoring system for the ICU, developed by the Knowledge Systems Laboratory of Stanford University (Hayes-Roth *et al.*, 1992). The application area is the monitoring of post-operative cardiac surgery patients. The project was developed as a proof of concept system, not designed for practical application. It is based on a Blackboard architecture (Hayes-Roth, 1990), where the knowledge is organised into a collection of independent knowledge sources (KS). The architecture of blackboard systems will be described in more details in section 2.3.5. The knowledge in GUARDIAN is represented using a conceptual graph formalism. The system implements a scheme for data reduction by changing the sampling frequency and filtering thresholds (Washington and Hayes-Roth, 1989). The validation of the system has been in the form of investigating its use in medically simple scenarios. The system monitored a patient and ventilator simulator, using 20 patient variables. The real-time performance of GUARDIAN was tested by presenting the system with four events, which the knowledge sources of the system knew about. The utility of the GUARDIAN's

response was calculated using a combination of correctness, specificity, timeliness and the criticality of the event presented. A high utility was reported.

GUARDIAN seems to be more of a prototype exemplar for AI techniques rather than trying to address real problems of clinical monitoring.

2.3.3 SIMON

The SIMON project represents on-going work in the Department of Electrical Engineering of Vanderbilt University (Dawant *et al.*, 1993). It has been named SIMON for Signal Interpretation and MONitoring. The initial application domain is management of assisted ventilation of premature infants with respiratory distress syndrome (RDS). Two major issues addressed by SIMON are context dependence and correctness of measurements.

SIMON uses a model based approach. The modelling paradigm, YAQ, (Uckun *et al.*, 1992; Uckun *et al.*, 1993) is based on the work by Forbus on Qualitative Process Theory (QPT) (Forbus, 1984). Qualitative modelling may seem an attractive option in the physiological domain where, exact mathematical representations are not always available. However, the ambiguity in qualitative simulation, is a major drawback of qualitative modelling. Qualitative simulation differs from numerical simulation in that it attempts to predict all possible behaviours. Thus a qualitative simulation produces a set of behaviour sequences. As well as lack of precision, a practical implication of this inability to reduce the solution space, is rapid and non-linear growth in the simulation time. YAQ attempts to contain this exponential growth of the solution space by limiting the

simulation at each stage to the generation of the immediate expectations for each quantity. Also no attempt is made to combine predictions of different quantities. Furthermore, YAQ supports the use of numerical as well as symbolic information in order to help resolve future ambiguities.

The architecture of SIMON is based on four major modules, each implemented as an independent UNIX process, (see figure 2.1). The modules communicate through a custom interprocess communication (IPC). The four modules are; data acquisition, data abstraction, patient model simulation and graphical user interface module.

At the time of writing the SIMON project has not been sufficiently developed to undergo test and validation.

2.3.4 PATRICIA

PATRICIA, an intelligent monitoring system for the management of artificially ventilated patients, was developed at the University of La Coruña, Spain (Moret-Bonillo *et al.*, 1993). Essentially, PATRICIA is a rule-based system based on heuristic expert knowledge. A context dependent transformation between numeric values and symbolic representation is employed. The use of "context" here is static *i.e.* it can be set to different values at the start of a session but it is not modified while the system is running. For instance the context can be patient age or history but not the current patient state, since the latter may change during a session.

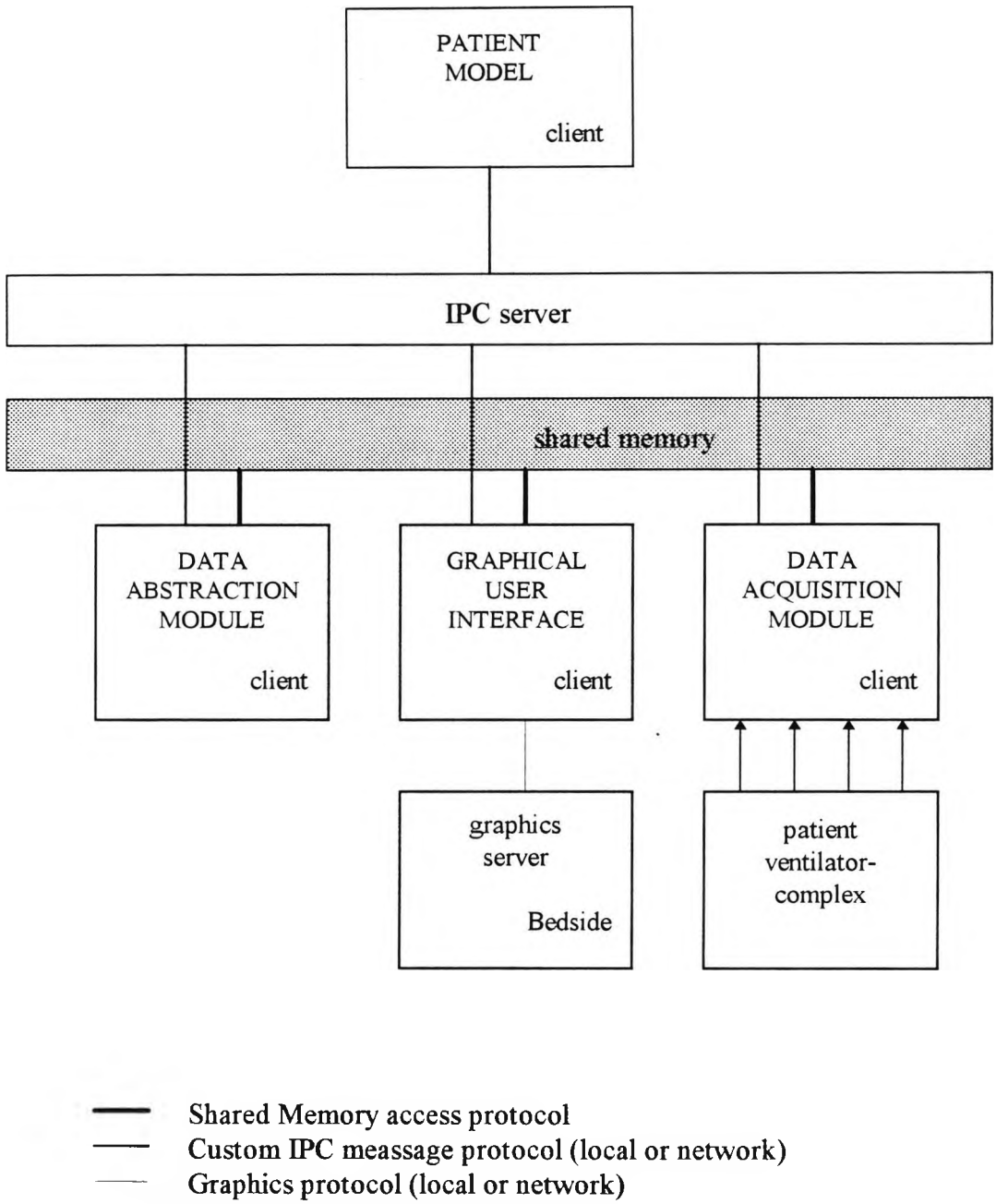

Figure 2.1 The Conceptual Architecture Of SIMON (after Dawant *et al.* 1993)

Each variable is sampled at a rate determined by its "*persistence*"[11] and this sampling rate may be changed dynamically. Numerical data are checked to see if they fall within physiologically plausible ranges. The rule-based inferencing is invoked at regular intervals determined by "*Decision Time*"[12], or alternatively when an alarm condition is detected. Artifact detection rules have been proposed to trap invalid symbolic data by considering causality among related variables. The so-called temporal rules use the value of a variable or its trend **during** the *decision time* as their antecedent part.

PATRICIA was validated by assessing the rate of agreement of the system with the human experts on whose knowledge the system is based, and with the attending physician, using retrospective clinical data. In this case agreement means being of the same opinion in respect of both result interpretation and suggested therapeutic strategy. The attending physician and the human expert were also compared mutually. This approach to validation is typical of the validation procedure for many rule-based systems. The results from the analysis of 119 patients are as summarised in table 2.1

PATRICIA represents no methodological advance over VM (Fagan, 1980). Both systems encode the knowledge in the form of rules and utilise forward chaining or data driven mode of inferencing. The systems are also similar in their partition of knowledge into facts and heuristics. Facts represent the shared knowledge, which is commonly available and agreed upon, whereas heuristics represent the

[11] Persistence of a variable is the length of time following a sample, for which the value of the variable can be trusted to be unchanged. Persistence and Decision Time are part of the vocabulary of the NEXPERT real-time reasoning.

[12] See foot-note 11.

knowledge of the local experts, which are typically judgmental and have remained undocumented until the knowledge elicitation. The heuristic knowledge captured in the knowledge-base is the result of knowledge elicitation from one or more experts. However, the implementation of PATRICIA is based on NEXPERT, which is typical of a new generation of expert system shells[13], supporting Graphical User Interface (GUI) design, and also incorporating facilities for reasoning in time.

Rate of agreement between man/man and man/machine	
human expert and expert system	92%
attending physician and expert system	78%
attending physician and human expert	79%

Table 2.1 Validation Results For PATRICIA

(after Moret-Bonillo *et al.*, 1993)

2.3.5 AIM INFORM

The Advanced Informatics in Medicine (AIM) research programme of concerted actions is a collaborative European effort into application of AI in Medicine. The INFORM project was concerned in particular with applications of decision support in the HDE. The long-term goal of INFORM is to develop, implement and

[13] Shells are tools for building knowledge based systems. They include a knowledge representation scheme, which is typically based on some rule formats, and an inference engine with standard techniques *e.g.* forward chaining and backward chaining.

evaluate a new generation of Information Systems (IS) for hospital HDEs. The exploratory phase of the project was completed in 1990.

Although there was no implementation in the exploratory phase, this was a very important project because it attempted to address some fundamental issues, such as user requirements and integration of decision support systems into routine clinical activities. The work was carried out in two major strands. One was an analysis based on an object oriented extension of the entity-relation methodology leading to a data model (Leaning *et al.*, 1991) and the other a specification of a proposed software architecture (Hunter *et al.*, 1991) based on the Blackboard model of problem solving.

Blackboard Architecture

A blackboard architecture is an AI control technique based on a distributed model of problem solving. There are three main components in a blackboard system, namely the knowledge sources, the control, and the blackboard. Specialised knowledge is organised into a number of Knowledge Sources (KS), sometimes referred to as agents, which co-operate to solve the problem. The blackboard is implemented by a global data structure; and the control may be implemented by a finite state machine. The blackboard architecture is so called because it imitates a group of experts sitting around a blackboard to solve a problem. When an expert sees that a contribution can be made, based on the specialised knowledge, that person raises a hand. The chairman of the group monitors the experts and selects their contributions in order, according to the agenda visible on the blackboard. The contribution might be to confirm or refute a hypothesis on the blackboard or add a new one. Thus the blackboard holds a set of partial or full solutions, known as the solution space. As the experts or knowledge sources add, delete or modify the

blackboard entities, the solution space becomes the current best hypothesis for the problem. The experts communicate with each other only through the blackboard. Extensions of the blackboard architecture have been applied in real-time problems (Nicholls and Shenton, 1992).

2.4 Summary

All projects considered in this review use symbolic rather than numerical techniques. All except SIMON use rules or frames for knowledge representation, and rule-based inferencing for "utilising" the knowledge. Most systems are confined to the Artificial Intelligence (AI) laboratories, with little evaluation or field testing. The reviewed work reveals that intelligent patient monitoring is in its foetal stages, all are development systems, and that almost no clinical systems yet exist. The rare exceptions to this are COMPAS (Sittig *et al.*, 1989) and PONI (Garfinkel *et al.*, 1988). Both of these systems were based on home-made Hospital Information Systems (HIS), so unlike commercially available HIS and Patient Data Management Systems (PDMS), they were flexible enough and "open" enough to allow integration with knowledge-based system. Having reviewed some applications of intelligent patient monitoring in the HDE, the next chapter will review the Multigraph Architecture (MA) and its application in intelligent process control.

CHAPTER 3

MULTIGRAPH ARCHITECTURE & INTELLIGENT PROCESS CONTROL SYSTEM

3.1 Introduction

Having in the previous chapter reviewed the intelligent monitoring in High Dependency Environment (HDE), this chapter is concerned with the Multigraph Architecture (MA) in the context of intelligent process control systems. The purpose of this chapter is twofold. The first is to describe in some detail the Multigraph Architecture (MA), and a prototype framework for Intelligent Process Control Systems (IPCS) based on the MA. The second is to emphasis the model-based approach of software design, and the role of models in intelligent systems. In the section describing the MA, particular attention is paid to comparing this to the traditional approach (*i.e.* rule-based) of implementing knowledge based systems, as well as highlighting the benefits associated with this approach. The section on IPCS mainly describes the major abstractions, and how their associated objects may be used in an application.

3.2 MA Concepts

MA is a model-based architecture for the design of complex real-time knowledge-based systems, (Biegl, 1988; Sztipanovits and Bourne, 1985). It is a **multi**-layered[14] architecture which uses a **graph**-based model of computation. The MA views applications as consisting of the following components in ascending order of abstractions:

- Physical layer

[14] Layer is a level of abstraction.

- System layer
- Module layer
- Knowledge-based layer.

These layers implement the following functionalities:

- The **physical layer** is the hardware on which the system runs. It may be based on a single processor or be multi-processor based,
- The **system layer** comprises the software responsible for resource allocation and scheduling; it is closely related to the hardware,
- The **module layer** implements the algorithmic modules; it supports the graph-based model of computation,
- In the **knowledge-based layer** symbolic and knowledge-based functionalities are realised.

Each layer is implemented in the following way:

- The physical layer may be implemented by any kind of hardware architecture *e.g.* RISC (Reduced Instruction Set Computer) based workstation, transputer array;
- The system layer is implemented by the operating system, usually supplied by the hardware manufacturer;
- The module layer is implemented by source or object code libraries;
- The knowledge-based layer is implemented by very high level declarative languages which are specific to the problem domain. Knowledge about the problem is expressed in declarative languages, usually in terms of models.

Figure 3.1 demonstrates the model-based architecture of MA. The top layer of the system comprises the model builder environment or the **Multigraph Programming Environment** (MPE). The models are created and saved at this level. Model building is supported by high level tools, such as a graphical model builder environment. The models may be heterogeneous structures, *i.e.* include both qualitative and quantitative aspects of knowledge about the system.

The middle layer consists of the model interpreters. The model-based architecture of MA means that the computational structure is derived from the models. The **model interpreter** therefore maps the models into computational structures for execution by the **Multigraph Execution Environment** (MEE).

The bottom layer represents the MEE. The computations in MEE are represented in the form of a bipartite directed graph called the **control graph**. The control graphs are generated through the model interpretation process. The control graphs consist of two kind of nodes, namely **actor nodes** and **data nodes,** and the arcs of the graph represent the data flow among the nodes. The actor nodes represent a computational block and are associated with code segments known as actor **scripts**. Model interpretation combines declarative knowledge, in the form of models, with the algorithmic knowledge, available in terms of object libraries of procedural code, to generate the execution structure. The actor nodes are scheduled for execution by the **multigraph kernel** whenever they are triggered. The multigraph kernel together with the hardware and operating system implement

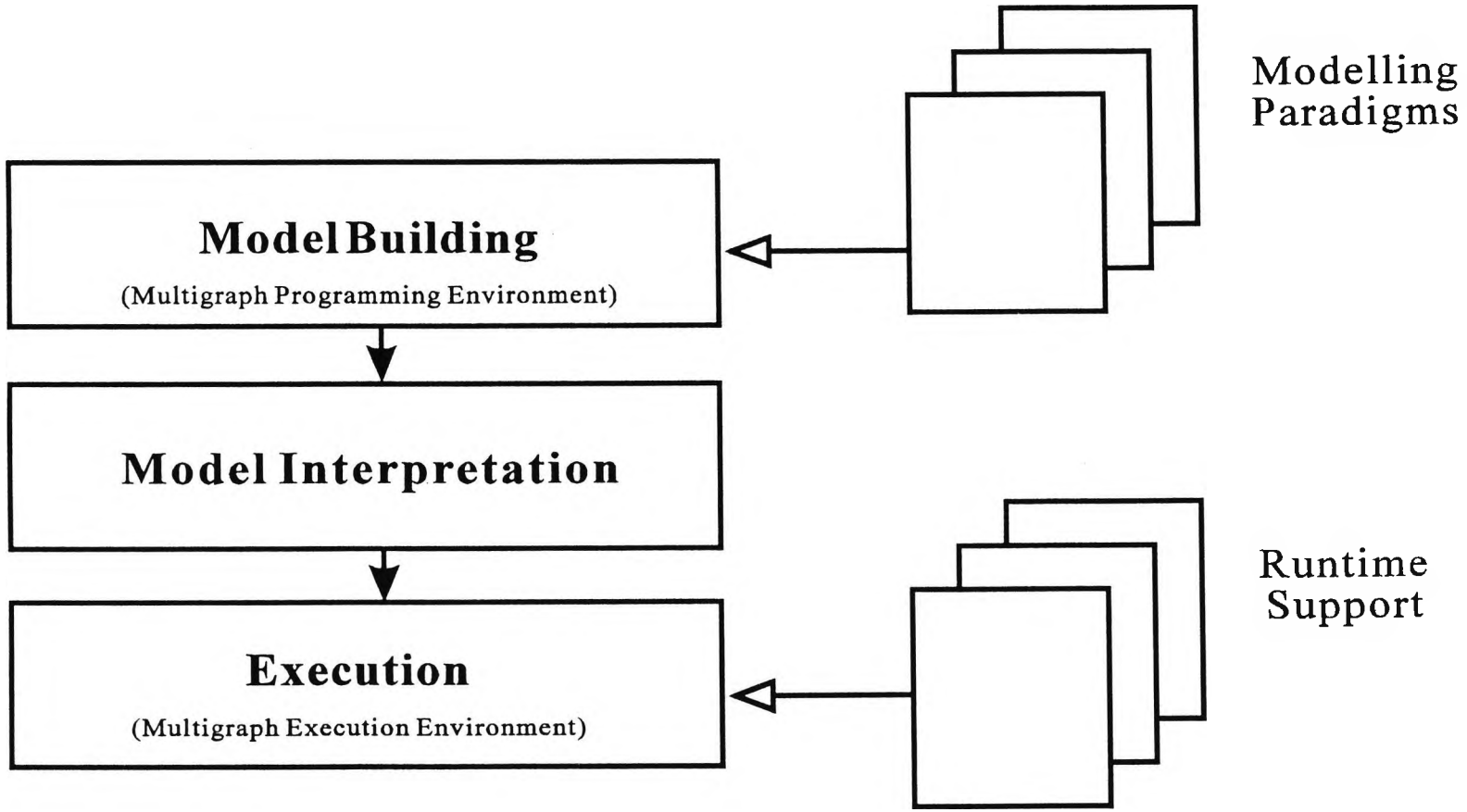

Figure 3.1 The Multigraph Architecture

a virtual **Large Grain Data Flow** (LGDF)[15] machine. The multigraph kernel calls may be grouped into three main categories:

- System interface calls, for interfacing the kernel to the system layer,
- Module interface calls, to interface to algorithmic codes (scripts),
- Knowledge-based interface calls, to interface with the knowledge expressed in the declarative languages.

MA is a generic architecture and hence can be extended to support new modelling paradigms. It is possible to use multiple modelling paradigms within one application to represent different types of knowledge. The extensibility of the architecture also implies that the potential for application is not limited to any one domain. These are key concepts which allow the integration of different modelling paradigms and migration of the architecture to new domains such as patient monitoring.

3.3 Benefits Of The Multigraph Architecture

The advantage of a multi-layered system is that it allows the structural complexities and algorithmic complexities to be addressed at different layers. Hence, there is the possibility of knowledge-based control and dynamic reconfiguration of time-critical low level functions. The usual way to achieve this is to activate low level functions from symbolic operations. The obvious

[15] Grain refers to complexity of code, small grain or *micro* deals with blocks of single or few machine instruction, whereas large grain or *macro* deals with code segments of the order of signal processing subroutines, such as FFT block.

consequence of this is that the speed of program execution will be limited by the symbolic processing. In MA, however, this is overcome by separating the model interpretation and the execution of the control graph. Therefore MA provides an efficient integration of symbolic and numerical processing.

Another advantage of using the MA is that it is portable. The multigraph kernel implements a virtual data flow machine and all the computation is expressed in terms of data flow graphs. Hence, provided the kernel is portable to different hardware platforms, the system is also portable.

A further advantage is derived from using the control graphs as the computational model. Any number of actors on parallel paths on the graph may be executed simultaneously provided the resources, (*i.e.* processing power and the necessary memory), exist. This can be exploited to create parallel and distributed execution by simultaneously executing segments of the control graph on arrays of processors or distributed workstations.

The MA affords a transparent testing process. In knowledge-based systems knowledge is often expressed as rules or as frames of rules. The validation of such systems is a classic problem. It is impossible to test all possible combinations of all rules under all scenarios (O'Keefe *et al.*, 1987). Yet this unknown combination of rules is the reason for using them. In MA the knowledge is organised in terms of models, and validation takes place by ensuring correctness and consistency in the models.

Finally, the MA allows for simpler consistency checking. To ensure consistency it is sufficient to check the models, rather than the entire execution code.

3.4 Real-Time Performance

Another reason for the use of MA is its real-time performance capability. Before discussing the MA and real-time there is the need to define our understanding of the term "real-time", since it has been used with different meanings in the literature (O'Rielly and Cromarty, 1985). A common understanding of real-time is **fast** or fast enough processing compared with the available data rate. Often real-time is used to mean on-line which may also imply **reactivity** *i.e.* a system whose behaviour is dependent on sensed events. A more strict definition is a system in which the correctness of results depend not only upon the logical correctness of computations but also upon the time at which the results are produced. Therefore if the timing constraints are not met the system is said to have failed. The terminology adopted here will be based on the division by Stankovic (Stankovic and Ramamritham, 1988) into **hard** and **soft** systems. The term hard real-time is used for systems in which the timing constraint is guaranteed to be met, and soft real-time is used when a reduced timing constraint applies, *i.e.* in cases where there is some value in tasks which are completed after their deadline.

3.4.1 Real-Time Performance And Symbolic Processing

Traditionally AI and expert systems are associated with symbolic processing. The most widely used language for symbolic processing is LISP. LISP is a dynamic

language. This means that the data structures are created during the run-time in contrast to block structured languages such as PASCAL or C. In other words the program consumes memory, hence the need for memory management operations to free memory, these being called **garbage collection**. The garbage collection associated with symbolic processing creates other problems besides speed of execution. Firstly the continuity of operation is violated. Secondly, because garbage collection can come into action at unpredictable times and for unpredictable lengths of time, predictability cannot be guaranteed.

The review of real-time expert systems by Laffey *et al.* (1988) contains a comprehensive list of problems with traditional AI shells[16] and why they can not be used in real-time applications as summarised below. Expert system shells:

- are not fast enough,
- do not facilitate reasoning about time evolution of data,
- are difficult to integrate with other software and Input/Output (I/O) systems,
- do not have a mechanism for focusing attention,
- have no facility to handle asynchronous events,
- cannot efficiently acquire data from external sources,
- cannot provide reliable response time, as most search algorithms employed are of **exponential** complexity.

[16] Shells are tools for building knowledge based systems. They include a knowledge representation scheme, which is typically based on some rule formats, and an inference engine with standard techniques *e.g.* forward chaining and backward chaining.

- cannot run continuously,
- have no mechanisms to deal with change in work load or availability of resources.

For these reasons conventional AI systems based on expert system shells cannot be used for real-time applications.

3.4.2 Review Of Real-Time KBS

Since 1986, however, there has been a steady introduction of a range of commercial tools for development of real-time knowledge based systems. Some examples of such tools include G2, COGSYS, PROMASS and RTWorks. All such systems use rule-based inference as their reasoning mechanism. C is invariably used as the implementation language, in order to avoid the problems associated with symbolic processing languages, such as speed and continuity of operation as described above. Therefore, the real-time performance problem has been solved by sacrificing complexity of knowledge representation and using a restricted rule format and simple inference method. Also the timing constraint is met by using priorities, in task scheduling. Commonly such systems provide **time-stamps** and **histories** to facilitate **temporal reasoning**.

3.4.3 The Real-Time Aspect Of MA

MA has been applied in soft real-time applications, *e.g.* Intelligent Process Control System (IPCS). MA can be applied to real-time problems because it allows for separation of the model interpretation and execution. The run-time system can be

entirely created from modules written in C language and hence the speed of execution is comparable with conventional systems. The simple method of parallelisation of the execution structures described above is another reason why MA is suitable for real-time applications, *i.e.* meeting real-time demands by using parallel hardware. Using hierarchical models it is possible for some tasks such as diagnosis to be performed in an incremental manner, for instance by starting at the root and progressively increasing the resolution of the diagnosis; in other words, using the best intermediate result as a way of meeting timing constraints. Finally in MA applications in monitoring such as IPCS, diagnostic algorithms of **polynomial** complexity[17] are used, therefore the timing constraint are met by using predictable and bounded response time.

In order to implement hard real-time capabilities in the MA it is necessary to augment the computational graphs with timing constraints. The computational graphs can then be scheduled for execution in such a way to meet the timing constraints. This has been the subject of research by Waknis (1993).

The disadvantage of using MA is the overhead associated with developing the declarative languages and the corresponding model interpreters.

Using a macro data flow graph as the computational model has a potential disadvantage. There is an overhead associated with the scheduler, *i.e.* the processing necessary between invoking the scripts of two successively fired actor

[17] This is the standard way of comparing algorithm efficiency in which time taken to solve a problem is specified as a function of N, where N in some way represents the size of the problem. Exponential algorithms are generally to be avoided because they can only be used with problems of very restricted size *i.e.* they are not **scaleable**.

nodes. In order for this overhead to be negligible the size of the script has to be at least an order of magnitude larger than the size of the scheduler overhead. This implies that for the kernel to be efficient the computation must be capable of being represented as a data flow graph with medium sized (of the order of 1000 machine instructions) actor scripts.

3.5 MA Applications

MA has been applied to build parallel distributed systems. Abbott and colleagues describe an image processing demonstration system (Abbott *et al.*, 1990) and a real-time digital signal processing system (Abbott *et al.*, 1993) for the analysis of turbine engine test data. The image processing system was implemented on a network of four INMOS transputers running the transputer implementation of the Multigraph Kernel (MGK), giving very high processing speed up factors (3.95 at best, out of a theoretical maximum of 4). The turbine engine monitoring system runs on a heterogeneous network of around 100 processors, capable of providing around 0.2 Gflops. Neither application required any parallelisation of code, except at the control graph level.

Because the process of model interpretation can be restarted at run-time, this has been used to create structurally adaptive[18] systems, that reconfigure themselves during runtime (Blokland and Sztipanovits, 1988; Sztipanovits *et al.*, 1993; Sztipanovits and Wilkes, 1988; Wilkes *et al.*, 1990). The computational structure not just its parameters may be changed, secondary to changes in the environment, through

[18]In **structurally adaptive** systems the computation structure is varied dynamically, *cf.* **parametric adaptivity** where the structure is fixed but only parameter values may be varied.

manipulation of the control graph. An application in sonar signal processing has been described by Misra *et al.* (1990).

MA has been used to develop plant-wide process monitoring control and diagnostic (MCD) systems (Karsai *et al.*, 1992; Karsai *et al.*, 1990; Padalkar *et al.*, 1991; Sztipanovits *et al.*, 1990; Sztipanovits *et al.*, 1987). The MCD applications include the monitoring of a cogenerator plant and are based on the Intelligent Process Control System (IPCS) framework, detailed in the next section.

3.6 Intelligent Process Control System (IPCS)

As shown in the previous section a software architecture is a partial specification of an application. In other words it specifies the form, but not the details, of an application program. A framework on the other hand is based on an architecture but is augmented with a set of development tools as well as a set of pre-defined objects or object classes that may be useful in a group of applications. IPCS is a framework for intelligent monitoring.

In this section IPCS (Karsai *et al.*, 1992; Padalkar *et al.*, 1991) is described as an example of a general purpose intelligent Monitoring, Control and Diagnostic (MCD) system, implemented using the MA. The term intelligent refers to the fact that the run-time behaviour is derived from the knowledge embedded in the system. The term model-based indicates that the knowledge is organised in terms of models.

IPCS is an integrated set of high level software tools for design and implementation of model-based intelligent process control system. IPCS consists of two main components:

- a **development shell** which is a graphical environment for model building,
- a **target shell** which is the run-time environment.

Models in IPCS are graphical and syntactical structures which represent expert knowledge. The model builder environment is a graphical editor. The user creates structures by placing and connecting icons and editing their properties. Declarative language statements are generated from graphical models. Once the models are created they are loaded in the run-time environment to build and execute the run-time system.

The IPCS run-time functionalities include:

- Data collection and processing,
- Fault detection and real-time diagnosis,
- Simulation and control,
- Operator interface for input and output.

The modelling philosophy is specifically designed to support engineering problems. The most important concepts are hierarchical decomposition and multiple aspect modelling. Together these concepts provide an effective approach for dealing with the complexity of monitoring large process plants.

3.6.1 Model-builder Editors

There are five declarative languages defined for the IPCS. Each one has its associated graphical editor.

Processor Editor

Here the signal processing blocks are created using the Hierarchical Description Language (HDL). The function of the HDL editor is to capture the digital signal processing functions of the application in a hierarchical manner, using user defined or standard libraries of signal processing blocks.

Panel Editor

The panel editor is used to define the operator interface panels. The graphical user interface and all user inputs through mouse and keyboard and system outputs to the screen are defined here.

Physical Component Editor

The physical component editor is used for building a hierarchical representation of the plant in terms of physical components. Each component can have its associated failure states.

Process Model Editor

This is a multiple aspect modelling editor. The process is modelled from six view points which are as follows:

- **Structural view,** which shows how the process model is decomposed into simpler models, the related process variables, the process states and parameters;
- **State transition view,** to define the state transitions and link them with events that trigger the state transitions;
- **Monitoring and control view,** which represents signal processing and control functionalities including interface to the external world and data buffers;
- **Simulation view,** which describes simulation blocks providing simulated data for the monitoring and control;
- **Failure propagation view,** which represents how the failures of component processes interact and how they are related to alarms;
- **Operator interface view,** which shows the operator interface panel connections to the monitoring objects.

PML-Physical Editor

This editor is a double hierarchy editor used to specify the failure mode/failure state associations. For example, a leak which is a flow process failure mode is associated with a pipe-broken which is a physical component failure state. The plants functional hierarchy and physical component hierarchy are thus linked through failure mode/failure state links.

The collection of the graphical model editors as described above allow all aspects of the system to be designed by **visual programming**. This means that an application can be developed by placing and connecting icons of the relevant objects.

3.7 Summary

In this chapter concepts of the Multigraph Architecture (MA) have been described, together with the benefits associated with using this architecture. The Intelligent Process Control System (IPCS) has been used to indicate how MA is applied to real problems, and also to describe the abstractions and their associated software objects that may be used in design of process control applications. In later chapters it will be shown how these may be used for patient monitoring and also their limitations and what extensions are necessary for this domain. The next chapter describes the generic specification of a model-based framework for patient monitoring.

CHAPTER 4

GENERIC SPECIFICATION OF A MODEL-BASED FRAMEWORK FOR PATIENT MONITORING

4.1 Introduction

The earlier chapters have described how a model-based framework can help in design of intelligent real-time software. Therefore the next step is to design a model-based framework. This chapter aims to highlight some of the general requirements of a framework for patient monitoring in High Dependency Environment (HDE), through a generic design. The purpose of the generic design is to identify some common software requirements which will serve as a guideline for the following stages. The generic design is independent of the implementation but will indirectly dictate the choice of the programming environment and style. The generic design is intended to serve as the basic ground work that may be used by any designer wishing to implement a software framework for patient monitoring. Having a design or specification for the framework, an incremental approach will then be taken in the corresponding implementation.

4.2 General Requirements

This section contains a description of some general software requirements, such as instrumentation interface, data archiving, graphical user interface and data visualisation; as well as some desirable attributes of the software architecture, such as visual programming, generic, model-based, distributed, open and portable architecture, seamless integration and interoperability of different components.

4.2.1 Interface

To allow for automatic data acquisition from the HDE instrumentation, the framework should contain generic interface modules that may be reused with minimum requirements for writing new code. For modularity the main task should be separated from the data acquisition task, so that the changes in the data interface will not affect the main software components in any way. This means that an application can be developed without specific knowledge of the bedside instrumentation. Thus, the investments in application development will be protected from changes in bedside instrumentation and their interfaces. A solution would be to have a separate process responsible for generating data and for this data server process to communicate with the main application through inter-process communication protocol, using for instance queues, named pipes and sockets. The data server may generate data by interfacing with the instruments through dedicated hardware or a machine with a fast data bus, or the data may be generated by interfacing with other applications, for example to the patient chart database through a Structured Query Language (SQL)[19] interface. For test and simulation runs the data interface will simply read from data files.

4.2.2 Archiving Of Data

It is necessary for a framework for real-time monitoring applications to have facilities for archiving of captured data, to facilitate the recording of the real-time data from the monitors. This should also include the ability to play back recorded data, in order to give access to past data. The play back of archived data will be

[19] Structured Query Language (SQL) is the official standard language for interfacing with relational type database systems.

invaluable during testing and debugging of the other components. Also if the system needs to reason in time, having access to past data becomes necessary.

4.2.3 Data Visualisation Modules

Integrated and intelligent instrumentation in HDEs is required to present a large amount of information, which may include several channels of sensed data as well as the result of the interpretation of the data, to the user. One of the functions of the intelligent instrument in HDE is to aid the clinician with the assimilation and interpretation of data. Therefore the visualisation of data for effective and efficient flow of information to the user becomes a crucial issue. Hence the framework should contain software modules, such as graph widgets (for various types of two and three dimensional plots) and display bars, as easy to use building blocks for an efficient implementation of Graphical User Interface (GUI). In order to facilitate the visualisation of captured data the dynamic graph widgets must link to real-time data objects.

4.2.4 Visual Programming Environment

In a visual programming environment, the application developer creates new applications by placing and connecting together icons, these icons representing code segments or modules. Supporting the framework with a visual programming environment will make it accessible to users with a lower level of ability in the technology. This is specially important because the application developers can then afford to spend less time understanding the technology and more time on understanding the domain and the user requirements. Visual programming can also

help in eliminating the need to write code for every step, which is slow and repetitive.

4.2.5 Open, Distributed, And Portable Architecture

As far as possible the framework should be based on open technology and comply with recognised communication standards. The term "open" implies that the hardware/software technology should be available from more than one vendor, that it should be extendible by third party vendors and users, and that the users and third party vendors should be provided with access to internals of the software through a well defined and supported Application Programmers Interface (API). Example of open hardware architecture is IBM's PC architecture while UNIX and X-windows are commonly considered as open software. Complying with communication standards should, at least in theory, eliminate the need for user programming when interfacing with instruments from different vendors.

The framework should be distributed, so that a monitoring environment may be spread over a network. This flexibility to distribute the application over a number of independent processors can be used to maximise the system response. For instance, if a task is computationally expensive requiring faster more expensive hardware, the task can be run on such a machine remotely. The bedside workstation can be of lower specification with one faster workstation designated as a central resource for processing power which may be shared between a number of beds.

The framework should be easily portable across hardware platforms or operating environments, otherwise applications developed using the framework will be tied to specific hardware.

4.2.6 Model-Based

Modelling is central to monitoring. All measurements and monitoring are based on models. The role of models in intelligent monitoring has four aspects. First the models are used as a coherent, structured and explicit representation of the knowledge. In this respect models help deal with complexity. Second, the models are used in software synthesis (Abbott *et al.*, 1993). In model based software synthesis, system behaviour is specified through the models. So making new applications is largely a matter of building the models rather than coding. Third, the models are used in the reasoning and interpretation. Fourth, the adaptive system behaviour and reconfiguration requires self modelling, (Sztipanovits, 1989) *i.e.* model of the monitoring system itself, its current and possible configurations and the available resources. For the above reasons it is desirable that the framework for intelligent patient monitoring be model-based.

4.2.7 Generic

The underlying architecture must be generic. In other words it must not be closely linked with any single modelling or reasoning paradigm that will exclude the use of other reasoning techniques. On the contrary it must be flexible enough to allow integration of different modelling paradigms; so that the framework can be

extended to incorporate multiple aspect modelling and new modes of reasoning and modelling paradigms.

4.2.8 Integration/Interoperability

The framework should allow for integration of different aspect of monitoring, *i.e.* data acquisition, processing, display and user input. It should make it possible for interoperation between different types of modules such as AI modules or database modules. The framework should allow easy integration of user code with the rest of the application.

4.3 Summary

The aim of this chapter was to identify some general software requirements for a framework for intelligent patient monitoring. This was achieved through a generic design. Eight features have been identified in the generic design. These included facilities for design of instrument interface, data archiving, data visualisation, integration and interoperation, visual programming as well as architectural features of generic, model-based, open, distributed and portable. A framework that fulfils all the above requirements is Intelligent Process Control System (IPCS). In the following chapter it will be shown how this framework was adopted for patient monitoring.

CHAPTER 5

CUSTOMISATION AND EXTENSIONS TO IPCS FOR PATIENT MONITORING

5.1 Introduction

While the last chapter dealt with some common requirements for an Integrated Model-based Development Environment (IMDE) to form a generic specification of such systems, this chapter deals with the work done in developing a prototype framework for an Intelligent Patient Monitoring System (IPMS). The prototype IPMS was implemented by adopting the Intelligent Process Control System (IPCS) framework and adding extensions or modifications as and where necessary. These tasks were carried out by identifying the concepts that are relevant to patient monitoring but that were missing from the IPCS, and by their subsequent implementation. Therefore, the aim of this chapter is to report on the implementation of some candidate exemplars using Multigraph as the underlying software architecture. This includes implementation of three software libraries: one each for signal processing; a fuzzy controller; and probabilistic modelling.[20] The following three sections describe the work done in each area.

5.2 Signal Processing

It is often helpful to visualise the process of the transformation of data into information as taking place at different layers. A graphical representation of the layered view is shown in figure 5.1. In this abstraction, the processes at the bottom layer will correspond to acquisition of raw data, and the top level will correspond to processing of concepts with higher level of information, such as identification of

[20] See sections F.1-F.5 in Appendix F for code listings.

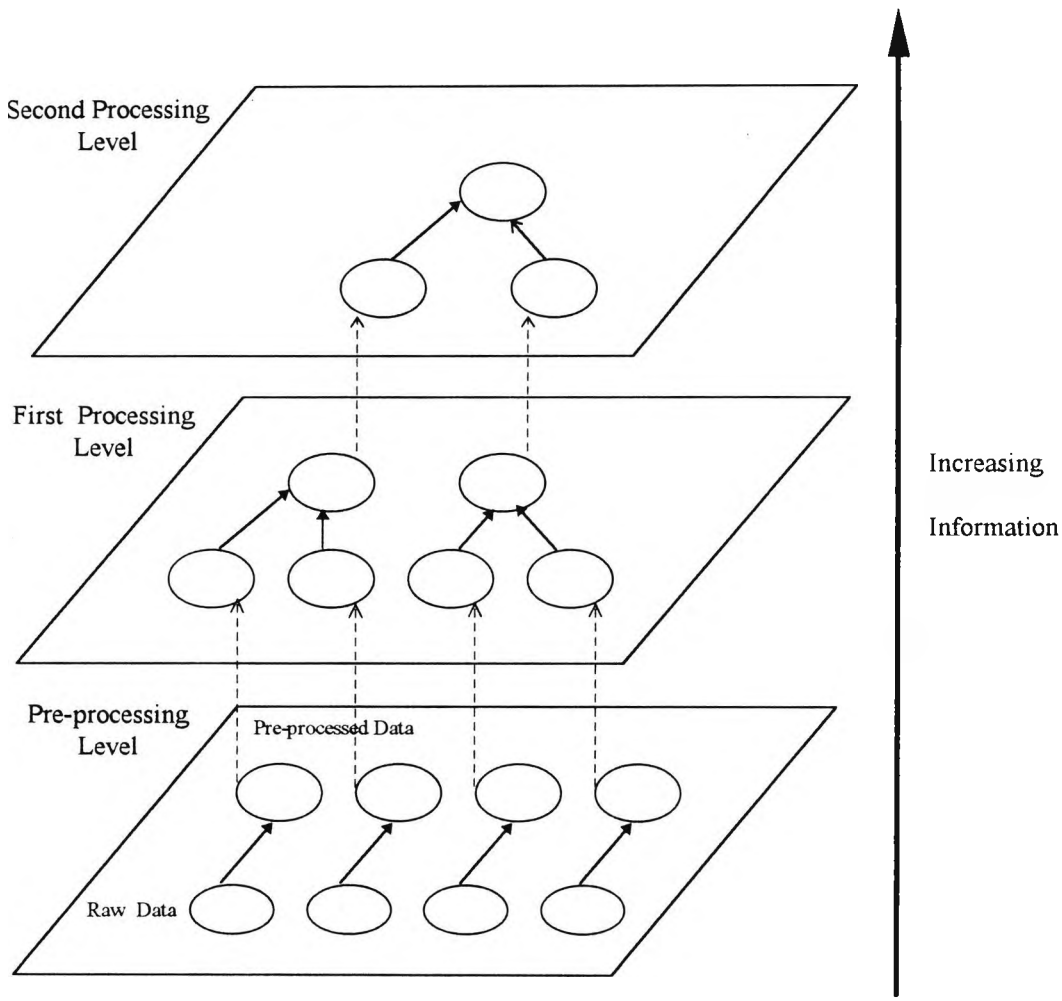

Figure 5.1 The Layered View Of Data Transformation

patient state (Factor *et al.*, 1990). It is usually necessary for some kind of pre-processing of the raw data, generated by the instrument, to take place before higher levels of signal interpretation. This may be necessary because of contamination of signal by noise, or simply to extract the desired signal component. This is represented in the layered view as the first layer (*i.e.* pre-processing). Therefore the implementation of this layer includes algorithms of different digital signal processing techniques such as filtering and power spectrum estimation. The Hierarchical Description Language (HDL) (Karsai, 1988) allows for a hierarchical description of the signal processing as networks of processing blocks.

To illustrate this process, two types of filtering that find application in patient monitoring have been implemented. The first is a generic time domain digital filter that can be used for all types of filtering *i.e.* low pass, high pass, band pass and band stop filtering. For real-time applications it is often necessary to process a continuous stream of data at the rate at which those data are received. Under these circumstances it is necessary to filter in the ***time domain***. The software implementation of the generic filter uses a cascade realisation of blocks of second-order Infinite Impulse Response (IIR), these blocks being implemented in Direct Form type II (IIRDF2), see Appendix A. Software implementation of filter design techniques such as the bilinear transformation method may be used to obtain the filter parameters by specifying the desired frequency response (Taylor, 1994).

The other type of filter implemented is a generic running median filter. Median filtering is a discrete time operation, and hence suitable for real-time applications, where time domain filtering is desirable. They have the desired characteristic of reducing noise and transients without removing the signal's sharp edges. The

output of a median filter on a window length of 2k+1 is the median of the values inside the window. The output is delayed by k number of samples. Median filters are characterised as non-linear systems with memory and hence cannot be analysed by linear methods (Nodes and Gallagher Jr., 1984). Instead the concept of root signals may be used (Gallagher Jr., 1981). A root signal is what remains after repeated filtering of a signal. The filter passband is determined by the root signals. Median filters remove any variability shorter than the filter delay. Therefore they are used for extracting variability in a desired time band. Applications in intelligent alarms include signal smoothing for reducing false alarms or trend detection for creating trend alarms (Mäkivirta *et al.*, 1989).

Because the implementation of the filters is generic they include a set of parameters whose instantiated values determine the exact filter behaviour. This means that they are fully reusable.

5.3 Fuzzy Paradigm

Fuzzy, or multivalued, set theory was first developed by Zadeh (1965). Mathematical fuzziness corresponds to different degrees of ambiguity or indeterminacy. Although originally applied as a mathematical framework for representing "grey" relationships in a knowledge base (Zadeh, 1983), the focus of (commercial) applications soon shifted towards control. Essentially a fuzzy system transforms or maps inputs to outputs. The structure of the fuzzy system defines this mapping. In practice, the input **data** (*i.e.* exact values, not a fuzzy set) are mapped to control or classification **data**. Hence, most implementations of fuzzy control are similar to a forward chaining rule-based inference system, where the

rules use a symbolic representation of the variables based upon a fuzzy membership function, and the output set is defuzzified to give a single value output.

By associating fuzzy subsets of real line to fuzzy variables, figure 5.2, a fuzzy rule is enabled to embody a rich structure (Kosko, 1992a). Kosko (1992b) proposes to call fuzzy rule-based systems **principle-based systems**, for this reason. Also, **all** rules fire to different degrees. Consequently far fewer rules are required to represent the same behaviour. This has meant that fuzzy control has been successful where traditional expert system control approaches have failed.

The use of classical control techniques is best suited to applications where the system dynamics may be modelled mathematically. Physiological systems, however, are difficult to model exactly and contain non-linearity. This implies that the design of controllers for the physiological domain must be "model-free[21]" and accommodate non-linearity. Fuzzy controllers embody both of these properties. Thus application of fuzzy techniques in closed loop control of physiological variables is appropriate. Ying *et al.* (1988) developed the first fuzzy controller for Mean Arterial blood Pressure (MAP). This was tested on computer simulations. This controller was later modified and applied to the control of MAP in pigs (Ying and Sheppard, 1989). In 1992 Ying *et al.* reported the first successful **clinical** application of fuzzy logic in control of MAP for post-surgical cardiac patients. This application will be described in some detail in the next paragraph. Other significant work in this area includes closed loop control in delivery of anaesthetic

[21] Without specifying mathematically how the outputs of the system to be controlled depend on inputs.

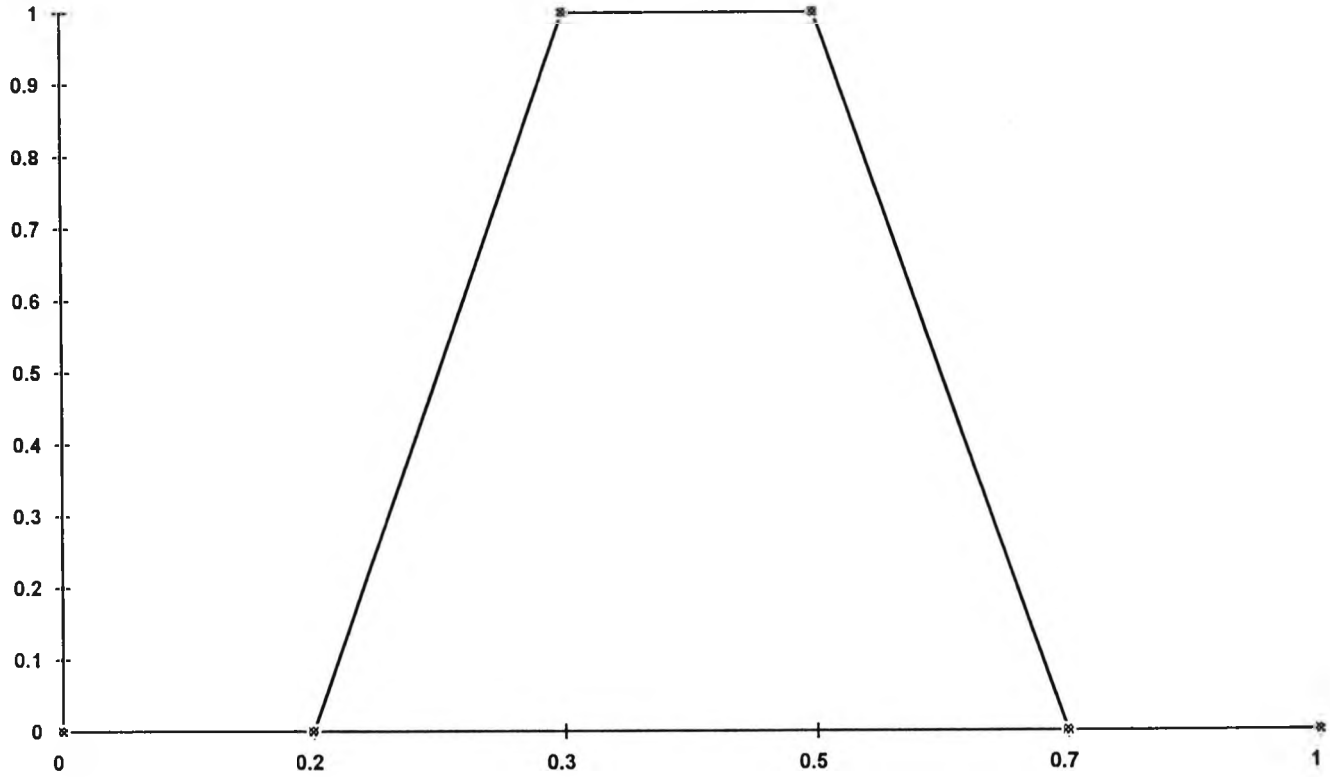

Figure. 5.2 A Four Point Trapezoid Fuzzy Membership Function

gas (Meier *et al.*, 1992) or muscle relaxant (Linkens and Mahfouf, 1988; Linkens and Hasnain, 1991). A review of closed loop control of MAP can be found in Isaka and Sebald (1993).

The fuzzy controller of Ying *et al.* (1992), shown in figure 5.3, is applied to the control of MAP in post-surgical cardiac patient. Post-surgical patients in intensive care can develop hypertension, which if not treated can cause further complications. The controller works by adjusting intravenous infusion of sodium nitroprusside (SNP), a potent fast acting vasodilator. SNP works by relaxing the muscles of the peripheral vasculature thus lowering the blood pressure. The MAP values were fed from a Hewlett-Packard 78534 monitor to an IBM PS/2 model 70 Personal Computer (PC). The controller running on this PC was implemented in the form of 10 control algorithms programmed in C. The controller outputs were sent to a digital infusion pump, manufactured by Abbott/Shaw LifeCare, which infused the SNP intravenously to the patients.

The clinical trial took place in the Cardiac surgical Intensive Care Unit (CICU) of the Carraway Methodist Medical Center, Birmingham, Alabama, with twelve post-operative Coronary Artery Bypass Graft (CABG) patients who had elevated MAP. The fuzzy controller was on trial on the 12 patients for a total of 95 hr and 13 min; the length of time individual patient's MAP was controlled by the fuzzy controller ranged from 1 hr 45 min to 18 hr 7 min. The MAP was sampled every 10 seconds. The trial showed that the MAP stayed within 90% to 110% of the desired value (MAP_d) on average for 89.3% of the time with a standard deviation of 4.96.

Although the IPCS provides implementation of control functions, this is limited to simple linear control. Therefore it is necessary to include a generic fuzzy controller

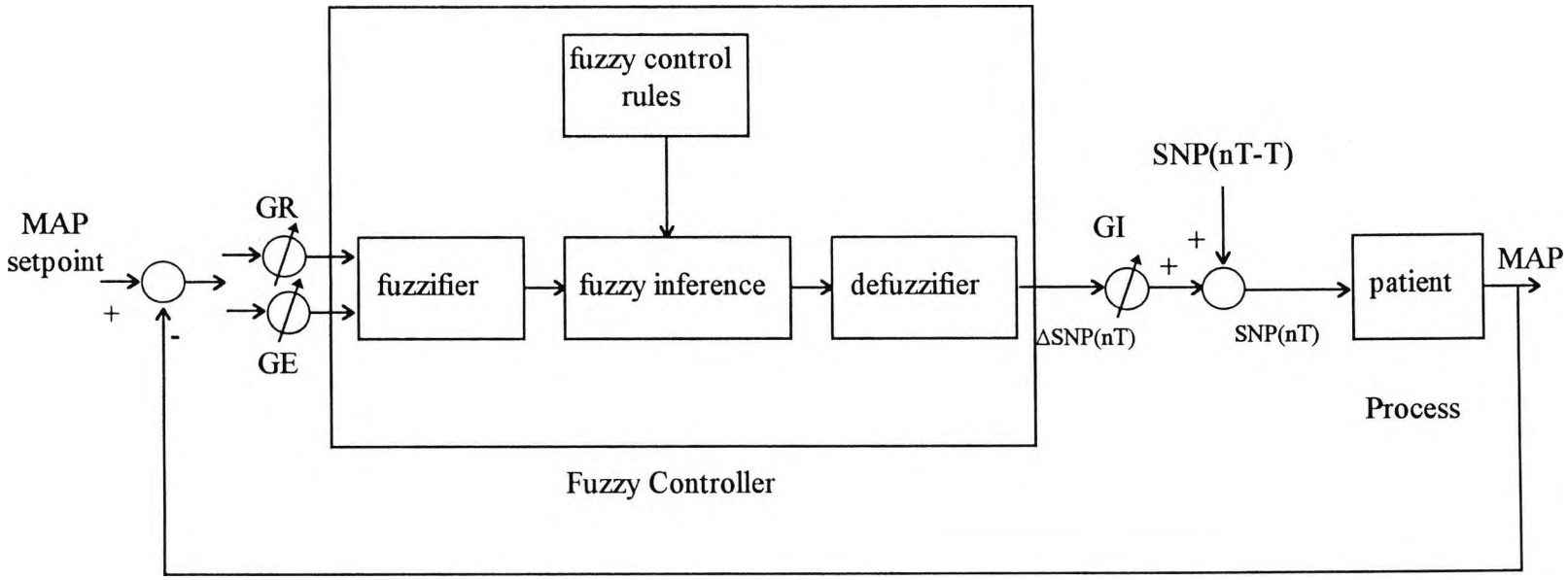

Figure 5.3 Block Diagram Of Fuzzy MAP Controller (after Ying and Sheppard, 1994)

in the framework for patient monitoring. The fuzzy controller is a general purpose fuzzy logic inference engine. Any number of rules with any number of antecedents and consequences, with variables of any number of four point (trapezoid) fuzzy membership functions can be used. **Min**, **Max** and **Centroid** operators are used for inference. All variables have a 0-255 range, so that integer arithmetics can be used instead of floating point. The use of integer arithmetics together with the application of some known optimisations of the fuzzy computations, make this implementation of the fuzzy logic library very efficient, and suitable for real-time control applications. The implementation of the rule-base is based on linked list structures (Viot, 1993). There is a list of inputs, a list of outputs, a list of rules. Each input or output has a list of membership functions. Each rule has an if_side list (antecedents) and a then_side list (consequences).

The fuzzy logic library can be used to implement a MAP controller, similar to the one implemented by Ying *et al.* (1992) as described, without the need for any programming. All that the designer has to do is to specify the variables and the rules in a simple format which is detailed in Appendix B.

While the fuzzy logic library has been designed to be general purpose the application is mostly in fuzzy control. This is because more sophisticated tasks such as diagnosis, where it is possible for intermediate facts to be inferred, *i.e.* output of some rules may become input to other rules, require some mechanisms to deal with ***truth maintenance*** and ***conflict resolution***, (these are concerned with the selection and order of the rules to be evaluated and the consequences of these on the integrity of the knowledge base). The current implementation of the fuzzy logic library does not incorporate such mechanisms, and therefore its application is

limited to fuzzy control, where typically the rule antecedents and consequences form mutually exclusive sets.

5.4 Probabilistic Modelling Paradigm

5.4.1 Uncertainty And Reasoning

The need for representing uncertainty when building intelligent medical applications has been recognised, and has formed the focus of much research into probabilistic and other approaches, ever since the implementation of **Certainty Factors** (CF) in MYCIN (Shortliffe, 1976).

MYCIN, developed by Shortliffe, is an expert system for the diagnosis and treatment of meningitis and bacteraemia. Shortliffe and Buchanan (1975) described the difficulties with using a simple Bayes model in a rule-based system, including assessment of conditional and *a priori* probabilities, cognitive complexity, and conditional independence assumptions of Bayes, and concluded that there was a need for a modular approach to handling uncertainty, which need led to the creation of their CF model. In this model CFs are combined using ***parallel*** or ***series*** combination functions. The parallel combination function was later revised to be more consistent with common sense (Van Melle, 1981). Heckerman (1986) later reformulated the CF model giving it an interpretation consistent with probability theory. In this interpretation the CFs may be understood as measures of change in belief. In later reviews Shortliffe (Heckerman and Shortliffe, 1992) admits that the assumptions of the CF model are stronger than the conditional

independence assumptions of Bayes, the subjective assessment of CFs from experts are still as difficult, that the modular approach of CFs is inappropriate since uncertain reasoning is inherently much less modular than certain reasoning, and finally that medical knowledge is usually predictive (from disease to symptom) not diagnostic (from findings to disease).

The Causal Probabilistic Network (CPN), also known as a Bayesian belief network, offers a numerical framework for dealing with uncertainty in a systematic and consistent manner. Equations representing Bayes' conditional probability form the foundations for the CPN algorithms. CPNs provide a modelling (knowledge representation) formalism, where the concepts are represented by nodes of a directed graph and the (causal) relation between concepts is represented by the arcs of the graph. Associated with each node is a discrete probability distribution and with each arc a conditional probability table. CPNs avoid problems of CFs by allowing the representation of conditional independence through lack of arcs in a network, also by not requiring indirect dependencies to be assessed by the user, and in general providing a more modular approach in knowledge representation.

Since, the IPCS framework does not support a formalism for handling uncertainty, it was decided that a prototype Intelligent Patient Monitoring System (IPMS) should extend the modelling paradigms of IPCS to include uncertain reasoning in the form of a CPN representation.

CPN Computations

Given any belief network it is possible to compute any set of probabilities from the joint probability distribution function represented implicitly by that network. Several algorithms have been developed that implement the computations within a belief network. These include the node elimination method by Shachter (1986), for influence diagrams (Howard and Matheson, 1981), the message passing scheme of Kim and Pearl (1983) for singly connected trees, Pearl's conditioning method (Pearl, 1986) and stochastic simulation (Pearl, 1987), and finally Lauritzen and Spiegelhalter's clustering method (Spiegelhalter 1988; Lauritzen and Spiegelhalter, 1988). Essentially the Lauritzen and Spiegelhalter algorithm works by transforming the network into a tree by forming cluster of nodes called ***cliques***. The cliques are formed in such a way that each clique is only connected to its parent nodes and children. This allows for propagation of evidence to take place only through local computations.

The complexity of evidence propagation computations in a belief network is NP-hard[22]. The Lauritzen and Spiegelhalter propagation algorithm is exponential in the largest clique size, and linear in the number of cliques in the network. The size of the largest clique in a network depends on its connectivity.

[22] NP-hard means that it is unlikely that an algorithm can be developed to solve the problem in polynomial time.

5.4.2 The CPN Graphical Model Editor

In the IPMS the graphical model editor includes a separate modelling language and graphical representation dedicated to CPN. The IPCS model builder consists of the generic graphical model editor plus a language definition file which is loaded at the start of a session. The model editor is therefore adapted to each modelling language by writing an Editor Definition File, (*.edf), which file specifies the allowed objects, their attributes and their relations (Karsai, 1990). The CPN model editor likewise is really a customisation of the generic graphical model builder. The CPN model editor supports hierarchical model building as do other modelling languages of IPMS. Therefore a CPN model can contain primitive objects as well as compound objects, *i.e.* networks of CPNs, as parts. The CPN primitive objects, ***nodes*** and ***relations***, correspond to nodes and arcs of a CPN graph. The states and conditional probability tables are implemented as attributes of ***node*** and ***relation*** objects respectively.

The current implementation of the CPN model editor contains definitions for primitive objects of type node (CPN-NODE), three types of relation objects (CPN-ADD, CPN-MULT and CPN-NORM), corresponding to addition, multiplication and sampled normal relation, plus one type of compound object (CLUSTER-COMPONENT) which is used to represent a collection of related nodes as a part. The node object has four attribute slots, for number of states, list of state names, list of state values, the distribution table. The addition, multiplication and normal relations each have a slot for addition factor, multiplication factor and a distribution table correspondingly. All objects have an additional slot for comments.

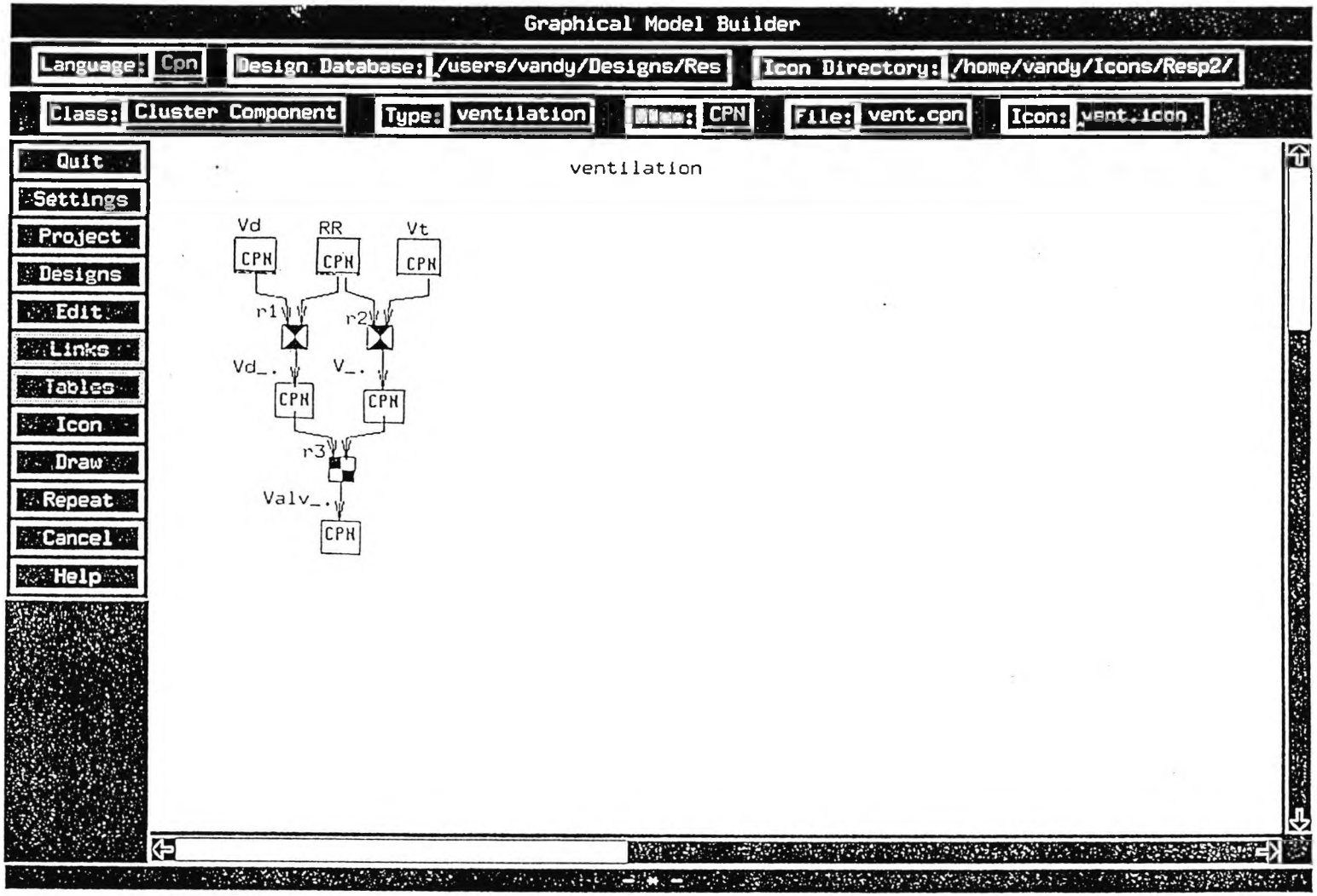

Figure 5.4 The CPN Graphical Model Editor, Containing The Ventilation Model Segment

Figure 5.4 shows the CPN model editor and the network representing the relation between alveolar and total ventilation, which forms a small part of the total respiratory model. Appendix C contains the textual model database file which is automatically generated by the model editor from this graphical model.

5.4.3 The CPN Runtime

To fully integrate the CPN into IPMS, it is necessary to provide a model interpreter for converting CPN models into internal representations for use by the runtime, as well as implementing the CPN runtime for the Multigraph kernel. This is a huge task. It was decided that for this prototype implementation of IPMS it would be adequate to use the CPN library of xhugin[23] as the run-time support (Andersen *et al.*, 1989) and map the entire CPN network into a single signal processing block as an HDL[24] structure. This HDL structure will have all the observable nodes as input and the relevant nodes as output. The xhugin library implements a modified (Jensen *et al.*, 1990) version of the belief updating by network propagation algorithm of Spiegelhalter and Lauritzen. The model interpreter therefore, is a LISP program which converts the CPN specification generated by the model editor into a format suitable for use by the xhugin library. The xhugin library must be loaded as an object module into the Multigraph kernel. The three components, *i.e.* the model editor, the model interpreter and the runtime support, together implement the CPN modelling paradigm.

[23] xhugin is a development environment for CPNs on X-windows.

[24] Hierarchical Description Language (Karsai, 1988)

The implementation of the CPN modelling paradigm in the Intelligent Patient Monitoring System (IPMS) represents an integration of different modelling paradigms within one framework. This integration is very significant because the same framework can be used to implement other aspects of the application as well as the probabilistic reasoning. Also during the run-time other components of the patient monitoring can access the results of probabilistic reasoning and simulation.

In order to demonstrate the use of CPNs, a generic CPN processing block was implemented using the xhugin Application Programmers Interface (API). This works in two modes, ***enter_evidence*** and ***retract_evidence***. In the enter_evidence mode the block is triggered by the arrival of evidence, *i.e.* measurements, which evidence is then propagated around the network to calculate the *a posteriori* distributions. For all root[25] and non-root[26] nodes, the probability distributions are calculated by the CPN algorithms. If an observation entered earlier is no longer valid, it can be retracted. In the retract_evidence mode the block is triggered by the arrival of a retract signal, on the corresponding node. The network is then updated to reflect the effect of retraction of evidence on that node. In the propagate function the *a posteriori* tables are used as old beliefs together with likelyhood ratios to calculate new beliefs. Thus the propagate function of xhugin allows for the incremental computation of the accumulative effect of evidence (Pearl, 1988), when the measurement set includes more than one item of data. All that the user (programmer) has to do is just to ensure that the inputs, outputs and parameters defined in the script are consistent with those defined for the block.

[25] Nodes that have no parents.

[26] Nodes that do have parents.

Potentially the CPN may be used for simulations. Assuming that a patient model can be created that is identifiable[27], the CPN models can be used to pose "what if" questions to predicate the effect of change in therapy before actually administering this change to the patient.

5.5 Summary

Using an application framework for the development of intelligent patient monitoring systems simplifies the implementation and allows reuse of software components. Software reuse will shorten development time; it should also lead to fewer bugs and can help hide a lot of implementation details from the application developer. Some general concepts have been identified that may form part of a patient monitoring systems. These components which included a signal processing library, a fuzzy logic library and a Causal Probabilistic Network (CPN) modelling and run-time environment, have been implemented in a generic form and integrated into the Intelligent Patient Monitoring System (IPMS) framework. This work, therefore, embodies two major achievements; the integration of these aspects within a single framework, as well as the reuse of the generic modules within each aspect. So the application developer can use this framework to implement, for instance, the signal processing or the fuzzy controller, by reusing the generic filter blocks or the general purpose fuzzy controller. The next chapter discusses the application of the framework in the development of a monitoring system for the respiratory system.

[27] The model parameters may be uniquely estimated.

CHAPTER 6

RESPIRATORY MONITORING APPLICATION

6.1 Introduction

In chapter 4 some general issues common to any intelligent patient monitoring system were addressed as well as some desired aspects of a model-based development environment for such systems. Implementation of some of these and their relevance to patient monitoring were discussed in chapter 5. The aim of this chapter is to show the use of the techniques in an exemplar application. An application for monitoring of the respiratory system is described.[28] Accordingly, this chapter consists of the following,

- Background to the problem
- Development and implementation of the application prototype
- The evaluation.

6.2 Background

The Adult Intensive Care Unit (AICU) of the Royal Brompton Hospital (RBH) cares for about 2080 patients annually, around 1400 of whom are made up of post-operative cardiac patients. The average Length Of Stay (LOS) of post-operative cardiac patients is two days with a range of less than one day (which is treated as zero) to over fifty days. The post-operative cardiac patients form a unique and interesting population. They are physiologically severely deranged yet they are low

[28] See sections F.6-F.8 in Appendix F for model and code listings.

risk with a mortality rate of less than 5%. The most common type of cardiac surgery is the bypass operation.

Coronary Artery Bypass Graft Surgery

Coronary Artery Bypass Graft (CABG) surgery is also called direct myocardial revascularisation or bypass surgery. The surgical procedure aims to restore myocardial perfusion[29] by bypassing the blockage in the myocardial circulation. The obstructed coronary artery may be the left main coronary artery, the left anterior descending artery, the right distal coronary artery, the posterior descending artery or the marginal branch of the circumflex artery. The bypass graft may be the saphenous vein harvested from the leg, or the internal mammary artery dissected from the anterior chest wall. About 90% of the CABG patients in the RBH have a combination of vein graft bypasses and arterial bypasses.

In the RBH all patients undergoing CABG return to the AICU immediately after surgery. So post-operative care in the AICU resembles recovery room care. Annually about 200 post-operative cardiac patients are managed under a so-called "fast track" style, which means that provided that there are no complications they will be discharged from the AICU and admitted to the general surgical ward as soon as the patients recover from the effects of the anaesthesia and are able to breathe spontaneously and be extubated. Fast track patients typically spend only six to eight hours in the AICU.

Respiratory complications are one of the most common types of all the complications that can occur after thoracic and cardiac surgery, and play a

[29]Blood flow to the muscles of the heart.

significant part during the recovery from cardiac surgery. All of the patients will suffer from some degree of acute lung injury, because typically during the surgery fluid in the interstitial spaces increases, leaving the lungs congested and waterlogged. Also the extracorporeal perfusion can damage the alveolar-capillary membrane causing interstitial oedema[30]. Other factors, such as the mechanical disturbance of the thorax caused by a median sternatomy[31] and the dissection of the mammary artery, effect of anaesthesia, sedation, pain and immobility can combine to cause hypoventilation and inadequate aeration of the lung bases and in turn causing atelectasis[32]. About 2% of post-operative cardiac patients develop Adult Respiratory Distress Syndrome (ARDS).

Adult Respiratory Distress Syndrome

First described in 1967 by Ashbaugh *et al.* (1967) and later named by Petty and Ashbaugh (1971), Adult Respiratory Distress Syndrome (ARDS) can most accurately be described as a pathological complex. ARDS is, therefore, an umbrella term for the composite manifestations of an evolving severe diffuse acute lung injury from all known and unknown causes. ARDS is characterised by refractory hypoxaemia[33], and decreased lung compliance secondary to permeability oedema[34].

---

[30] Lung water

[31] Split down the breast bone

[32] Collapse of the alveoli

[33] Low partial pressure of oxygen in arterial blood, due to improper oxygenation of the blood in the lungs, typically necessitating mechanical ventilation with a high fraction of inspired oxygen, FIO2>50% for P_aO2>6.8 kPa.

[34] Transfer of protein-rich fluid from the pulmonary capillaries to the interstitium and the alveolar spaces, (the gas-exchange compartment of the lungs), as the result of increased permeability of the alveolar wall *cf.* haemodynamic pulmonary oedema caused by high hydrostatic pressure in the capillaries. Pulmonary oedema is indicated on the chest radiography by the presence of bilateral diffuse infiltrates.

In the U.S. an estimated 150,000 cases, and in Britain (Wardle, 1984) between 10,000 and 15,000 cases, occur each year. Despite considerable improvements in the understanding of the mechanisms leading to ARDS, and significant advances in diagnosis and therapy, mortality remains high (Petty, 1985), over 50% and if accompanied by sepsis this is raised to 85%.

The administration of therapy must be optimised because the treatment can itself cause further damage. For instance, the administration of high fractions of inspired oxygen can cause fibrosis[35], and the long term use of positive pressure ventilation can cause barotrauma[36].

6.3 Application Prototype

6.3.1 The Model Building Process And The Models

The Monitoring Structure

The first step in building the application is to represent the structure of the monitored process. The structure meaning the arrangement and interactions of the processes that make up the monitored process. This involves a **hierarchical organisation** of the processes from a functional perspective. At a simple level the

[35]Scaring of the parenchyma, replacing the normally distensible tissue with dense fibrous tissue.

[36]Damage to the lung tissue as the result of local distending and shearing forces, caused by high pressures or volumes during positive pressure ventilation.

function of the respiratory system is the supply of the oxygen from air or inspired gas to tissue and the removal of carbon dioxide from the body. It also involves maintenance of the acid-base balance. In this prototype the function of the top level in the process hierarchy has been defined as the maintenance of the oxygen status even though the respiratory system also contributes to the maintenance of acid-base balance, through removal of carbon dioxide. A diagram of the respiratory system is shown in figure 6.1. In this diagram the lung is represented by a compartment representing the conducting airways, called the anatomical dead space, and the respiratory zone or the gas-exchange compartment, called the alveolar space. The alveolar space is perfused by the pulmonary circulation through the pulmonary capillary blood vessels. The arterial and venous vascular beds in the systemic circulation are simplified to a single artery and vein. Similarly the oxygen consumption of different tissue types is represented by a single tissue exchange. Figure 6.2 shows the respiratory process tree. The top level consists of the following processes:

- **O2 consumption**, representing the oxygen release and consumption at the tissue level;
- **O2 delivery**, with the oxygen flux or oxygen availability as the output, and consisting of the following three processes;
- **Sys circ**, representing the delivery of oxygen to the site of consumption through the systemic circulation;
- **O2 transport**, corresponding to the oxygen carriage in arterial blood by haemoglobin;
- **Oxygenation**, which represents the oxygen uptake in the lungs.

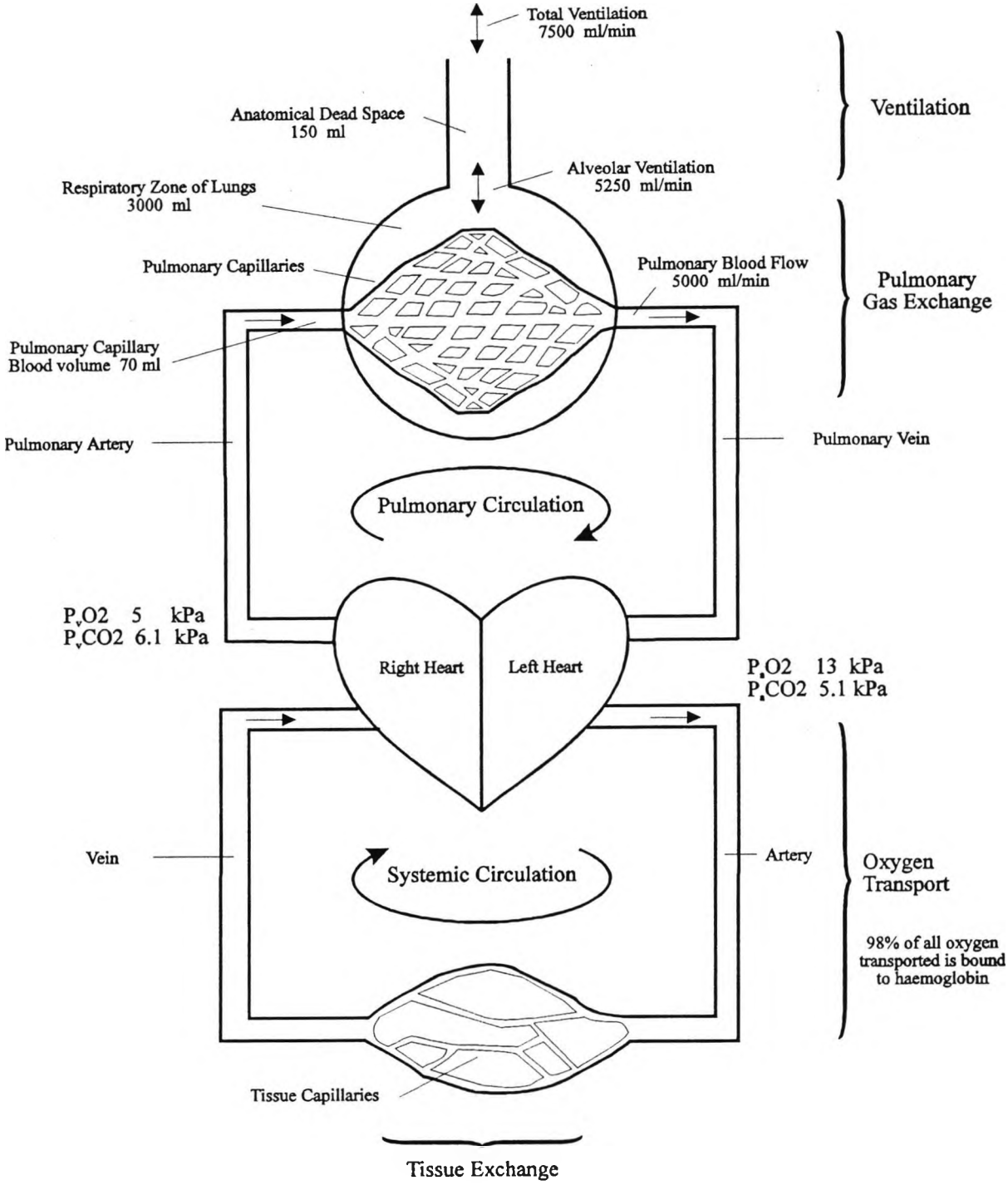

Figure 6.1 Diagram Of Respiratory System

Numbers represent typical values for normal adults at rest.

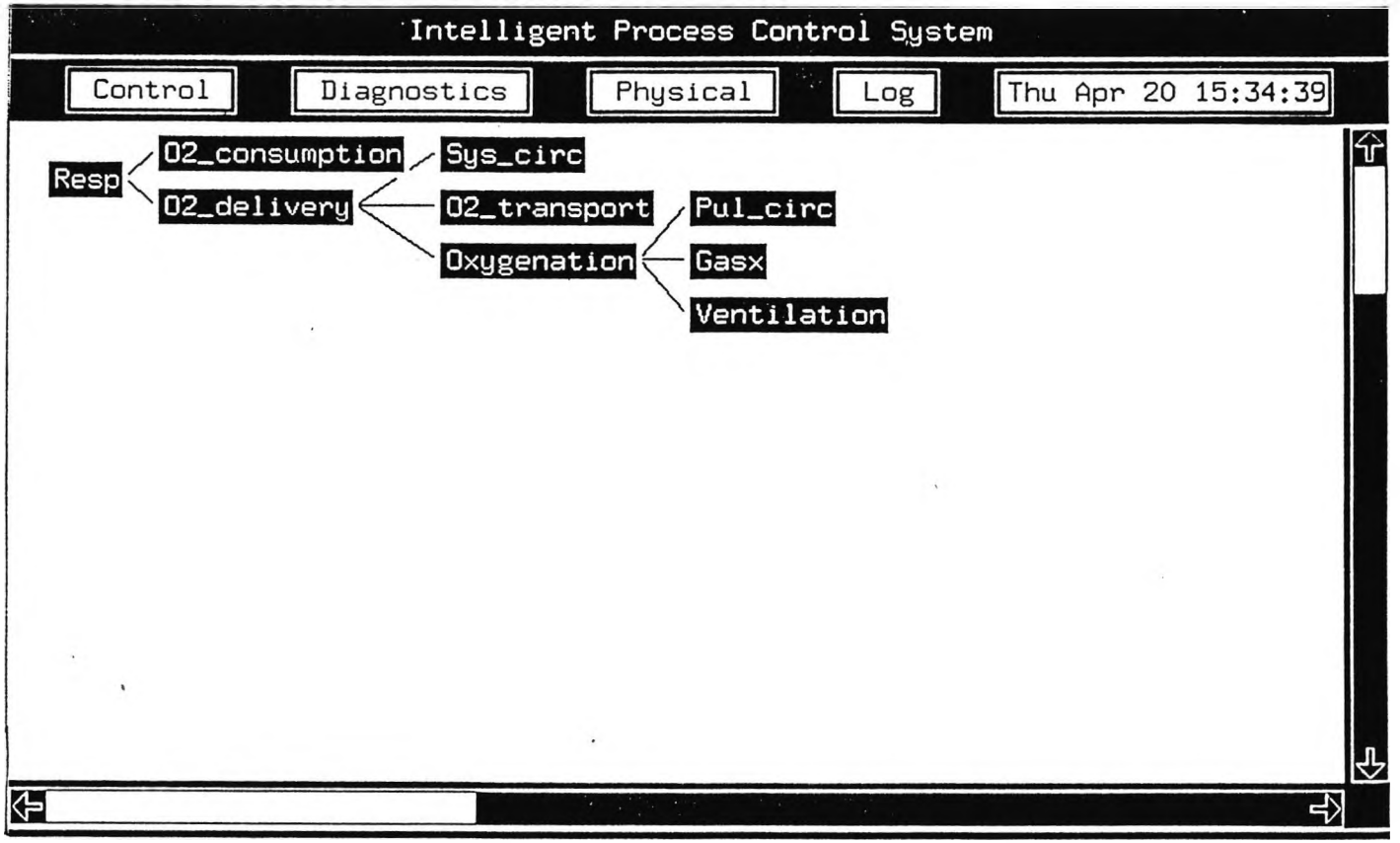

Figure 6.2 Operator Interface Root Panel Showing The Process Hierarchy

Finally the oxygen uptake at the lungs is made up of the interaction of the bottom three processes, namely

- **Ventilation**, or flow of inspired gas to the site of gas exchange;
- **Gasx**, corresponding to the transfer of gas across the blood-gas barrier by diffusion; and
- **Pul_circ**, corresponding to the pulmonary circulation.

The next step is to define the measurements and monitoring signals for all sub-processes, including the information about how each item is captured, displayed and recorded. For instance the top level should include all measurements related to the assessment of tissue oxygen status. This includes venous oxygen partial pressure, venous oxygen saturation, and blood lactate[37]. This type of information is captured in the **Monitoring** and **Operator interface** aspects of the model editor. Figure 6.3 shows the monitoring aspect of the ventilation process, in which the monitoring signals available from the ventilator are shown on the left hand side, together with the interface point objects feeding the corresponding signal object. The specification of the user interface panel for the ventilation process is shown in figure 6.4 which shows the signals feeding into the graph objects for x-y trend plots.

Table 6.1 contains all measurements and monitoring signals and calculated variables for all the processes. The first column lists the monitored processes, as

[37] Measure of lactic acidosis (lactacidosis) caused by anaerobic metabolism. It is not routinely used at Brompton AICU. It is possible under disease conditions such as ARDS or sepsis that the oxygen extraction at tissue becomes inefficient. Hence tissue hypoxia can be present and not indicated by a lowered PvO2. The lactate levels will be necessary to assess hypoxia. Venous levels >2mmol/l are significant and >4 mmol/l very significant.

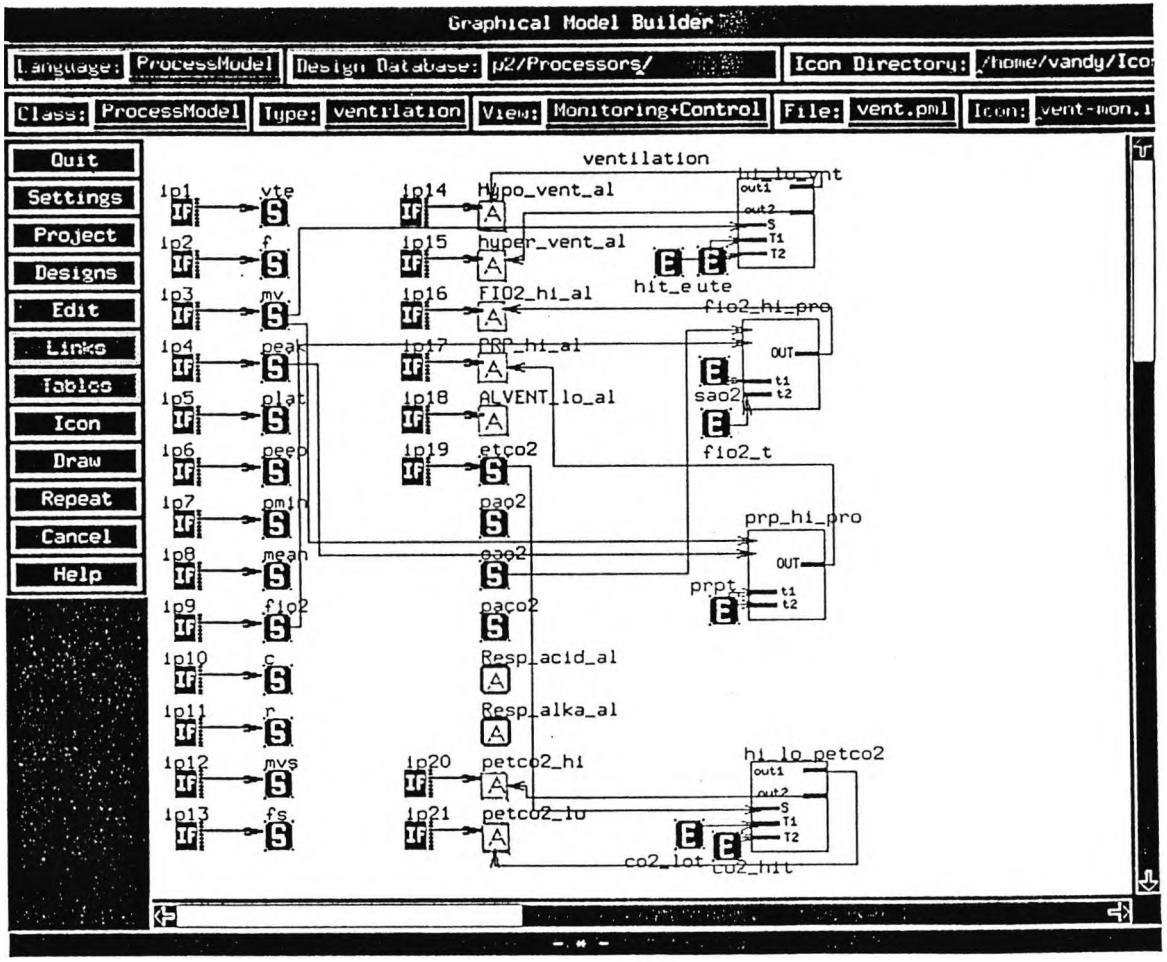

Figure 6.3 Monitoring Aspect Model For The Ventilation Process

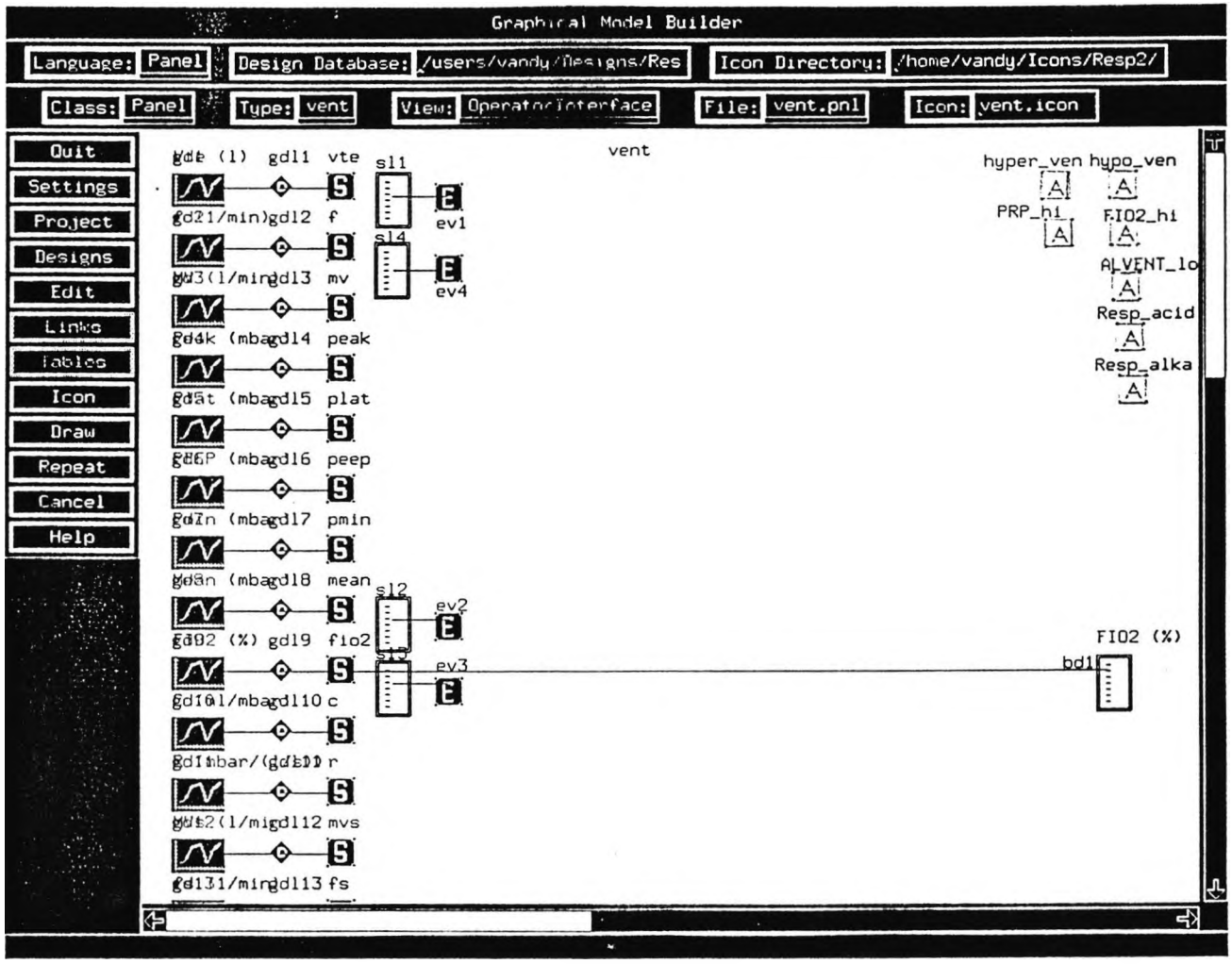

Figure 6.4 Operator Interface Model For The Ventilation Process

Process Name	Measurements		Measurement Description
	Measured	Calculated	
Resp	Svo2		venous oxygen saturation
O2_consumption		O2ER	oxygen extraction ratio; C[a-v]O2/CaO2
		VO2I	oxygen consumption index; C[a-v]O2.CI
O2_delivery		O2AVI	oxygen availability index; CaO2.CI
Sys_Circ	HR		heart rate
	ABPs		systolic arterial blood pressure
	ABPd		diastolic arterial blood pressure
	ABPm		mean arterial blood pressure
	CVP		central venous pressure
	CO		cardiac output
		CI	cardiac index; CO/BSA
		Pulse_P	pulse pressure; ABPs-ABPd
		SI	stroke index; CI/HR
O2_transport	CaO2		arterial oxygen content
	Hb		haemoglobin concentration
	p50		50% saturation pressure
	Dys_Hb		dyshaemoglobin fraction
Oxygenation	PaO2		arterial oxygen partial pressure
	SaO2		arterial oxygen saturation
	PaCO2		arterial carbon dioxide partial pressure
		AaDO2	alveolar-arterial oxygen difference; (BP-VP).FIO2-PaCO2.(FIO2+(1-FIO2)/R)-PaO2
Pul_Circ	PAPs		systolic pulmonary artery pressure
	PAPd		diastolic pulmonary artery pressure
	PAPm		mean pulmonary artery pressure
		Qs/Qt	intrapulmonary shunt; derived from model
Gasx		Rdiff	oxygen diffusion resistance;derived from model

Table 6.1 Process Measurements (continued on following page)

Process Name	Measurements		Measurement Description
	Measured	Calculated	
Ventilation	Vte		expired tidal volume
	f		respiratory frequency
	MV		minute volume
	Peak		peak airway pressure
	Plat		plateau airway pressure
	PEEP		positive end-expiratory pressure
	Pmin		minimum airway pressure
	Mean		mean airway pressure
	FIO2		fraction of inspired oxygen
	C		dynamic lung and thorax compliance
	R		airway resistance
	MVs		spontaneous minute ventilation
	fs		spontaneous breathing frequency
	PetCO2		end tidal partial pressure of carbon dioxide
		ALVENT	alveolar ventilation; MV-Vd.f
		PRP	peak respiratory power; MV.PIP

Table 6.1 Process Measurements (continued from previous page)

they have been defined. The measurement column, contains all the measured and calculated signal and data objects that are monitored by this prototype. So, for instance, the pulse pressure signal "Pulse_P", belongs to the systemic circulation process (Sys_Circ), and is calculated from the difference between the systolic and diastolic arterial blood pressures. The ventilation variables are sampled once every five seconds corresponding to a nominal breathing frequency of 12 breaths/min. All other variables are sampled once every two seconds. If the value of a variable is only available intermittently, such as the cardiac output or the haemoglobin concentration, the last available value is used. The unobservable variables *e.g.* alveolar ventilation are derived from measurements, whenever all necessary inputs are available *i.e.* the actor scripts responsible for these calculations propagate outputs at the rate of the slowest input. Alternatively they may be derived with reference to a model, for instance the shunt and diffusion resistance may be derived from the CPN oxygen status model (Summers *et al.*, 1993).

The Diagnostic Knowledge

Having defined the monitoring structure, the diagnostic knowledge is captured through the three abstractions, **Alarms**, **Failure modes** and **Causal links**. Alarms may be **On-line**, in which case they are generated or acquired by the system or **Off-line**, *i.e.* entered in by the user. The failure modes are directly derived from the functional definition of the process. Therefore a process is in failure whenever it fails to achieve its designated function or equally if it violates the constraints within which it is supposed to operate. The causal links are used to express the propagation of failures.

For the ventilation process, seven on-line and two off-line alarms have been defined. The on-line alarms may be created from thresholding single variables such as the hypo- and hyper-ventilation alarms created from the measured expired minute volume, or using a logical combination of several signals or alarms for intelligent alarming. For instance, the high peak respiratory power alarm (PRP_hi_al) is created from peak pressure and minute volume signals, or the excessive inspired oxygen alarm (FIO2_hi_al), which is generated using the fraction of inspired oxygen and arterial oxygen saturation.

The function of the ventilation process is to provide adequate alveolar ventilation, hence the failure modes, "low alveolar ventilation" (vdot_a_lo) and "low minute volume" (mv_lo). But the alveolar ventilation must be delivered in such a way not to cause respiratory alkalosis by excessively high minute volume (mv_hi), damage to the lungs through barotrauma (prp_hi), or fibrosis of the lung through toxic levels of inspired oxygen (fio2_hi). Each failure mode may be associated with one or more alarms, *e.g.* hypo-ventilation alarm, high end tidal partial pressure of carbon dioxide or the respiratory acidosis off-line alarm all can indicate the mv_lo failure mode (Figure 6.5).

The failure propagation of the top level is shown in figure 6.6. Having defined oxygen status as the top level, then the top level failure will be tissue hypoxia. This may be caused by increased oxygen consumption, related to for instance sepsis, or as the result of low oxygen delivery. In either case the imbalance in oxygen status will be reflected in an increase in the oxygen extraction ratio. Low oxygen delivery may be caused by low cardiac output or low arterial content (figure 6.7[38]). The

[38] In figures 6.7 and 6.8. some colours do not show due to the transformation from colour display to the black and white print-out. Where necessary these have been replaced by hand written notes.

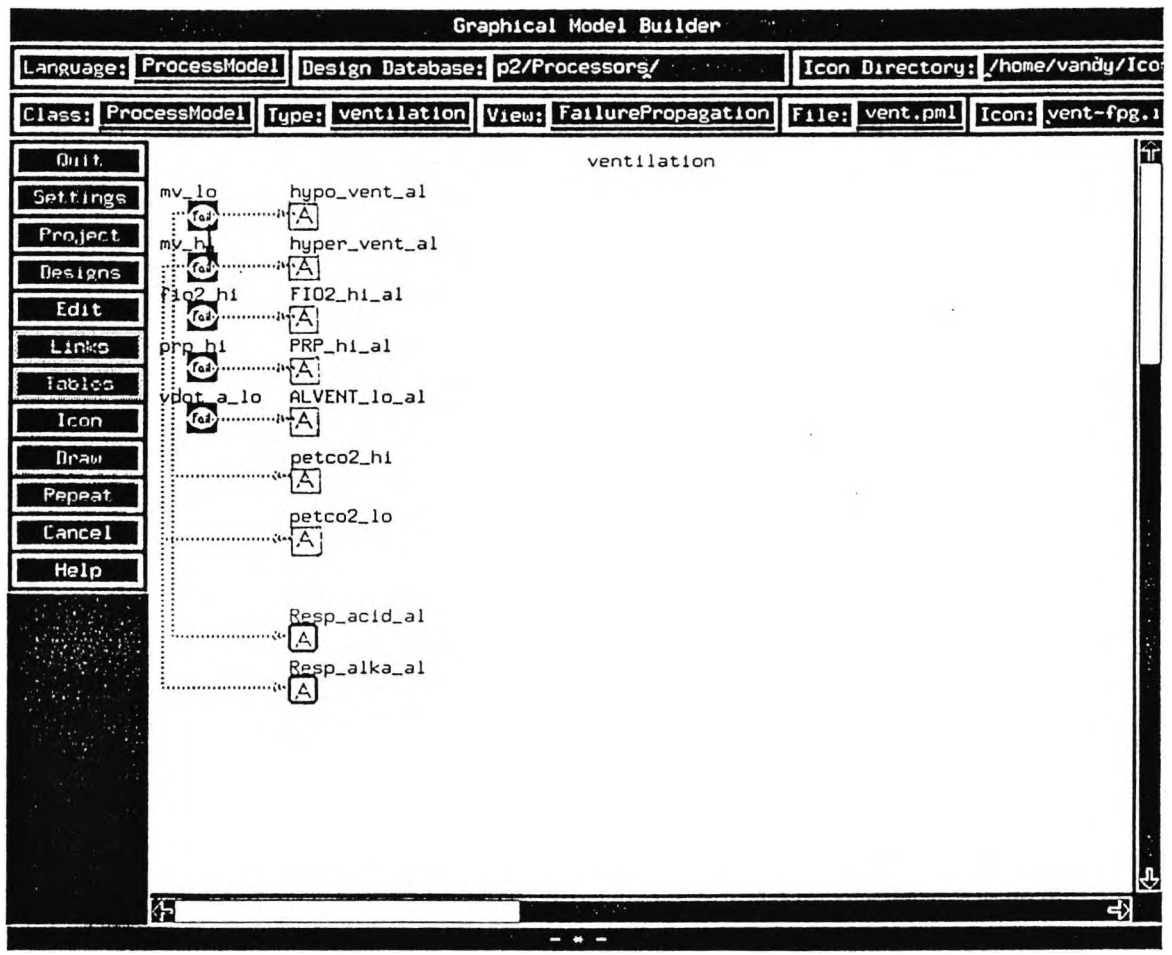

Figure 6.5 Failure Propagation Aspect For The Ventilation Process

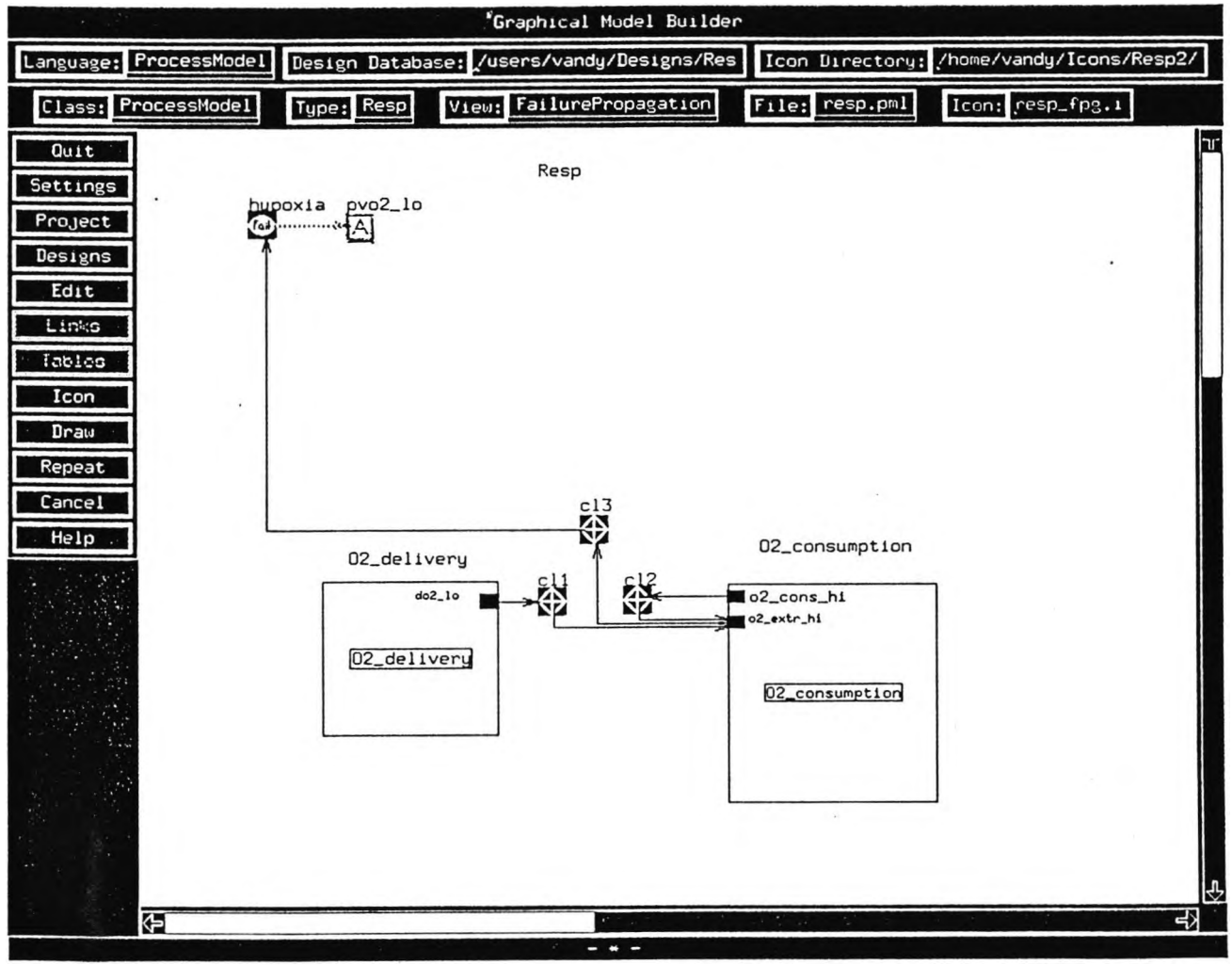

Figure 6.6 Failure Propagation Aspect For The Top Level

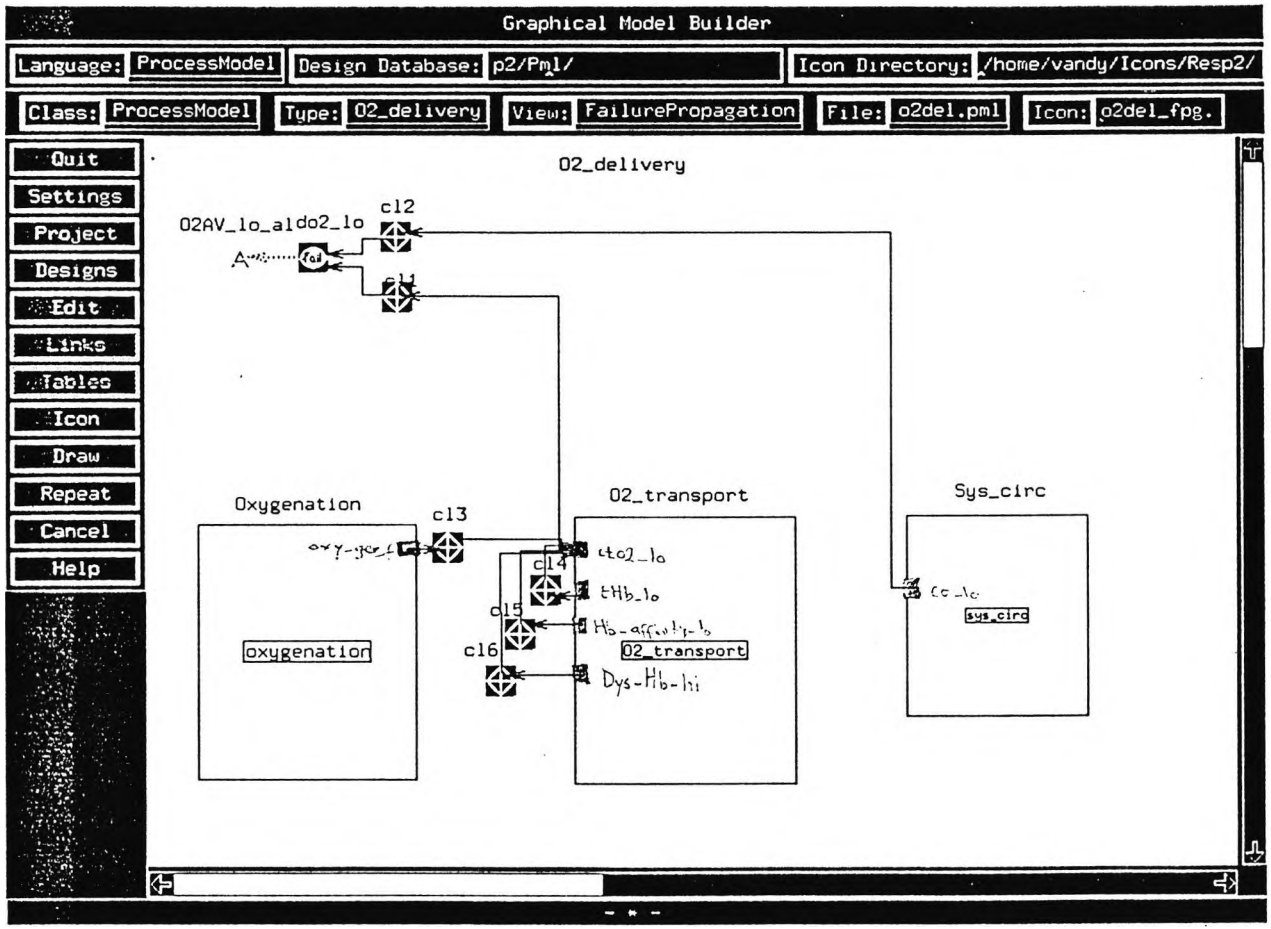

Figure 6.7 Failure Propagation Aspect For The O2_delivery Process

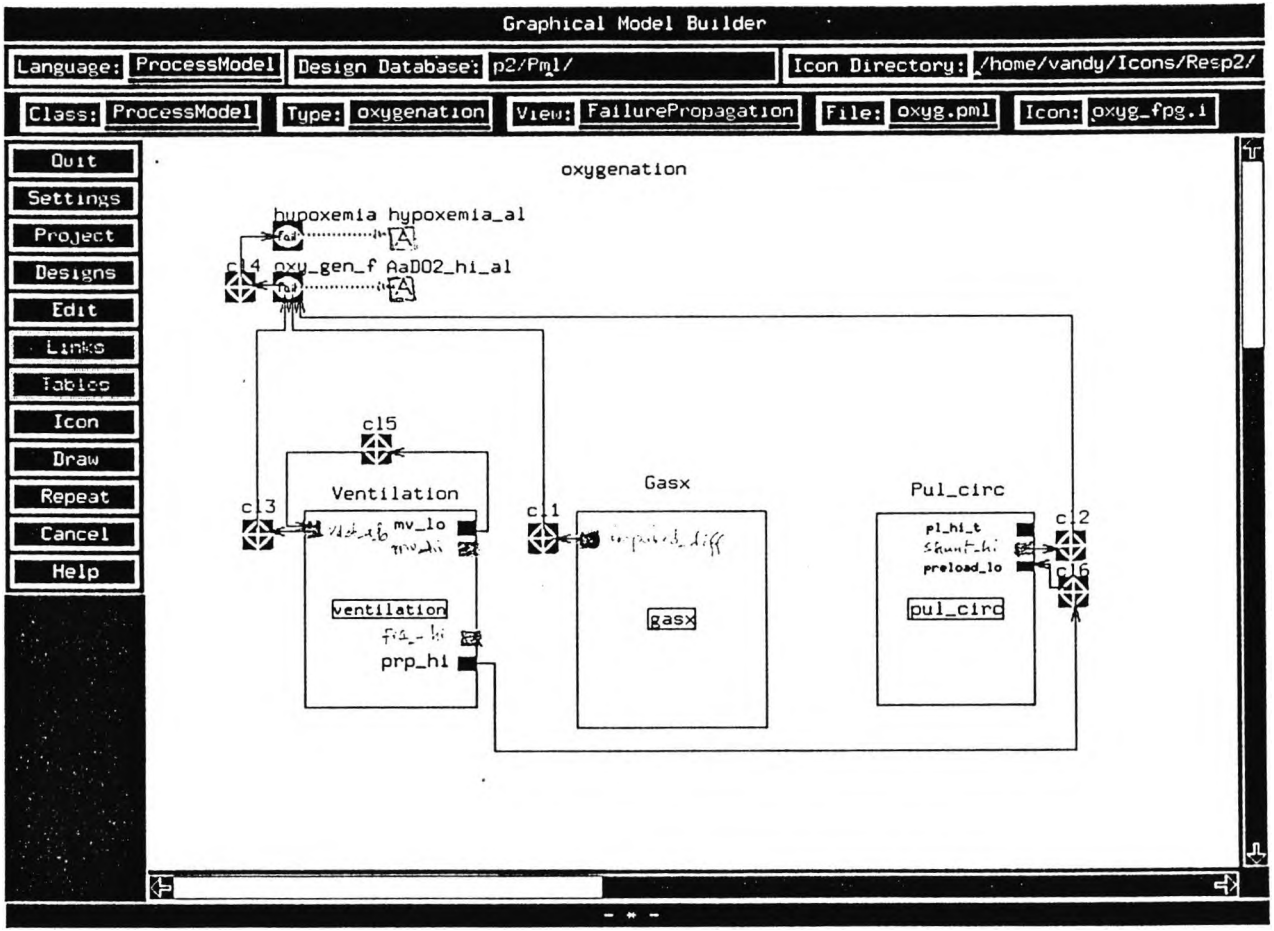

Figure 6.8 Failure Propagation Aspect For The Oxygenation Process

arterial oxygen content depends on oxygen uptake (**Oxygenation**) at the lung and the oxygen carrying capacity of the blood (**O2_transport**). The oxygen carrying capacity of the arterial blood depends on the concentration of total haemoglobin in the blood, the haemoglobin affinity, and the fraction of dyshaemoglobin. Figure 6.8[39] shows that a low oxygen uptake at the lungs may be caused by low alveolar ventilation, high pulmonary shunt fraction or impaired diffusion represented by a high diffusion resistance.

Tables 6.2 and 6.3 show all failure modes and alarms for the entire system. In both tables the monitored processes are entered in the first column, and the corresponding failures or alarms in the middle columns and the last column contains a description of the failure or alarm. So for instance, two failure modes have been defined for the oxygen consumption process (**O2_consumption**), namely high oxygen consumption "o2_cons_hi" and high oxygen extraction "o2_extr_hi"; and the oxygen delivery process (**O2_delivery**) has one on-line alarm, which is the low oxygen availability index "O2AVI_lo_al".

6.3.2 The Run-time System

The User Interface

The user interface for the run-time system is made up of a number of panels. There is a panel corresponding to each of the processes in the hierarchy, plus a root panel. Figure 6.2 shows the root panel of the user interface, which mirrors the process hierarchy. The state of each process is indicated by its colour in the root panel. The panels for individual processes may be accessed by clicking on the

[39] See foot-note 38.

Process Name	Failure Name	Failure Description
Resp	hypoxia	tissue hypoxia
O2_consumption	o2_cons_hi	high oxygen consumption
	o2_extr_hi	high oxygen extraction
O2_delivery	do2_lo	low oxygen delivery
Sys_Circ	co_lo	low cardiac output
O2_transport	cto2_lo	low oxygen content
	tHb_lo	low total haemoglobin concentration
	Hb_affinity_lo	low haemoglobin affinity
	Dys_Hb_hi	high dyshaemoglobin fraction
Oxygenation	hypoxaemia	hypoxaemia
	oxy_gen_f	general failure of oxygenation
Pul_circ	pul_hi_tens	pulmonary hypertension
	shunt_hi	high fraction of intrapulmonary shunt
	preload_lo	low preload
Gasx	impaired_diff	impaired oxygen diffusion
Ventilation	mv_lo	low minute volume
	mv_hi	high minute volume
	fio2_hi	high fraction of inspired oxygen
	prp_hi	high peak respiratory power
	vdot_a_lo	low alveolar ventilation

Table 6.2 Process Failures

Process Name	Alarm		Alarm Description
	On-line	Off-line	
Resp	SvO2_lo		low venous oxygen saturation
O2_consumption	O2ER_hi		high oxygen extraction ratio
	VO2I_hi		high oxygen consumption index
O2_delivery	O2AVI_lo_al		low oxygen availability index
Sys_Circ	Sin_Brady		sinus bradycardia
	Sin_Tachy		sinus tachycardia
		SI_lo	low stroke index
	ABPs_lo		low systolic arterial pressure
	ABPs_PP_lo		low systolic and pulse pressure
	CVP_lo		low central venous pressure
		CI_lo	low cardiac index
O2_transport		CaO2_lo_al	low arterial oxygen content
		Hb_lo_al	low haemoglobin concentration
		p50_hi	high p50
		Dys_Hb_hi_al	high dyshaemoglobin
Oxygenation	hypoxemia_al		hypoxemia
	AaDO2_hi_al		high alveolar-arterial difference
Pul_Circ	Pul_hi_t_al		pulmonary hypertension
	Qs/Qt_hi_al		high shunt
		PAPd-PAWP_hi	high difference of pulmonary diastolic and wedge pressure
		PAWP_hi	high pulmonary wedge pressure
		PAWP_lo	low pulmonary wedge pressure
Gasx	Rdiff_hi		high diffusion resistance
Ventilation	hypo_vent_al		low minute volume
	hyper_vent_al		high minute volume
	FIO2_hi_al		high fraction of inspired oxygen
	PetCO2_hi		high end tidal CO2 pressure
	PetCO2_lo		low end tidal CO2 pressure
	ALVENT_lo_al		low alveolar ventilation
	PRP_hi_al		high peak respiratory power
		Resp_alka_al	respiratory alkalosis
		Resp_acid_al	respiratory acidosis

Table 6.3 Process Alarms

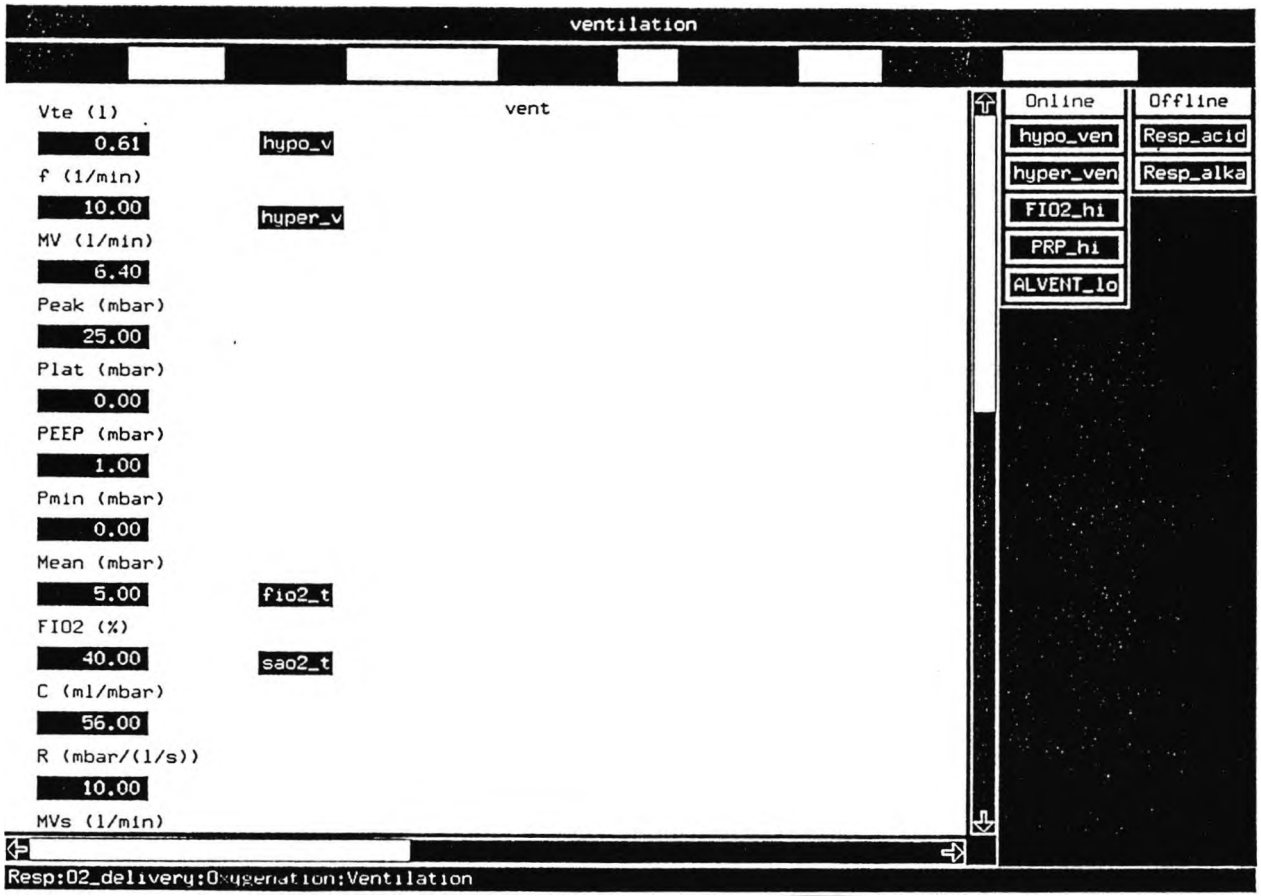

Figure 6.9 Operator Interface Panel For The Ventilation Process

process name in the root panel. The panel for each process contains all alarms, monitoring signals and the user inputs associated with that process. The panel for the ventilation process is shown in figure 6.9. The numerical displays on the left show the values of the ventilator parameters. Clicking on the numerical values opens the trend graph windows. The buttons on the next column open slider inputs for user adjustable alarm thresholds. The area on the right side of the panel displays the alarm's status.

The Diagnostic Messages

The alarms may or may not be associated with failure modes. If an alarm is not associated with a failure, the diagnostic log indicates that the alarm is ringing together with a time stamp, whenever it is triggered. For instance, the systemic circulation process (**Svs_Circ**) has sinus bradycardia[40] "Sin_Brady" and sinus tachycardia[41] "Sin_Tachy" alarms which are not associated with the only failure mode, low cardiac output "CO_lo". In the cases where alarms are associated with process failures, the diagnostic algorithm will select the smallest subset of failure modes as source of failure that is consistent with the information in the failure propagation models. If for instance the oxygen availability index low alarm "O2AVI_lo_al", the arterial oxygen content low alarm "CaO2_lo_al" and haemoglobin concentration low alarm "Hb_lo_al" are all ON, then the diagnostic will indicate only "Hb_lo" failure mode. Figure 6.10 shows a sample of the messages found in the diagnostic log.

[40] ECG with regular rhythm, rate less than 60 beats per minute.

[41] ECG with regular rhythm, rate between 100 and 160 beats per minute.

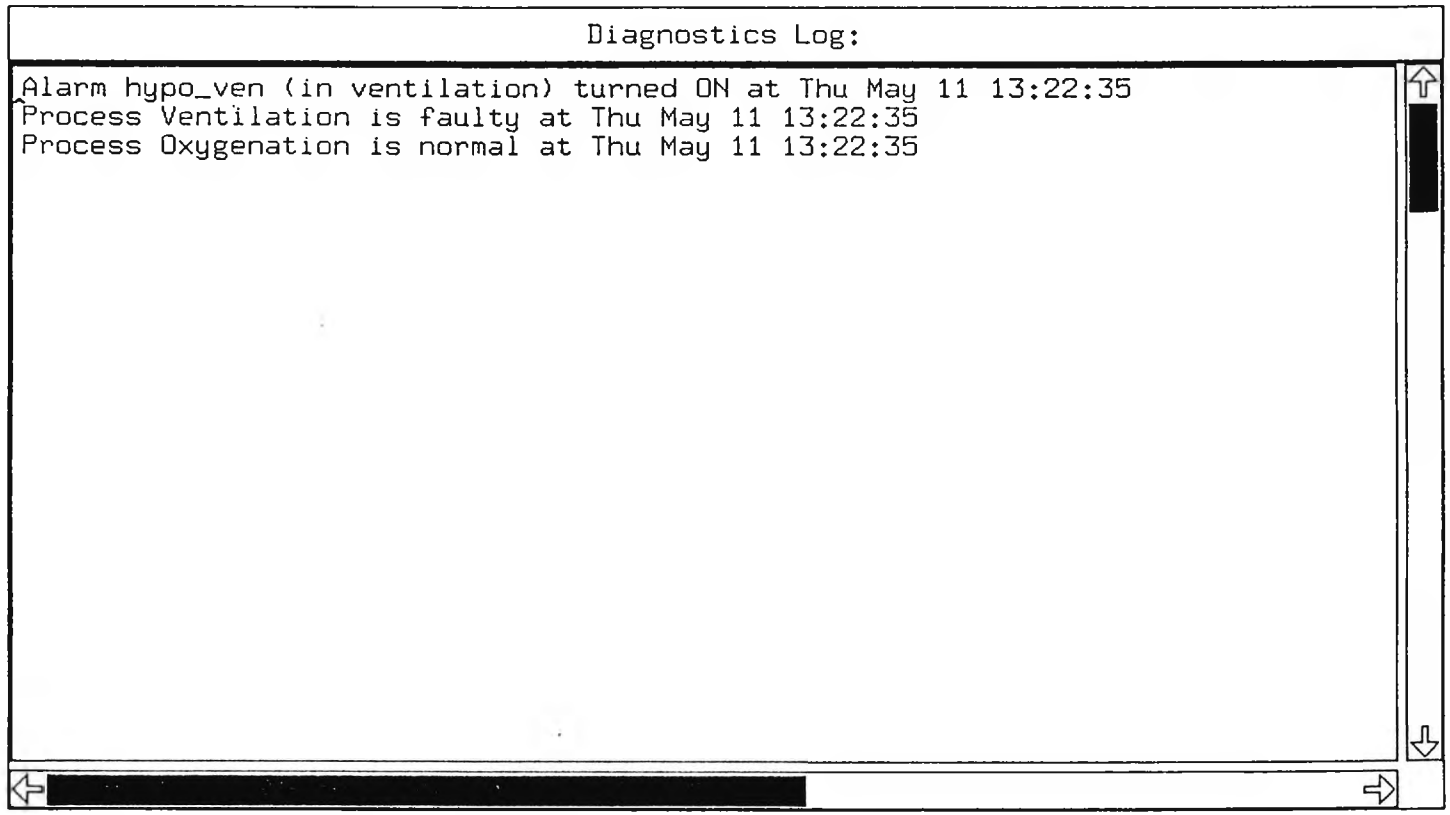

Figure 6.10 Sample Of Diagnostic Messages In The Log

Clinical Relevance

The system allows an integrated approach to monitoring the oxygen status of the patient. The special alarms PRP_hi_al and FIO2_hi_al should remind the less experienced clinicians of the need to optimise ventilator support in order to avoid further damaging the lungs with positive pressure ventilation. Because the structure of the monitoring is based on the process structures it reflects the different treatment options. These can be grouped into two broad categories; "Systemic" and "Pulmonary". In the systemic group treatment includes reducing the oxygen demand, ensuring adequate oxygen carrying capacity of the blood and giving cardiovascular support. The pulmonary group includes treatment for improving oxygenation such as shifting the oedema in the lungs by turning the patient or reducing the shunt fraction and dead space ventilation with lung volume recruitment techniques, overcoming diffusion resistance with higher fractions of inspired oxygen, maintaining adequate alveolar ventilation by increasing minute ventilation and finally measures to improve the parameters of the lung mechanics *i.e.* the airway resistance and compliance through suctioning and administration of bronchodilators.

To validate the system, data were collected from the AICU at Royal Brompton Hospital (RBH) and a retrospective study used these data for simulation when checking the system for consistency. Also the knowledge-base (presented as graphical models) was checked against the knowledge of a domain expert.

6.4 System Validation

The purpose of the data collection exercise was to collect data for use in the retrospective validation of the prototype. The data included printouts of patient charts from the Hewlett-Packard networked Patient Data Management System (PDMS) CareVue 9000, ventilator measurements and all blood gas report slips. The data for the simulation were generated by the data generator process from values stored in two types of files; one containing the ventilator data and the other containing all other measurements. The data from the ventilator data acquisition were recorded as ASCII files which the simulation accessed directly. The values for all other measurements (as shown in table 6.1) were taken from the entries in the patient charts and blood gas report slips to create the second data file. Appendix D contains a sample of the patient chart and the ventilator data file.

The ventilator interface was re-written in VISUAL BASIC for MS_WINDOWS to allow data collection from the ventilators, through the RS232 serial port, using a lap-top PC. This interface is able to diagnose communication failures. For instance it is capable of detecting the loss of connection and restarting itself once the connection is re-established. It times out from loops, and validates the data by checking the length of messages, the start and stop characters and the ranges of variables in addition to the one bit parity check for each character.

6.5 Model-Based Diagnosis

Model-based diagnosis consists of **prediction** of system behaviour according to the model, **comparison** of observed behaviour with predicted one, **change** of model parameters or hypotheses.

Typically in diagnostic systems models are rules describing the causal relations. However, in the engineering domain they can be the exact system equations in which case the comparison is called **residual generation**.

The IPCS diagnostic algorithm (Padalkar *et al.*, 1991) differs from similar systems in that it uses graph algorithms directly, rather than rule-based inference on a network of rules created from the graphs. Other similar systems use causal networks to represent failure propagation but then generate rules from the networks, and use Assumption based Truth Maintenance Systems (ATMS) algorithms on rules. In IPCS the diagnostic does a search on graphs of causal networks directly, utilising the timing constraint.

ATMS facilitate nonmonotonic reasoning by remembering the assumptions in reaching conclusions, so later they can be retracted if necessary. So each rule has two types of antecedents, **precondition** and **assumption**. The information necessary for resolving conflicts, *i.e.* history of the inferences and their associated assumptions, are recorded in assumption sets, the so-called **environment**. In contrast, in monotonic reasoning only the final conclusions are kept, so new facts about assumptions cannot be incorporated.

The IPCS diagnostic algorithm (Padalkar *et al.*, 1991) is basically a hypothesis and test loop. Two types of algorithms are used in the diagnostic system, the Faulty Component Identification Algorithm (FCIA) and the Interlevel Migration Process (ILMP). First the FCIA algorithm selects the largest subset of processes consistent with the current alarm information, then reduces (prunes the process tree) this subset using a timing constraint. Finally the ILMP migrates the FCIA to processes present at lower levels of process hierarchy and thus a higher resolution on the diagnosis is achieved.

In the respiratory monitoring application the physical (component) hierarchy is not used. In the industrial domain the physical hierarchy is the dominant hierarchy, because the aim is to identify a faulty component with a high degree of resolution. In this domain however the reasoning at organ level is sufficient and the aim of diagnosis is to differentiate between different disorders. Therefore the disorders were represented by the failures of the functional (process) tree, rather than the faults of the physical tree.

The diagnostic reasoning can reduce the number of suspect processes. So it implements the principle of parsimony *i.e.* if the higher level alarms ringing are consistent with the knowledge of failure dependence specified in the failure propagation models it will select the smallest subset of processes as in failure.

6.6 Summary

This chapter focused on the application of intelligent monitoring for the respiratory system. First the clinical background of the problem was described, which was then followed by a description of the prototype respiratory monitoring application. This included a summary of the model building process, a description of the diagnostic knowledge and the corresponding failure propagation models. It has been shown how using the Intelligent Process Control System (IPCS) concepts have helped organise the monitoring and gave it structure, by viewing it as a process hierarchy and organising the monitored signals accordingly; how multiple variables and their logical combinations were used to create alarms; how the diagnostic algorithm works; and how the model of behaviour was used in conjunction with models of failure. It has been demonstrated how the Intelligent Patient Monitoring System (IPMS) framework was used to integrate the components of patient monitoring and add on some degree of intelligence. A full discussion of the methodology and its application to patient monitoring can be found in the next chapter.

CHAPTER 7

DISCUSSION

7.1 Introduction

This chapter aims to bring together, and to discuss fully, the two aspects of this research, namely the methodological perspective (software aspect) and its application to patient monitoring (clinical application aspect). Therefore this chapter comprises two main sections. In the methodology section (7.2), the Multigraph Architecture (MA), is summarised and the model-based methodology is described. In the application section (7.3), the application of this methodology to patient monitoring, and the benefits gained are described.

7.2 Model-Based Methodology

7.2.1 Multigraph Architecture

The MA is a model-based architecture. In the MA the modelling languages allow knowledge to be expressed in the vocabulary of the domain, in contrast to first generation rule-based systems where no abstractions were defined. Another novelty of the MA approach, is in the use of a macro data flow, as a computational model, which facilitates parallel execution. MA integrates declarative and procedural styles of programming, by using the model interpreters to create execution structures (data flow graphs) from declarative structures (models). MA is a generic architecture which can be extended to new domains.

The model-based software synthesis allows programming at a higher level than merely coding. The benefits of this are the capability to handle the complexity associated with large scale applications, and fewer lines of code which is easier to debug.

The model-based framework that was used supported visual programming by providing graphical model building tools. Benefits of visual programming are that the technology is made accessible to a wider population, as well as making the development quicker for those familiar with the technology. Visual programming facilitates the reuse of code by representing generic functions or code segments with an icon and linking to other graphical parts. Reuse of code means development is faster and less repetitive.

7.2.2 Steps In Designing A Model-Based Framework

There are three methodological steps involved in designing a model-based framework, as was illustrated through the work on CPNs in chapter 5. These are design of modelling concepts, implementation of the algorithms and the corresponding model interpreters. The three steps are summarised below.

1. Design of the modelling concepts: The first step is to decide on the modelling **concepts** and appropriate modelling paradigms and their corresponding **representation**, *i.e.* the type of knowledge about the system that we wish to represent and the appropriate knowledge representation scheme. Issues that are to be considered here include:
    - the type of knowledge relevant to the **problem**,

- the modelling paradigms that are going to be suitable for the **domain**,
- design of the modelling concepts and their relations (such as inheritance) based on the appropriate **paradigm**, or in other words the design of the abstractions, *i.e.* the vocabulary of the domain, necessary for the knowledge representation.

2. The algorithms and model solutions: The next step is to specify the **algorithms** that operate on the modelling concepts and their corresponding **implementation** in software. This forms the generic run-time support for the models which includes the computation blocks of code and the object code libraries.
3. The model interpreters: This stage is specific to the MA. The model interpreters are responsible for generating the computational structure based on the models of the system and the algorithms. The model interpretation can be viewed as a mapping of models to computation structures or from declarative to procedural transformation. The model interpreters are specific to each modelling paradigm, but are independent of the model databases that specify a particular application.

7.3 Application To Patient Monitoring

Most of the other work in the field of intelligent patient monitoring consists of just one off applications. The approach described here is different because it has involved the design of a framework. This has not been done before. Furthermore it is a model-based framework. A special feature of this approach is that making new

applications becomes largely a matter of building the models, rather than writing code.

An important feature of the patient monitoring framework is the integration of different aspects of the application. In patient monitoring, the impact of this on the implementation stage is of special importance. For instance, the tools for knowledge-based software design typically do not allow for the design of user interface, or tools for probabilistic reasoning do not allow for data acquisition. The current prototype of the framework can be used to design all aspects of monitoring including data acquisition, graphical user interface, signal processing and control, and model-based reasoning. Furthermore it supports the development of causal probabilistic models using the same graphical editor as is used for other aspects. Also the framework has the flexibility to allow the integration of users own code with these kinds of systems. The inter-operation of different types of software, *e.g.* database and AI, is simplified by using the integrated development framework.

A major issue in the design of intelligent clinical informatics is the integration of the intelligent software with the existing computing environment. The Patient Data Management System (PDMS) at the Royal Brompton Hospital (RBH), is based on hp-9000 workstations from Hewlett-Packard, running the Hewlett-Packard version of UNIX (HP-UX), and Motif windowing system. The prototype respiratory monitoring system developed at City University is based on the Intelligent Process Control System (IPCS) framework which works on SUN work stations, running SUN's version of UNIX (SunOS) and OPEN LOOK windowing system. However the IPCS developed at Vanderbilt University also has an Hewlett-Packard port. Applications developed on one platform also work directly on the other one. Therefore the application can potentially run on the bedside workstations of the

PDMS, provided the manufacturers, in this case Hewlett-Packard, open their system to user or third party modifications.

One of the requirements of intelligent patient monitoring is robustness. One way of making the monitoring robust is to make it adaptive, *i.e.* dynamically modifying the behaviour according to some context. Therefore the purpose of context dependent monitoring is to make the monitoring adaptive and thereby making the monitoring more robust. In patient monitoring the therapeutic goals and therefore signal interpretation may change depending on for example the patient history. For instance, in the presence of Chronic Obstructive Pulmonary Disease (COPD), the clinician may aim for a small amount of hypercapnia rather than aiming for normocapnia, if it can be more dangerous to the patient to try to correct this hypercapnia than to allow it to persist. The context may be static *e.g.* age, sex, patient history, or dynamic *e.g.* patient state.

The key concept is to alter in some way the monitoring according to changes in the monitoring environment, *i.e.* changes in the patient and the therapeutic interventions. Mathematically this can be represented as an adaptation algorithm g, which maps the environment model M at time t, M_t, to the monitoring function f at time t, f_t, g: $M_t \rightarrow f_t$ (Sztipanovits *et al.*, 1993). The implication of this is that it is necessary to have some way of modelling the patient. Therefore the monitoring context is a model of the patient and/or the interventions. For example the monitoring context may be a single variable of discrete values, in which case it will have to summarise all the available information about the patient. The mechanisms for implementing dynamic reconfigurability already exist within the framework for patient monitoring. Therefore the respiratory monitoring application can be made more robust by using this feature of the framework. However this was not

implemented in the current prototype because it was not clear how to define the context, and the transition in the context and the change in the monitoring structure as the result of the change in context. More research is required in this area to answer these questions (see section on further work in the next chapter.)

The advantages of using this framework over a rule-based shell were the capability to:

- design the graphical user interface without having to know about X- windows programming, by using the panel editor of the IPCS framework,
- handle the management of data collection through interface objects and the data acquisition server,
- organise the knowledge through the use of models and the abstractions and the modelling concepts,
- graphically represent the knowledge; this meant that clinicians were able to understand and modify the system by graphical editing, *i.e.* drawing diagrams, rather than having to look at code,
- extend the framework for example by integrating the CPN modelling into IPMS and link to run-time system through actor scripts using the Hugin Application Programmers Interface (API),
- have automatic code generation, by using the model interpretation process to create the data flow graph of the application based on the models and the run-time support,
- run in real-time, by scheduling the actors of the data flow graph to trigger whenever their input data arrive,

- link knowledge about system failures, *i.e.* failure propagation graphs, with knowledge of the system behaviour, in causal network models.

The implementation of this prototype, using the framework together with the application independent building blocks, took over 100 man hours. This does not include the time spent in the design stage or the implementation of the generic blocks. An incremental approach was taken in the implementation, *i.e.* a small but functional system was first developed which was subsequently grown to the final version. This was partly responsible for slowing down the implementation. About 90% of the implementation time was spent on graphical model building. In visual programming every component of the system must be represented by an icon. Therefore a significant portion of this graphical model building time was spent on creation of icons. The model building process can be significantly sped up, and made more pleasant and acceptable to the application developer by supporting it with very sophisticated graphical and icon editors.

Although the monitoring application is not sufficiently developed to be used in a real clinical setting the application is very significant because:

- it is based on a novel architecture and represents a methodological advancement,
- it addresses all aspects of application development, such as data acquisition, user interface, signal processing, model-based reasoning,
- it addresses the integration of different types of software such as signal processing, AI, database,
- it contains over 50 on-line and off-line variables and alarms,

- it embodies a reorganisation of the respiratory knowledge in the form of the structural models of the respiratory system.

7.4 Summary

This chapter contained two main sections. In the methodology section the Multigraph Architecture (MA) and the design steps for a model-based architecture were summarised. The methodology section also highlighted the aspects which make the approach described in this thesis unique. In the following section the application of the multigraph architecture to patient monitoring was discussed. This latter section included a discussion of the benefits of using the MA and the Intelligent Patient Monitoring System (IPMS) framework for implementing patient monitoring applications. These include, for instance, integration of different aspects of application development, portability of the application, reuse of code and the impact of this on the development time, use of models for software synthesis, model-based reasoning and knowledge organisation, the extension of the modelling paradigms to include Causal Probabilistic Network (CPN), benefits of visual programming. The next chapter contains some concluding remarks, including a statement of how the objectives were achieved, possible plans for further work and the contribution to the field that is represented by this work.

CHAPTER 8

CONCLUSIONS

Achievement Of Aims And Objectives

The aim of the research work was to investigate the suitability and applicability of the multigraph architecture in building intelligent patient monitoring systems. In doing so a systemic approach has been adopted, by first showing the short comings of current knowledge-based techniques and proposing that a model-based approach will improve on these techniques. Also in agreement with good software engineering practices of modular and reusable design, the adoption of a framework has been proposed that can be used in development of intelligent model-based patient monitoring systems. Some software requirements for such a framework were then created, based on the experiences of the engineering community in intelligent process monitoring and the Intelligent Process Control System framework. Next some techniques that have been successfully applied in patient monitoring, that reflect the unique requirements of this domain, were identified and implemented as extensions to the Intelligent Process Control System to make the Intelligent Patient Monitoring System. Finally a prototype application based on the prototype framework was developed.

Contributions To The Field

This thesis explored and discussed the issues relating to High Dependency Environment (HDE) data systems, from computational, organisational and management perspectives in order to form a framework for design and implementation of intelligent patient monitoring systems in HDE. The work described in chapters four, five and six form the original contribution to the field. Chapter four described the generic software requirements as a specification for the framework. In chapter five some commonly used components were identified,

implemented in a generic form, and integrated within the framework. This work represents the extensions to the Multigraph Architecture (MA) tool set for three different purposes. The median and Infinite Impulse Response (IIR) filters for signal processing, the fuzzy library for use in a fuzzy controller, and the Causal Probabilistic Network (CPN) for probabilistic modelling. The work in chapter six represents extension of the MA application to the clinical domain. This work also includes an original restructuring of the respiratory knowledge to facilitate diagnostic reasoning. The three chapters together contain the design, implementation, and application of a model-based framework for use in patient monitoring.

The work contributes to the field of AI in medicine, by the application of a novel software architecture and the benefits it brings. Also part of the work is independent of implementation and the lessons learned are of general interest. Two kinds of users may benefit from the work: the application developer who may use the extended IPCS tool set in medical as well as other domains; and implicitly the clinical end-user who benefits from an enhanced instrument functionality.

Further Work

An important feature of intelligent patient monitoring systems is robustness. As described in the previous chapters one way of making the monitoring robust is to make it adaptive by dynamically modifying the behaviour according to some context. The framework already supports mechanisms for dynamic reconfigurability, however more research is necessary before these can be applied usefully. There are plans for introducing protocol guided patient care, CarePlan, in the Royal Brompton Hospital (RBH). For CarePlan, very specific diagnostic

patient groups need to be defined, so that the therapeutic goals and strategies can be accurately defined. This same classification may be used as the monitoring context. This classification may also include information about the therapeutic interventions, *e.g.* the ventilation mode, and the drugs that are in effect. The patient database of the Patient Data Management System (PDMS) can be used in estimating mean values of variables for different diagnostic groups. This requires a Structured Query Language (SQL) interface to down-load over the network the database from the Hewlett-Packard database to a PC database, before the data are permanently removed.

REFERENCES

Abbott, B., Biegl, C., Sztipanovits, J. 1990, "Multigraph for the transputer", Proceedings of 3rd North American Transputer User Group, Santa Clara, CA, pp. 25-36.

Abbott, B., Bapty, T., Biegl, C., Karsai, G., Sztipanovits, J. 1993, "Model-based software synthesis", IEEE Software, vol. 10, no. 5, pp. 42-52.

Andersen, S. K., Olesen, K. G., Jensen, F. V., Jensen, F. 1989, "HUGIN - A shell for building bayesian belief universes for expert systems", Proceedings of 11th International Joint Conference on Artificial Intelligence In (IJACAI-89), Sirdharan, N. S., (ed.), San Mateo, Morgan Kaufmann, vol. 2, pp. 1080-1085.

Ashbaugh, D. G., Bigelow, D. B., Petty, T. L., Levine, B. E. 1967, "Acute Respiratory Distress in Adults", Lancet, vol. 2, pp. 319-323.

Biegl, C. 1988, "Design and implementation of an execution environment for knowledge based systems", Ph.D. thesis, Department of Electrical Engineering, Vanderbilt University, Nashville, TN.

Blokland, W. and Sztipanovits, J. 1988, "Knowledge-based approach to re-configurable control systems", Proceedings of American Control Conference, Atlanta, GA.

Carson, E. R., Chelsom, J. J. L., Summers, R. 1991, "Progress with measurement, information and decision-making in critical care medicine", Measurement, vol. 9, no. 3, pp. 104-110.

Dawant, B. M., Uckun, S., Manders, E. J., Lindstrom, D. P. 1993, "The SIMON project", IEEE Engineering in Medicine and Biology Magazine, vol. 12, no. 4, pp. 82-91.

Factor, M., Sittig, D. F., Cohn, A. I., Gelernter, D., Miller, P. L. and Rosenbaum, S. H. 1990, "A parallel software architecture for building intelligent medical

monitors", International Journal of Clinical Monitoring and Computing, vol. 7, no. 2, pp. 117-128.

Fagan, L. M. 1980, "VM: Representing time dependent relations in a medical setting", Ph.D. Dissertation, Stanford University, CA.

Forbus, K. D. 1984, "Qualitative process theory", Ph.D. Dissertation, MIT, Cambridge MA.

Ford (ed.), K. M. 1993, "Knowledge acquisition as modelling", New York, John Wiley.

Gallagher Jr., N. C., Wise, G. L. 1981, "A theoretical analysis of the properties of median filters", IEEE Transactions on Acoustics, Speech, Signal Processing, vol. ASSP-29, pp. 1136-1141.

Garfinkel, D. *et al.* 1987, "HORNET Hospital Operating Room Network, A First Description", Proceedings of Symposium on Computer Applications in Medical Care 1987, p.817.

Garfinkel, D., Matsiras, P. V., Lecky, J. H., Aukberg, J. S., Mavrides, T. G., Matschinsky, B. 1988, "PONI: An intelligent alarm system for respiratory and circulation management in the operating room", Proceedings of Symposium on Computer Applications in Medical Care 1988, p. 13.

Harel, D. 1992, "Biting the silver bullet - Toward a brighter future for system development", IEEE Computer, vol. 25, no. 1, pp. 8-20.

Hayes-Roth, B. 1990, "Architectural foundations for real-time performance in intelligent agents", Real-time systems: The International Journal of Time Critical Systems, vol. 2, pp. 99-125.

Hayes-Roth, B., Washington, R., Ash, D., Hewett, R., Collinot, A., Vina, A., Seiver, A. 1992, "GUARDIAN: A prototype intelligent agent for intensive care monitoring", Artificial Intelligence in Medicine, vol. 4, no. 2, pp. 165-185.

Heckerman, D. E. 1986, "Probabilistic interpretations for MYCIN's certainty factors", In Uncertainty in Artificial Intelligence, Kanal, L. N. and Lemmer, J. F. (eds.), New York, North-Holland, pp. 167-196.

Heckerman, D. E. and Shortliffe, E. H. 1992, "From certainty factors to belief networks", Artificial Intelligence in Medicine, vol. 4, no. 1, pp. 35-52.

Henderson, S., Crapo, R. O., Wallace, J., East, T. D., *et al.* 1992, "Performance of computerized protocols for the management of arterial oxygenation in an intensive care unit", International Journal of Clinical Monitoring and Computing, vol. 8, pp. 271-280.

Hernández-Sand, C., Moret-Bonillo, V., Alonso-Betanzos, A. 1989, "ESTER: An Expert System for Management of Respiratory Weaning Therapy", IEEE Transactions on Biomedical Engineering, vol. 36, no. 5, pp. 559-564.

Higgins, G. 1992 "The cognitive psychology of the monitoring environment", Abstracts of Annual meeting of European Society for Computing and Technology in Anaesthesia and Intensive Care (ESCTAIC-92), p. F5.

Howard, R. A. and Matheson, J. E. 1981, "Influence diagrams", In Readings on the Principles and Applications of Decision Analysis, vol. 2, R. A. Howard and J. E. Matheson (eds.), Menlo Park, CA, Strategic Decision Group, pp. 721-762.

Hunter, J., Chambrin, M., Collinson, P., Groth, T., Hedlund, A., Kalli, S., Kari, A., Lenoudias, G., Ravaux, P., Ross, D., Salle, J., Sukuvaara, T., Summers, R., Zaar, B. 1991, "INFORM: integrated support for decisions and activities in

intensive care", International Journal of Clinical Monitoring and Computing, vol. 8, no. 3, pp. 189-199.

Isaka, S. and Sebald, A. V. 1993, "Control Strategies for arterial blood pressure regulation", IEEE Transactions on Biomedical Engineering, vol. 40, pp. 353-363.

Jensen, F. V., Olesen, K. G., Andersen, S. K. 1990, "An algebra of bayesian belief universes for knowledge based systems", Networks, vol. 20, no. 5, pp. 637-659.

Johnson, R. E. 1988, "Designing reusable classes", The Journal of Object-Oriented Programming, vol. 1, no. 3, pp. 22-35.

Karsai, G. 1988, "Declarative programming techniques for engineering problems", Ph.D. Dissertation, Department of Electrical Engineering, Vanderbilt University, Nashville, TN.

Karsai, G. 1990, "How to write a new graphical model builder", Technical report, Department of Electrical Engineering, Vanderbilt University, Nashville, TN.

Karsai, G., Sztipanovits, J., Padalkar, S., Biegl, C., Miyasaka, N., Okuda, K. 1990, "Model-based techniques for intelligent process control", Applications of Artificial Intelligence in Engineering, vol. 2, Manufacturing and Planning, Rzevski, G., (ed.), pp. 76-96.

Karsai, G., Sztipanovits, J., Padalkar, S., Biegl, C. 1992, "Model based intelligent process control for co-generator plants", International Journal of Parallel and Distributed Computing, vol. 15, no. 2, pp. 90-102.

Kim, J. H. and Pearl, J. 1983, "A computational model for combined causal and diagnostic reasoning in inference systems", Proceedings of 8th International Joint Conference on Artificial Intelligence, Los Angeles, pp. 190-193.

Koski, E. M. J., Mäkivirta, A., Sukuvaara, T., Kari, A. 1990, "Frequency and reliability of alarms in the monitoring of cardiac postoperative patients", International Journal of Clinical Monitoring and Computing, vol. 7, no. 3, pp. 129-133.

Kosko, B. 1992a, "Neural networks and fuzzy systems", Englewood Cliffs, NJ, Prentice-Hall International Inc., chap. 8, pp.229-338.

Kosko, B. 1992b, "Neural networks and fuzzy systems", Englewood Cliffs, NJ, Prentice-Hall International Inc., chap. 1, pp. 32-34.

Laffey, T. J., Cox, P. A., Schmidt, J. L., Kao, S. M., Read, J. Y. 1988, "Real time knowledge based systems", Artificial Intelligence Magazine, vol. 9, no. 1, pp. 27-45.

Laffey, T. J. 1991, "The real-time expert", Byte, vol. 16, no. 1, pp. 259-264.

Lauritzen, S. L. and Spiegelhalter, D. J. 1988, "Local Computations with probabilities on graphical structures and their applications to expert systems", Journal Royal Statistical Society, series B, vol. 50, no. 2, pp. 157-224.

Leaning, M. S., Yates, E. C., Patterson, D. L. H., Ambroso, C., Collinson P. O. and Kalli, S. T. 1991, "A data model for intensive care", International Journal of Clinical Monitoring and Computing, vol. 8, no. 3, pp. 213-224.

Linkens, D. A. and Mahfouf, M. 1988, "Fuzzy logic knowledge based control for muscle relaxant anaesthesia", 1st International Federation of Automatic Control symposium on modelling and control of biomedical systems, pp. 185-190.

Linkens, D. A. and Hasnain, S. B. 1991, "Self-organising fuzzy logic control and application to muscle relaxant anaesthesia", IEE Proceedings D., vol. 138, no. 3, pp. 247-284.

Mäkivirta, A. 1989, "Towards reliable and intelligent alarms by using median filters", A thesis for the Degree of Licentiate of technology, Tampere University of Technology, Finland.

McCarthy, J. 1960, "Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I", Communications of the Association for Computer Machinery, vol. 3, no. 4.

McCarthy, J. 1978, "History of LISP", ACM SIGPLAN Notices, vol. 13, no. 8. Also in History of Programming Languages, Wexelblat, Richard L., (ed.), Academic Press, New York.

Meier, R., Nieuwland, J., Hacisalihzade, S., Steck, D., Zbinden, A. 1992, "Fuzzy control of blood pressure during anesthesia with isoflurane", Proceedings of IEEE International Conference on fuzzy logic, pp. 981-987.

Misra, A., Abbott, B., Sztipanovits, J. 1990, "Performance optimization in signal processing systems", Proceedings of 22nd Southeastern Symposium on System Theory, Cookville, TN, pp. 641-645.

Moret-Bonillo, V., Alonso-Betanzos, A., Martín, E. G., Canosa, M. C., Berdiñas, B. G. 1993, "The PATRICIA project", IEEE Engineering in Medicine and Biology Magazine, vol. 12, no. 4, pp. 59-68.

Nicholls, A., Shenton, M. 1992, "The 'REAKT' project- Supporting RTKBS development", IEE colloquium on real-time knowledge based systems, digest No. 1992/217.

Nodes, T. A., Gallagher Jr., N. C. 1984, "The output distribution of median type filters", IEEE Transactions on Communication, vol. COM-32, pp. 532-541.

O'Keefe, R., Balci, O., Smith, E. 1987, "Validating expert system performance", IEEE Expert, vol. 2, no. 4, pp. 81-89.

O'Rielly, C. A., Cromarty, A. S. 1985, "Fast is not real-time in designing effective real-time AI systems", In Applications of Artificial Intelligence, vol. II, Bellingham, WA, International Society of Optical Engineering, p. 548.

Padalkar, S., Karsai, G., Biegl, C., Sztipanovits, J., Okuda, K., Miyasaka, N. 1991, "Real-time fault diagnostics", IEEE Expert, vol. 6, no. 3, pp. 75-85.

Pearl, J. 1986, "A constraint-propagation approach to probabilistic reasoning", In Uncertainty in Artificial Intelligence, L. N., Kanal and J. F., Lemmer (eds.), Amsterdam, North Holland, pp. 357-370.

Pearl, J. 1987, "Evidential reasoning using stochastic simulation of causal models", Artificial Intelligence, vol. 32, no. 2, pp. 245-257.

Pearl, J. 1988, "Probabilistic reasoning in intelligent systems", San Mateo, CA, Morgan Kaufmann Publishers Inc., chap. 2, pp. 37-39.

Petty, T. L. and Ashbaugh, D. G. 1971, "The adult respiratory distress syndrome: Clinical features, factors influencing prognosis and principles of management", Chest, vol. 60, pp. 233-239.

Petty, T. L. 1985, "Indicators of risk, course, and prognosis in adult respiratory distress syndrome (ARDS)", American Review of Respiratory Diseases, vol. 132. p. 471.

Pirjamali, R. N., Summers, R., Sztipanovits, J., Carson, E. R. 1993, "Application of the Multigraph architecture in intelligent patient monitoring", Proceedings of 15th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Szeto, A. Y. J., Rangayyan, R. M., (eds.), Piscataway, NJ, vol. 15, no. 2, pp. 608-609.

Proakis, J. G. and Manolakis, D. G. 1989, "Introduction to digital signal processing", New York, Macmillan Publishing Company.

Pryor, T. A., Gardner, R. M., Clayton, P. D., Warner, H. R. 1983, "The HELP system", Journal of Medical Systems, vol. 7, pp. 87-101.

Schecke, T., Langen, M., Popp, H.-J., Rau, G., Kasmacher, H., Kalff, G. 1992, "Knowledge-based decision support for patient monitoring in cardioanesthesia", International Journal of Clinical Monitoring and Computing, vol. 9, no. 1, pp. 1-11.

Shabot, M. M., LoBue, M., Leyerle, B. J., Dubin, S. B. 1990, "Decision support alerts for clinical laboratory and blood gas data", International Journal of Clinical Monitoring and Computing, vol. 7, no. 1, pp. 27-31.

Shachter, R. D. 1986, "Evaluating influence diagrams", Operations Research, vol. 34, no. 6, pp. 871-872.

Shortliffe, E. H. and Buchanan, B. G. 1975, "A model of inexact reasoning in medicine", Mathematical Biosciences, vol. 23, pp. 351-379.

Shortliffe, E. H. 1976, "Computer-based medical consultation: MYCIN", New York, Elsevier.

Sittig, D. F., Pace, N. L., Gardner, R. M., Beck, E., Morris, A. H. 1989, "Implementation of a computerized patient advice system using the HELP clinical information system", Computing in Biomedical Research, vol. 22, pp. 474-487.

Spiegelhalter, D. J. 1988, "Fast algorithms for probabilistic reasoning in influence diagrams with applications in genetics and expert systems", Proceedings of Conference on influence diagrams, Berkeley, California, UCB.

Stankovic, J. A., Ramamritham, K.,(eds.) 1988, "Hard real-time systems: A tutorial", Washington, DC, Computer Society Press (IEEE).

Summers, R., Carson, E. R. 1991, "Evaluation of intelligent decision aids for application in critical care medicine", Proceedings of 13th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Nagel, J. H., Smith, W. M., (eds.), Piscataway, NJ, vol. 13, no. 3, pp. 1310-1311.

Summers, R., Andreassen, S., Carson, E. R., Egeberg, J., Schroter, M. P. 1993, "A causal probabilistic model of the respiratory system", Proceedings of 15th Annual Conference of IEEE Engineering in Medicine and Biology Society, Szeto, A. Y. J., Rangayyan, R. M., (eds.), Piscataway, NJ, vol. 15, no. 1, pp. 534-535.

Summers, R., Carson, E. R., Cramp, D. G. 1993, "Ventilator Management, The Role of Knowledge-based Technology", IEEE Engineering in Medicine and Biology magazine, vol. 12, no. 4, pp. 50-58.

Sztipanovits, J., Bourne, J. R. 1985, "Architecture of intelligent medical instruments", Proceedings of 7th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Lin, J. C., Feinberg, B. N., (eds.), Piscataway, NJ, vol. 7, no. 3, pp. 1132-1136.

Sztipanovits, J., *et al.* 1987, "Co-operative systems for real-time process monitoring and process diagnostics", Proceedings of Artificial Intelligence 87 JAPAN, Osaka, Japan, pp. 419-416.

Sztipanovits, J., Wilkes, M. 1988, "Structurally adaptive signal processing systems", Presented at the Digital Signal Processing Workshop, South Lake Tahoe, CA.

Sztipanovits, J. 1989, "Intelligent instruments", Measurement, vol. 7, no. 3, pp. 98-108.

Sztipanovits, J., *et al.* 1990, "Intelligent monitoring and diagnostics for plant automation", Proceedings of International Conference on Robotics and Automation, Los Almitos, CA, Computer Society Press, pp. 1390-1395.

Sztipanovits, J., Wilkes, D. M., Karsai, G., Biegl, C., Lynd, L. E. 1993, "The multigraph and structural adaptivity", IEEE Transactions on Signal Processing, vol. 41, no. 8, pp. 2695-2716.

Taylor, R. J. 1994, "Computer design and evaluation of recursive digital filters for use with physiological data", Medical & Biological Engineering & Computing, vol. 32, pp. 112-115.

Thull, B., Popp, H.-J., Rau, G., Schmitz, E., Hanrath, P., Effert, S. 1992, "Design of a PDMS to support a problem-oriented evaluation of data in an intensive care unit", Abstracts of Annual meeting of European Society for Computing and Technology in Anaesthesia and Intensive Care (ESCTAIC-92), p. A3.

Tong, D. A. 1991, "Weaning patients from mechanical ventilation, a knowledge-based system approach", Computer Methods and Programs in Biomedicine, vol. 35, pp. 267-278.

Uckun, S., Dawant, B. M. 1992, "Qualitative modeling as a paradigm for diagnosis and prediction in critical care environments", Artificial Intelligence in Medicine, vol. 4, no. 2, pp. 127-144.

Uckun, S., Dawant, B. M., Lindstrom, D. P. 1993, "Model-based diagnosis in intensive care monitoring: The YAQ approach", Artificial Intelligence in Medicine, vol. 5, no. 1, pp. 31-48.

Van der Aa, J. J. 1990, "Intelligent alarms in anesthesia", Ph. D. thesis, Eindhoven University of Technology, Netherlands.

Van Melle, W. 1981, "System aids in constructing consultation programs", Ann Arbor, MI, UMI Research Press.

Viot, G. 1993, "Fuzzy logic in C", Dr. Dobb's Journal, vol. 19, no. 2, pp. 40-49.

Waknis, P. 1993, "Hard real-time scheduling in model-based programming environment", Ph.D. Dissertation, Department of Electrical Engineering, Vanderbilt University, Nashville, TN.

Wardle, E. N. 1984, "Shock lungs, the post traumatic respiratory syndrome", Quarterly Journal of Medicine, vol. 53, pp. 517-529.

Washington, R., Hayes-Roth, B. 1989, "Input data management in real-time AI systems", Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, MI, Morgan Kaufmann, pp. 250-255.

Wilkes, M., Lynd, L., Sztipanovits, J., Karsai, G. 1990, "The multigraph approach to parallel, distributed, structurally adaptive signal processing", Proceedings of the 1990 International Conference on Acoustics Speech and Signal Processing, Albuquerque, NM, pp. 2037-2040.

Ying H., Sheppard L. C., Tucker D. M. 1988, "Expert system-based fuzzy control of arterial pressure by drug infusion", Medical Progress through Technology, vol. 13, pp. 202-215.

Ying H. and Sheppard L. C. 1989, "Real-time expert-system-based fuzzy control of mean arterial pressure in pigs with sodium nitroprusside infusion", Medical Progress through Technology, vol. 16, pp. 69-76.

Ying H., McEachern, M., Eddleman, D. W., Sheppard, L. C. 1992, "Fuzzy control of mean arterial blood pressure in postsurgical patients with sodium nitroprusside infusion", IEEE Transactions on Biomedical Engineering, vol. 39, pp. 1060-1070.

Ying H. and Sheppard L. C. 1994, "Regulating Mean Arterial Pressure in Postsurgical cardiac Patients", IEEE Engineering in Medicine and Biology magazine, vol. 13, no. 5, pp. 671-677.

Zadeh, L. 1965, "Fuzzy sets", Information and Control, vol. 8, pp. 338-353.

Zadeh, L. 1983, "The role of fuzzy logic in the management of uncertainty in expert systems", Fuzzy sets and systems, vol. 11, pp. 199-227.

APPENDIX A

LINEAR FILTERS

The purpose of this appendix is to provide the necessary background for understanding the implementation of the generic filters as described in chapter 5. As such it is only a brief overview. A comprehensive treatment of the subject may be found in (Proakis & Manolakis, 1989).

A linear time invariant discrete time system may be represented by the general linear constant coefficient difference equation

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k). \qquad \text{(Eq.A.1)}$$

If $N = 0$ the system is called **non-recursive** or *Finite Impulse Response* (FIR). For the case where $N \neq 0$ the system is called **recursive** or *Infinite Impulse Response* (IIR). Alternatively the system may be represented by its transfer function $\mathbf{H}(z)$, using the z transform notation,

$$\mathbf{H}(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}. \qquad \text{(Eq.A.2)}$$

The frequency response of the system, $\mathbf{H}(\omega)$, where $\omega = 2\pi f$, is the transfer function evaluated on the unit circle

$$\mathbf{H}(\omega) = \mathbf{H}(z)\Big|_{z=e^{j\omega}}. \qquad \text{(Eq.A.3)}$$

So

$$\mathbf{H}(\omega) = \frac{\sum_{k=0}^{M} b_k e^{-jk\Delta\omega}}{1 + \sum_{k=1}^{N} a_k e^{-jk\Delta\omega}} \qquad \text{(Eq.A.4)}$$

where Δ is the sampling period.

To design a filter we need to obtain a suitable set of coefficients *i.e.* as and bs, from a desired $\mathbf{H}(\omega)$. Many different techniques and their software implementations exist, such as the Remez Exchange Algorithm for FIRs or the bilinear transformation method for IIRs.

Recursive filters have a superior performance (*i.e.* sharper edges) compared with non-recursive filters.

Software Realisation Of IIRs

The transfer function of IIRs can be partitioned into two transfer functions for the numerator and the denominator,

$$\mathbf{H}(z) = \mathbf{H}_1(z)\mathbf{H}_2(z) \qquad \text{(Eq.A.5)}$$

where

$$\mathbf{H}_1(z) = \sum_{k=0}^{M} b_k z^{-k} \qquad \text{(Eq.A.6)}$$

and

$$\mathbf{H}_2(z) = \frac{1}{1 + \sum_{k=1}^{N} a_k z^{-k}}. \qquad \text{(Eq.A.7)}$$

Therefore the filter can be realised in two forms, either as an all zero filter followed by an all pole filter (*i.e.* form I) or an all pole filter $\mathbf{H}_2(z)$, feeding into the all zero filter $\mathbf{H}_1(z)$ (*i.e.* form II). In form II the output of the all pole filter forms a set of intermediate values, ws, hence reducing the storage requirements.

$$w(n) = -\sum_{k=1}^{N} a_k w(n-k) + x(n) \qquad \text{(Eq.A.8)}$$

The output of the filter will be

$$y(n) = \sum_{k=0}^{M} b_k w(n-k). \qquad \text{(Eq.A.9)}$$

The above two equations make up the Direct Form II realisation of IIR (IIRDF2). The transfer function can also be realised as K second-order transfer functions connected in series,

$$\mathbf{H}(z) = \prod_{k=1}^{K} \mathbf{H}_k(z) \qquad \text{(Eq.A.10)}$$

where K is the integer part of $(N+1)/2$. Then the implementation is known as the cascade realisation of IIR.

APPENDIX B

FUZZY KNOWLEDGE-BASE FORMAT

The fuzzy knowledge-base file format is as follows. First, input and output variables and their membership functions are defined, then the rules. There is a symbol "-->" to indicate end of inputs and beginning of outputs. Each variable is defined by a name followed by a blank line, followed by membership functions. Membership function format is a name followed by four integers that describe the trapezoid. After the output definitions there is a line starting with the keyword "SW" followed by the list of all inputs followed by the list of all outputs. Each rule is a line under this heading row. Under the column headed by the keyword "SW" is one of the keywords "ON" or "OFF". A rule starting by anything except the keyword "ON" will be ignored. Under each column of input or output a name of a membership function defined for that variable is allowed. The keyword "IG" indicates a don't care or ignore for that input or output in the rule. There must be a blank line at the end of the rules, and no blank lines in between. The following is a sample knowledge-base for the control of the Mean Arterial blood Pressure (MAP). The range of all variables is normalised to 0-255. The membership function names N, ZE, and P stand for Negative, Zero and Positive respectively. The input variables are the difference between the desired and measured MAP "error" and the first derivative of this difference "err_rate".

error

N	0	0	63	191
P	63	191	255	255

err_rate

N	0	0	63	191
P	63	191	255	255

-->

out

N	0	0	0	0
ZE	127	127	127	127
P	255	255	255	255

SW	error	err_rate	out
ON	P	P	N
ON	P	N	ZE
ON	N	P	ZE
ON	N	N	P

APPENDIX C

TEXTUAL CPN MODEL FROM THE GRAPHICAL EDITOR

```
;;; CPN database: "vent.cpn"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Cpn/")
;;; Definition for |ventilation|.
(DEF-CLUSTER-COMPONENT |ventilation| NIL ((CPN . "vent.icon"))
    ((CPN (O |Vd| |Cpn-node| 61 73 60 55)
          (O RR |Cpn-node| 123 75 122 57)
          (O |Vt| |Cpn-node| 187 76 189 57)
          (O |Vd_.| |Cpn-node| 90 181 57 164)
          (O V_. |Cpn-node| 158 182 133 163)
          (O |Valv_.| |Cpn-node| 125 284 80 265)
          (O |r3| |Cpn-add| 131 240 110 229)
          (O |r1| |Cpn-mult| 96 138 74 121)
          (O |r2| |Cpn-mult| 161 138 142 126)
          (C |CPNlink| (|Vd| NIL NIL NIL) (|r1| NIL NIL NIL) |Green| 75
             98 75 114 98 118 100 140)
          (C |CPNlink| (RR NIL NIL NIL) (|r1| NIL NIL NIL) |Green| 137
             101 137 113 110 119 110 139)
          (C |CPNlink| (RR NIL NIL NIL) (|r2| NIL NIL NIL) |Green| 143
             102 143 112 165 119 165 142)
          (C |CPNlink| (|Vt| NIL NIL NIL) (|r2| NIL NIL NIL) |Green|
             207 102 207 115 177 119 175 141)
          (C |CPNlink| (|r1| NIL NIL NIL) (|Vd_.| NIL NIL NIL) |Green|
             106 156 105 186)
          (C |CPNlink| (|r2| NIL NIL NIL) (V_. NIL NIL NIL) |Green| 172
             155 172 188)
          (C |CPNlink| (|Vd_.| NIL NIL NIL) (|r3| NIL NIL NIL) |Green|
             105 205 105 220 133 225 135 242)
          (C |CPNlink| (V_. NIL NIL NIL) (|r3| NIL NIL NIL) |Green| 173
             208 173 220 144 225 144 242)
          (C |CPNlink| (|r3| NIL NIL NIL) (|Valv_.| NIL NIL NIL)
             |Green| 140 259 139 289)
          (T |ventilation| 315 9)))
    (CPN-NODES
        (|Vd| "Dead Space" 5 ("0.12" "0.14" "0.16" "0.18" "0.20")
              (0.12000000000000001 0.14000000000000001 0.16
               0.18000000000000002 0.20000000000000001)
              (0.001 10.0 10.0 0.001))
        (RR "Respiratory Rate" 5 ("8" "10" "12" "14" "16")
            (8 10 12 14 16) (1.0 30.0 30.0 30.0 1.0))
        (|Vt| "Tidal Volume" 13
              ("0.25" "0.3" "0.35" "0.4" "0.45" "0.5" "0.55" "0.6"
               "0.65" "0.7" "0.75" "0.8" "0.85")
              (0.25 0.30000000000000004 0.35000000000000003
               0.40000000000000002 0.45000000000000001 0.5
               0.55000000000000004 0.60000000000000009
               0.65000000000000002 0.70000000000000007 0.75
               0.80000000000000004 0.85000000000000009)
              (0.001 1.0 1.0 1.0 10.0 10.0 10.0 10.0 1.0 1.0 1.0 1.0
               0.001))
        (|Vd_.| "Dead Space Ventilation" 4 ("1.0" "2.0" "3.0" "4.0")
                (1.0 2.0 3.0 4.0) NIL)
        (V_. "Total Ventilation" 11
             ("2.0" "3.0" "4.0" "5.0" "6.0" "7.0" "8.0" "9.0" "10.0"
              "12.0" "14.0")
             (2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 12.0 14.0) NIL)
        (|Valv_.| "Alveolar Ventilation" 10
            ("2.0" "3.0" "4.0" "5.0" "6.0" "7.0" "8.0" "9.0" "10.0"
             "12.0")
            (2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 12.0) NIL))
    (CPN-ADD (|r3| |Cpn-add| NIL 0.0))
    (CPN-MULT (|r1| |Cpn-mult| NIL -1.0) (|r2| |Cpn-mult| NIL 1.0))
    (CPN-NORM) (PARTS)
    (CONCS (|CPNlink| (|Vd| NIL |r1| NIL) (RR NIL |r1| NIL)
                      (RR NIL |r2| NIL) (|Vt| NIL |r2| NIL)
                      (|r1| NIL |Vd_.| NIL) (|r2| NIL V_. NIL)
                      (|Vd_.| NIL |r3| NIL) (V_. NIL |r3| NIL)
                      (|r3| NIL |Valv_.| NIL))))
```

APPENDIX D

PATIENT DATA

Patient Chart

Royal Brompton National Heart and Lung Hospital
FLOWSHEET Exact

[redacted] Patient
DOB: 29MAR1934

medRecNum: C93765
attendMD: CL

		30Sep94 1400	1415	1422	1445	1500	1530	1545	1600	1615	1700	1800
Vital Signs	Temperature Core					Ax 35.4					Ax 35.5	Ax 36.1
	Heart Rate				59	58	53	54	53		52	53
	Art Blood Pressure			117/ 57	117/ 59	124/ 60	116/ 57	118/ 59	124/ 60		115/ 55	114/ 55
	Blood Pressure Mean			77	75	82	76	79	83		75	75
	Cuff Pressure Mean											
	R Atrial Pressure			7	8	9	10	9	10		10	9
	Rhythms			SR	'S.Brady	S.Brady	S.Brady	S.Brady			S.Brady	S.Brady
Neurology	GCS Eye Response					Spont		Spont			Spont	
	GCS Motor Response					Commanc		Commanc			Command	
	GCS Verbal Response					Orientd		Orientd			Orientc	
	Glasgow Coma Score					15		15			15	
	Pain Score							10			6	
	Arm Movements					Spont		Spont				
	Hand Grip Str R/L					Wk/ Wk		Wk/ Wk				
	Leg Movements R/L					Spt/Spt		Spt/Spt				
	Leg Strength R/L					Wk/ Wk		Wk/ Wk				
	Pupil Reaction R/L					Sl/ Sl						
	Pupil Size R/L mm.					3/ 3						
Peripheral	Periph Circulation			C.Feet		C.Feet		CoolH+F			CoolH+F	CoolH+F
	Pulses D.Pedis R/L			Yes/Yes		Yes/Yes					Yes/Yes	Yes/Yes
Chronic Health	Operative Status											
	Chronic Health Pts											
Miscellaneous	Acute Renal Failure											
	Events Vital Signs											
	Care Unit											
	Documented by			SA	SA	SA	PHB	PHB	PHB		SA	SA
Ventilation	Mode			SIMV+PS		SIMV+PS			SIMV+PS		SIMV+PS	SIMV+PS
	Ventilator Type			Evita		Evita			Evita		Evita	Evita
	Airway Type			O.ETT		O.ETT			O.ETT		O.ETT	O.ETT
	FiO2			0.40		0.35			0.30	0.26	0.26	0.26
	Tidal Volume (Vent)			810		760			760		780	880
	Ventilator Set Rate			10		10			10		10	10
	Spont Resp Rate			2		1			0		2	4
	I:E Ratio			1:2		1:2			1:2		1:2	1:2
	Peak Insp Pressure			30		30			31		31	31
	PEEP			3		3			3		3	3
	Pressure Support/ASB		20			20			20		20	20
	Expired Min Volume			9.0		8.3			8.3		8.9	10.5
	Oxygen Admin Method			Vent		Vent			Vent		Vent	Vent
	Oxygen Litres/min											
	Weaning Score			60		60			60		60	60
	Humidifier Type			Edith		Edith			Edith		Edith	Edith
	Events Ventilation											
Blood Gases	O2 Saturation (P)			100		100			100		100	100
	H+			42.0		39.9			40.2		39.5	
	pH (from H+)			7.38		7.40			7.40		7.40	
	pCO2			4.9		4.7			4.7		4.5	
	pO2			21.5		17.8			17.4		16.1	
	HCO3			21		22			22		21	
	Base Excess (vt)			-2.8		-1.6			-2.0		-2.3	
	Documented by		SA	SA		SA			PHB,SA	SA	SA	SA
I	Dextrose Saline								100.0		100.0	100.0

Input/Output continued on the next page ...

Printed: 01 Oct 94 1023

0150 Evita 00-Time-t-h:min- 0,00-23,59 01-Exp. tidal volume-VTe-L-0,00-2,00 02-Breathing frequency-f-1/min- 0-60 03-MV-MV-L/min- 0,0-41,0 04-Peak-pressure-Peak-mbar- 0-99 05-Plateau-pressure-Plat-mbar- 0-99 06-PEEP-pressure-PEEP-mbar- 0-99 07-Minimum-pressure-Pmin-mbar--20-99 08-Mean-pressure-Mean-mbar- 0-99 09-Insp. O2-concentration-FIO2-%-15-99 10-Compliance-C-mL/mbar- 0-255 11-Resistance-R-mbar/(L/s)- 0-99 12-Spont. minute volume-MVs-L/min- 0,0-41,0 13-Spont. frequency-f s-1/min- 0-60 14-Airway temperature-Temp-deg C-18-45
0150 11date : 30 - 9 - 94 01O2 setting-value = 38 % 02Max. inspiratory flow = 60 L/min 03Insp. tidal volume = 0,81 L 04Frequency IPPV = 12 1/min 06I : E = 1,0 : 2,0 07Max. breathing pressure = 35 mbar 08Frequency IMV = 11,2 1/min 09PEEP / CPAP = 3 mbar 10ASB = 18 mbar 11Intern. PEEP = 18 mbar 12BIPAP_P_level_1 = 0 mbar 13BIPAP_P_level_2 = 10 mbar 14BIPAP_leveltime_1 = 10,0 s
0150,0014.30,010.79,0211,039.4,0430,0518,06 4,07—,0810,0940,10—,11—,12 0.0,13 0,14--
0150,0014.31,010.81,0211,039.1,0430,05--,06 5,07—,0810,0940,10—,11—,12 0.0,13 0,14--
0150,0014.31,010.80,0211,039.4,0430,0517,06 5,07—,0810,0939,10—,11—,12 0.0,13 0,14-- 04Frequency IPPV = 11 1/min
0150,0014.32,010.27,0212,038.8,0429,0521,06 2,07—,0810,0939,10—,11—,12 0.0,13 0,14-- 03Insp. tidal volume = 0.77 L 08Frequency IMV = 10.4 1/min 21MV low limit = 12.0 L/min
0150,0014.33,010.69,0215,03 8.8,0430,0521,06 5,07—,0811,0939,10—,11—,12 0.0,13 0,14--
0150,0014.33,010.74,0212,03 8.2,0430,0516,06 5,07—,0810,0939,10—,11—,12 0.0,13 0,14--
0150,0014.34,010.76,0210,03 8.1,0430,0516,06 4,07—,0810,0939,10—,11—,12 0.0,13 0,14--
0150,0014.34,010.76,0211,03 8.1,0430,05--,06 5,07—,0810,0939,10—,11—,12 0.0,13 0,14--
0150,0014.35,010.76,0211,03 8.1,0430,05--,06 4,07—,0810,0939,10—,11—,12 0.0,13 0,14--
0150,0014.35,010.75,0211,03 8.2,0430,0516,06 4,07—,0810,0939,10—,11—,12 0.0,13 0,14--
0150,0014.36,010.77,0210,03 8.2,0430,0516,06 5,07—,0810,0936,10—,11—,12 0.0,13 0,14-- 01O2 setting-value = 34 %
0150,0014.36,010.75,0211,03 8.2,0430,0516,06 4,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.37,010.77,0210,03 8.3,0430,0516,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.38,010.77,0211,03 8.2,0430,05--,06 4,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.38,010.77,0210,03 8.3,0430,0516,06 4,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.39,010.76,0211,03 8.2,0430,0516,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.39,010.79,0210,03 8.3,0430,0516,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.40,010.76,0211,03 8.3,0430,05--,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.40,010.76,0211,03 8.2,0430,05--,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.41,010.77,0211,03 8.2,0430,0516,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.41,010.77,0210,03 8.3,0430,0516,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.42,010.76,0211,03 8.3,0430,0516,06 4,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.42,010.79,0210,03 8.3,0430,0516,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.43,010.78,0211,03 8.3,0430,05--,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.44,010.78,0210,03 8.3,0430,0516,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.44,010.77,0211,03 8.3,0430,0517,06 4,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.45,010.78,0210,03 8.3,0430,0516,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
0150,0014.45,010.77,0211,03 8.3,0430,05--,06 5,07—,0810,0935,10—,11—,12 0.0,13 0,14--
□

APPENDIX E

LIST OF ACRONYMS

AI	Artificial Intelligence
AICU	Adult Intensive Care Unit
API	Application Programmers Interface
ARDS	Adult Respiratory Distress Syndrome
ASCII	American Standard Code for Information Interchange
ATMS	Assumption-based Truth Maintenance System
BSA	Body Surface Area
CABG	Coronary Artery Bypass Graft
CF	Certainty Factor
CICU	Cardiac Intensive Care Unit
COPD	Chronic Obstructive Pulmonary Disease
CPN	Causal Probabilistic Network
CPU	Central Processing Unit
FCIA	Faulty Component Identification Algorithm
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
GUI	Graphical User Interface
HDE	High Dependency Environment
HDL	Hierarchical Description Language
HIS	Hospital Information System
ICU	Intensive Care Unit
IEE	Institution of Electrical Engineers
IEEE	Institute of Electrical and Electronics Engineers
IIR	Infinite Impulse Response
IIRDF2	Infinite Impulse Response Direct Form realisation type 2
ILMP	InterLevel Migration Process
IMDE	Integrated Model-based Development Environment
IPC	InterProcess Communication

IPCS	Intelligent Process Control System
IPMS	Intelligent Patient Monitoring System
IS	Information System
KBS	Knowledge-Based System
KS	Knowledge Source
LGDF	Large Grain Data Flow
LOS	Length Of Stay
MA	Multigraph Architecture
MA(B)P	Mean Arterial (Blood) Pressure
MCD	Monitoring Control and Diagnostic
MEE	Multigraph Execution Environment
MGK	Multigraph Kernel
MIM	Measurement and Information in Medicine
MPE	Multigraph Programming Environment
NP	problems that can be solved by Nondeterministic algorithms in Polynomial time
OO	Object Oriented
OR	Operating Room
PC	Personal Computer
PDMS	Patient Data Management System
QPT	Qualitative Process Theory
RBH	Royal Brompton Hospital
RISC	Reduced Instruction Set Computer
SNP	Sodium NitroPrusside
SQL	Structured Query Language

APPENDIX F

SELECTED MODEL AND CODE LISTINGS

Contents

F.1 Actor Script For IIRs

The following code segment (script) implements the Infinite Impulse Response filter in time domain as a computational block (actor) for the multigraph computational model (section 5.2).

```
struct Cntxt1 {
        double  a1[15];                         /* as and bs filter coefficients      */
        double  a2[15];
        double  b1[15];
        double  b2[15];
        double  x1[15];                         /* xs filter weights                  */
        double  x2[15];
        int     initial;
        int     filter_order;
};

void gr_2ord_script(cntx)                       /* generic IIR filter_script          */
struct Cntxt1 *cntx;
{

        double temp,rawdata,filtdata;
        int i;


        if(!cntx->initial) {                    /* first time into script             */
          for(i=1;i<=cntx->filter_order;i++) {   /* initialise weights to zero         */
            cntx->x1[i]=0.0;
            cntx->x2[i]=0.0;
          }
          cntx->initial=1;
        }
        rawdata=mgk_d_receive(0);               /* read data just arrived             */
        for(i=1;i<=cntx->filter_order;i++) {     /* compute filter output              */
          temp=rawdata-(cntx->b1[i]*cntx->x1[i])-(cntx->b2[i]*cntx->x2[i]);
          filtdata=temp+(cntx->a1[i]*cntx->x1[i])+(cntx->a2[i]*cntx->x2[i]);
          cntx->x2[i]=cntx->x1[i];              /* shift stored data                  */
          cntx->x1[i]=temp;
          rawdata=filtdata;
        }
        mgk_d_propagate(0,filtdata);            /* push out result                    */
}                                               /* END gr_2ord_script                 */
```

F.2 Actor Script For Median Filter

The following multigraph script implements a median filter. The output of a median filter is the median of the last n data points, where n is the length of the data window (section 5.2).

```
#include <math.h>

#define DATA_WIN  21                        /* upper limit of data window      */

struct Cntxt2 {
        double   rd[DATA_WIN+1];
        double   std[DATA_WIN];
        int      n;                         /* data window; must be odd        */
};


void gr_med_script(cntx)                    /* generic median filter script    */
struct Cntxt2 *cntx;
{
        int i;
        double *rawdata,*sorted,*tempa,temp[(cntx->n)+1];


        rawdata=(cntx->rd)-1;               /* unit-offset arrays              */
        sorted=(cntx->std)-1;
        tempa=temp-1;

        i=cntx->n;
        while(i) {                          /* shift out old datum             */
         rawdata[i+1]=rawdata[i];
         i--;
        }
        rawdata[1]=mgk_d_receive(0);        /* shift in new datum              */
        for (i=1;i<=(cntx->n);i++) tempa[i]=sorted[i];      /* copy sorted data array  */
        --i;
        while(i> 0 && tempa[i] >  rawdata[1]) {   /* look for place to insert new  */
         tempa[i+1]=tempa[i];
         i--;
        }
        tempa[i+1]=rawdata[1];              /* insert it here                  */
        i=1;
        while( tempa[i] != rawdata[(cntx->n)+1] ) {/* copy until old datum      */
         sorted[i]=tempa[i];
         i++;
```

```
        }
        i++;                                    /* skip old datum          */
        while(i <=(cntx->n)+1) {                /* copy rest               */
         sorted[i-1]=tempa[i];
         i++;
        }
        mgk_d_propagate(0,sorted[((cntx->n)+1)/2]);     /* push out result     */
}                                               /* END gr_med_script       */
```

F.3 Fuzzy Logic Library

This library contains all the functions necessary to implement a simple fuzzy logic inference engine with typical applications in fuzzy control. Included is also a parser (initialize_system) to read and convert a text file containing the rules into the knowledge-base (section 5.3).

```
/*********************************************************
*
* fuzz1.c          for borlandc
*
* RNP              07/04/94
*
* Fuzzy inference for any number of rules with any number
* of antecedents and any number of consequences.
* Based on Greg Viot's fuzzy system -- DDJFEB93 & DDJAPR94
*
*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXNAME                 10              /* max length of variable names    */
#define UPPER_LIMIT             255             /* max value a vriable can take    */

/* rule parser return values */
#define RR_OK                   0
#define RR_SHORT_FILE           (-1)
#define RR_SHORT_LINE           (-2)
#define RR_BAD_TRAPEZOID        (-3)
#define RR_UNDEFINED_KEYWORD    (-4)

/* read line return values */
#define ELS_ERR                 (-1)
#define ELS_OK                  0

#define MAX_LINE_LENGTH     80

struct io_type *System_Inputs;                  /* anchor inputs                   */
struct io_type *System_Outputs;                 /* anchor output                   */

struct io_type{                                 /* input/output data type          */
        char name[MAXNAME];
```

```
        int value;
        struct mf_type *membership_functions;
        struct io_type *next;
};

struct mf_type{                                 /* membership function type     */
        char name[MAXNAME];
        int value;                              /* height of trapezoid          */
        int point1;                             /* left most point              */
        int point2;                             /* right most point             */
        float slope1;                           /* left side slope              */
        float slope2;                           /* right side slope             */
        struct mf_type *next;
};
```

```
struct rule_type{                                   /* rule type                    */
        struct rule_element_type *if_side;
        struct rule_element_type *then_side;
        struct rule_type *next;
};

struct rule_element_type{                           /* rule element type            */
        int *value;
        struct rule_element_type *next;
};

struct rule_type *Rule_Base;

void    fuzzification(void);
int     rule_evaluation(void);
void    defuzzification(void);
void    compute_degree_of_membership(struct mf_type *mf,int input);
float   compute_area_of_trapezoid(struct mf_type *mf);
void    put_system_outputs(void);
void    get_system_inputs(int input1, int input2);
int     initialize_system(FILE *fp);
int     EmptyLineSkip(char *Line, FILE *fp);
```

```
void fuzzification(void)
{
  struct io_type *si;
  struct mf_type *mf;

  for(si=System_Inputs;si!=NULL;si=si->next)        /* for each input                    */
    for(mf=si->membership_functions;mf!=NULL;mf=mf->next)
                                                    /* for each membership function      */
          compute_degree_of_membership(mf,si->value);
                                                    /* compute degree of membership      */
  return;
}                                                   /* END FUZZIFICATION */
```

```
int rule_evaluation(void)
{
  struct rule_type *rule;
  struct rule_element_type *ip;                    /* if pointer                    */
  struct rule_element_type *tp;                    /* then pointer                  */
  int strength;
  int vrules=0;
  int match=0;                                     /* test some rules               */

  for(rule=Rule_Base;rule!=NULL;rule=rule->next) /* for all rules in knowledge base */
  {
   strength=UPPER_LIMIT;
   for(ip=rule->if_side;ip!=NULL;ip=ip->next)      /* for all antecedents           */
         strength=min(strength,*(ip->value));
   for(tp=rule->then_side;tp!=NULL;tp=tp->next)    /* for all consequences          */
   {
         *(tp->value)=max(strength,*(tp->value));  /* calculate strength            */
         if(strength>0)match=1;                    /* some rules matched            */
   }
   if(match)vrules++;                              /* increment no. of valid rules  */
   match=0;
  }
  return(vrules);
}                                                  /* END RULE EVALUATION           */
```

```
void defuzzification(void)
{
  struct io_type *so;
  struct mf_type *mf;
  long   sum_of_products;
  long   sum_of_areas;
  long   area, centroid;

  for(so=System_Outputs;so!=NULL;so=so->next)   /* for all outputs                  */
  {
   sum_of_products=0;
   sum_of_areas=0;
   for(mf=so->membership_functions;mf!=NULL;mf=mf->next)
                                                /* for all membership functions     */
   {
    area=compute_area_of_trapezoid(mf);         /* compute area of trapezoid        */
    centroid=mf->point1+(mf->point2-mf->point1)/2;
                                                /* compute centroid of trapezoid    */
    sum_of_products+=area*centroid;
    sum_of_areas+=area;
   }
   if(sum_of_areas==0)
   {
    printf("For %s Sum of Areas = 0, will cause div error\n",so->name);
    printf("For %s Sum of Products= %d\n",so->name,sum_of_products);
    so->value=0;
   }
   else
   so->value=sum_of_products/sum_of_areas;      /* overall centroid                 */
  }
  return;
}                                               /* END DEFUZZIFICATION              */
```

```
void compute_degree_of_membership(struct mf_type *mf,int input)
{
  int delta_1, delta_2;
  delta_1=input - mf->point1;                    /* distance to left most point      */
  delta_2=mf->point2 - input;                    /* distance to right most point     */

  if((delta_1<=0)||(delta_2<=0))mf->value=0;
  else
  {
   mf->value=min((mf->slope1*delta_1),(mf->slope2*delta_2));
   mf->value=min(mf->value,UPPER_LIMIT);
  }
  return;
}                                          /* END DEGREE OF MEMBERSHIP         */
```

```
float compute_area_of_trapezoid(struct mf_type *mf)
{
  float run_1,run_2,area,top;
  float base;

  base=mf->point2 - mf->point1;
  run_1=mf->value / mf->slope1;
  run_2=mf->value / mf->slope2;
  top=base - run_1 - run_2;
  area=mf->value*(base+top)/2;
  return(area);
}                                              /* END AREA OF TRAPEZOID   */
```

```
void put_system_outputs(void)
{
  struct io_type *ioptr;
  struct mf_type *mfptr;
  struct rule_type *ruleptr;
  struct rule_element_type *ifptr;
  struct rule_element_type *thenptr;
  int cnt=1;

  for(ioptr=System_Inputs;ioptr!=NULL;ioptr=ioptr->next)   /* for all inputs              */
  {
    printf("%s: Value= %d\n",ioptr->name,ioptr->value);     /* print name and value   */
    for(mfptr=ioptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
                                                          /* for all membership functions    */
    {
         printf("  %s: Value %d Left %d Right %d\n",
           mfptr->name,mfptr->value,mfptr->point1,mfptr->point2);
                                                 /* print name, value, left and right points   */
    }
   printf("\n");
  }
  for(ioptr=System_Outputs;ioptr!=NULL;ioptr=ioptr->next)
                                                          /* for all outputs                 */
  {
    printf("%s: Value= %d\n",ioptr->name,ioptr->value);     /* print name and value   */
    for(mfptr=ioptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
                                                          /* for all membership functions    */
    {
         printf("  %s: Value %d Left %d Right %d\n",
           mfptr->name,mfptr->value,mfptr->point1,mfptr->point2);
                                                 /* print name, value, left and right points   */
    }
   printf("\n");
  }
                                    /* print values pointed to by rule_type (if & then)    */
  for(ruleptr=Rule_Base;ruleptr->next!=NULL;ruleptr=ruleptr->next)
  {
   printf("Rule #%d:",cnt++);
   for(ifptr=ruleptr->if_side;ifptr!=NULL;ifptr=ifptr->next)
    printf("  %d",*(ifptr->value));
   for(thenptr=ruleptr->then_side;thenptr!=NULL;thenptr=thenptr->next)
    printf("  %d",*(thenptr->value));
   printf("\n");
  }
  return;
}                                                  /* END PUT SYSTEM OUTPUTS */
```

```
void get_system_inputs(int input1, int input2)
{
  struct io_type *ioptr;

  ioptr=System_Inputs;
  ioptr->value=input1;
  ioptr=ioptr->next;
  ioptr->value=input2;
  return;
}                                          /* END GET SYSTEM INPUTS   */
```

```
int EmptyLineSkip(char *Line, FILE *fp)
{
    do
        if (fgets(Line, MAX_LINE_LENGTH + 1, fp) == NULL)
            return (ELS_ERR);
    while (sscanf(Line, "%*s") == EOF);
    return (ELS_OK);
}                                                   /* END EmptyLineSkip            */
```

```
int initialize_system(FILE *fp)
{
  char   Line[MAX_LINE_LENGTH + 1], *Tok;
  char   buff[MAXNAME];
  int    i, j;
  int    a,b,c,d;
  int  firstin;

  struct io_type *outptr;
  struct mf_type *top_mf;
  struct mf_type *mfptr;
  struct io_type *ioptr;
  struct rule_type *ruleptr;
  struct rule_element_type *ifptr;
  struct rule_element_type *top_if;
  struct rule_element_type *thenptr;
  struct rule_element_type *top_then;

  ioptr=NULL;
  ruleptr=NULL;
  ifptr=NULL;
  top_if=NULL;
  thenptr=NULL;
  top_then=NULL;

/***********************************************
 * read input variable definitions
 ***********************************************/
  if (EmptyLineSkip(Line, fp) == ELS_ERR)
        return (RR_SHORT_FILE);                 /* if nothing to read stop          */
  for (i = 0; ;i++)
   {
    sscanf(Line, "%s", buff);
    if(strcmp(buff,"-->")==0)break;    /* symbol "-->" indicates end of input definitions */
                                                /* get storage for input variable   */
    if(ioptr==NULL)
    {
        ioptr=(struct io_type *)calloc(1,sizeof(struct io_type));
        System_Inputs=ioptr;
    }
    else
    {
        for(ioptr=System_Inputs;ioptr->next;ioptr=ioptr->next);
        ioptr->next=(struct  io_type *)calloc(1,sizeof(struct io_type));
        ioptr=ioptr->next;
    }
    sprintf(ioptr->name,"%s", buff);            /* save name of the variable        */
    if (EmptyLineSkip(Line, fp) == ELS_ERR)
        return (RR_SHORT_FILE);
    mfptr=NULL;
    for (j = 0; ;j++)
```

```
    {                                       /* read the membership function defenitions*/
        if (sscanf(Line, "%*s") == EOF)
         break;                             /*if blank line come out of this loop      */
        if (sscanf(Line, "%s %d %d %d %d", buff,
                &a, &b, &c, &d) < 5) return (RR_SHORT_LINE);
                                            /* get storage for membership function     */
        if(mfptr==NULL)
        {
         mfptr=(struct mf_type *)calloc(1,sizeof(struct mf_type));
         top_mf=mfptr;
         ioptr->membership_functions=mfptr;
        }
        else
        {
         for(mfptr=top_mf;mfptr->next;mfptr=mfptr->next);
         mfptr->next=(struct mf_type *)calloc(1,sizeof(struct mf_type));
         mfptr=mfptr->next;
        }
        sprintf(mfptr->name,"%s",buff);             /* save membership function name */
                                                    /* save the trapezoid definition  */
        mfptr->point1=a;
        mfptr->point2=d;
        if(b-a>0) mfptr->slope1=UPPER_LIMIT/(b-a);
        else  return(RR_BAD_TRAPEZOID);
         /*{
           printf("Error- variable \"%s\" membership element \"%s\".\n", ioptr->name,mfptr->name);
           exit(1);
           }*/
        if(d-c>0) mfptr->slope2=UPPER_LIMIT/(d-c);
        else  return(RR_BAD_TRAPEZOID);
        if (fgets(Line, MAX_LINE_LENGTH + 1, fp) == NULL)
         return(RR_SHORT_FILE) ;                    /*if no more lines stop          */
    }
    if (EmptyLineSkip(Line, fp) == ELS_ERR)
        return (RR_SHORT_FILE);
   }
/* ***********************************************
 * read output variable definitions
 ***********************************************/
   ioptr=NULL;
   if (EmptyLineSkip(Line, fp) == ELS_ERR)
     return (RR_SHORT_FILE);                    /* if nothing to read stop          */
for (i = 0; ;i++)
{
    sscanf(Line, "%s", buff);
    if(strcmp(buff,"SW")==0)break;          /* symbol "SW" indicates end of output definitions  */
                                            /* get storage for output variable   */
if(ioptr==NULL)
    {
        ioptr=(struct io_type *)calloc(1,sizeof(struct io_type));
        System_Outputs=ioptr;
    }
```

```
    else
    {
        for(ioptr=System_Outputs;ioptr->next;ioptr=ioptr->next);
        ioptr->next=(struct  io_type *)calloc(1,sizeof(struct io_type));
        ioptr=ioptr->next;
    }
    sprintf(ioptr->name,"%s", buff);                    /* save name of the variable        */
    if (EmptyLineSkip(Line, fp) == ELS_ERR)
        return (RR_SHORT_FILE);
    mfptr=NULL;
    for (j = 0; ;j++)
          {                                             /* read the membership function defenitions*/
           if (sscanf(Line, "%*s") == EOF)
                 break;                                 /*if blank line come out of this loop       */
           if (sscanf(Line, "%s %d %d %d %d", buff,
                 &a, &b, &c, &d) < 5) return (RR_SHORT_LINE);
                                                        /* get storage for membership function      */
           if(mfptr==NULL)
           {
            mfptr=(struct mf_type *)calloc(1,sizeof(struct mf_type));
            top_mf=mfptr;
            ioptr->membership_functions=mfptr;
           }
           else
           {
            for(mfptr=top_mf;mfptr->next;mfptr=mfptr->next);
            mfptr->next=(struct mf_type *)calloc(1,sizeof(struct mf_type));
            mfptr=mfptr->next;
           }
           sprintf(mfptr->name,"%s",buff);              /* save membership function name */
                                                        /* save the trapezoid definition    */
           mfptr->point1=a;
           mfptr->point2=d;
           if(b-a>0) mfptr->slope1=UPPER_LIMIT/(b-a);
           else return(RR_BAD_TRAPEZOID);
           if(d-c>0) mfptr->slope2=UPPER_LIMIT/(d-c);
           else return(RR_BAD_TRAPEZOID);
           if (fgets(Line, MAX_LINE_LENGTH + 1, fp) == NULL)
                 return(RR_SHORT_FILE) ;                /*if no more lines stop             */
          }
          if (EmptyLineSkip(Line, fp) == ELS_ERR)
                 return (RR_SHORT_FILE);
    }
/*************************************************
 * read rules heading
 *************************************************/
    strtok(Line, "\n \t");
    for (ioptr=System_Inputs;ioptr!=NULL ;ioptr=ioptr->next)
                                                        /* for all inputs, check input names*/
     {
         Tok = strtok(NULL, "\n \t");
         if (strcmp(Tok,ioptr->name) != 0)
```

```
                return (RR_UNDEFINED_KEYWORD);
    }
  for(ioptr=System_Outputs;ioptr!=NULL ;ioptr=ioptr->next)
                                        /* for all outputs, check output names*/
{
        Tok = strtok(NULL, "\n \t");
        if (strcmp(Tok,ioptr->name) != 0)
                return (RR_UNDEFINED_KEYWORD);
    }
/************************************************
 * read rules
 ************************************************/
  if (EmptyLineSkip(Line, fp) == ELS_ERR)
        return (RR_SHORT_FILE);
  ruleptr=(struct rule_type *)calloc(1,sizeof (struct rule_type));
  Rule_Base=ruleptr;
  for (i = 0; ;I++)                     /* for all rules                */
  {
        firstin=0;
        sscanf(Line, "%s",buff);
        if(strcmp(buff,"ON")!=0)        /* if a rule is not "ON"        */
        {
         if (fgets(Line, MAX_LINE_LENGTH + 1, fp) == NULL)
            break;                      /* get a new line               */
         continue;                      /* skip back to top of this loop */
        }
        strtok(Line, "\n \t");
        for(ioptr=System_Inputs;ioptr!=NULL;ioptr=ioptr->next)
                                        /* for all inputs               */
        {
        Tok=strtok(NULL, "\n \t");      /* get a token                  */
        if(strcmp(Tok,"IG")==0) {continue;}     /* ignore this antecedent */
        for (mfptr=ioptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
                        /* check this token against all membership function names */
        {
        if(strcmp(mfptr->name,Tok)==0)
                                        /* get storage for rule element */
         {
         if(firstin==0)
         {
          ifptr=(struct rule_element_type *)calloc(1,sizeof(struct rule_element_type));
          top_if=ifptr;
          ruleptr->if_side=ifptr;
          firstin=1;
         }
         else
         {
          for(ifptr=top_if;ifptr->next;ifptr=ifptr->next);
          ifptr->next=(struct rule_element_type *)calloc(1,sizeof(struct rule_element_type));
          ifptr=ifptr->next;
         }
                                        /* save if side rule element    */
```

```
            ifptr->value=&mfptr->value;
            break;
           }
          }
         }
         firstin=0;
         for(ioptr=System_Outputs;ioptr!=NULL;ioptr=ioptr->next)
                                                  /* for all outputs                */
         {
          Tok=strtok(NULL, "\n \t");              /* get a token                    */
          if(strcmp(Tok,"IG")==0)  {continue;}
                          /* check this token against all membership function names  */
          for (mfptr=ioptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
          {
           if(strcmp(mfptr->name,Tok)==0)
                                                  /* get storage for rule element   */
           {
            if(firstin==0)
            {
              thenptr=(struct rule_element_type *)calloc(1,sizeof(struct rule_element_type));
              top_then=thenptr;
              ruleptr->then_side=thenptr;
              firstin=1;
            }
            else
            {
             for(thenptr=top_then;thenptr->next;thenptr=thenptr->next);
             thenptr->next=(struct rule_element_type *)calloc(1,sizeof(struct rule_element_type));
             thenptr=thenptr->next;
            }
                                                  /* save then side rule element    */
            thenptr->value=&mfptr->value;
            break;
           }
          }
         }
         if (fgets(Line, MAX_LINE_LENGTH + 1, fp) == NULL)
         break;
                                                  /* get storage for next rule      */
         for(ruleptr=Rule_Base;ruleptr->next;ruleptr=ruleptr->next);
         ruleptr->next=(struct rule_type *)calloc(1,sizeof(struct rule_type));
         ruleptr=ruleptr->next;
    }
    return (RR_OK);
}                                                 /* END initialize_system          */
```

F.4 CPN Model Editor

The Cpn.edf file defines the objects and their attributes as well as the allowed connections among them for the graphical model builder. It also defines the model database and database functions for storing and retrieving the CPN models during a graphical editing session (section 5.4.2).

/home/vandy/Xgem/def/Cpn.edf

```
;;; -*-Lisp-*-
;;; Cpn.edf
;;;
;;; ************************************************************************
;;;       UTILITY FUNCTIONS
;;;
;;; read_list function

(defun read-list (pvar type flag &rest attrs)
        (mapcar #'(lambda (pair)
                (if (listp pair)
                        (list* (car pair) type flag
                                (mapcar #' (lambda (a v) (cons a v))
                                        attrs (cdr pair)))
                        (list pair type flag)))
                pvar))

;;; write_list function

(defun write-list (pl flag &rest attrs)
        (let ((res nil))
                (mapc #'lambda (p)
                        (if (and (caddr p) (eq (caddr p) flag))
                        (push
                        (if (null attrs) (car p)
                                (cons (car p)
                                        (mapcar #'(lambda (aname)
                                                (cdr (assoc aname (cdddr p))))
                                                        attrs)))
                                res)))
                pl)
        (nreverse res)))
```

```
;;; ****************************************************************
;;; CPN objects
;;;
;;;

;;; cpn_node primitive object definition

(def-implicit |Cpn-node| "cpn0.icon"
        (remarks (page-s "Enter remarks for cpn-node:"))
        (|CpnStateN| (value "Enter number of states for cpn-node:"))
        (|CpnStateL| (value "Enter state name list for cpn-node:"))
        (|CpnStateD| (value "Enter state distribution table for cpn-node:")))

;;; cpn_relation primitive object definition

(def-implicit |Cpn-rel| "cpn3.icon"
        (remarks (page-s "Enter remarks for cpn-relation:")))

;;; Cluster Component compound object definition
(def-rep |Cluster Component|
        (var name icon pict remarks cpnn cpn-rel parts concs)
        (name name) (icon icon) (pict pict)
        (views
                (|CPN|
                        (iconp t)
                        (title "9x15" |White|)
                        (object "9x15" |White|)
                        (drawp nil)
                        (links
                        (cpnn |Cpn-node| |Cpn-node| cpn-link-read pn-link-write))
                        (attrs
                        (remarks (page-s "Enter remarks for cluster Component:")))
                        (conns (|CPNlink| (solid 1 line arrow)
                                ((|Cpn-node|) nil (|Cpn-rel|) nil)
                                ((|Cpn-rel|) nil (|Cpn-node|) nil)
                                ((|Cpn-rel|) nil (|Cluster Component|) (|Cpn-node|))
                                ((|Cluster Component|) (|Cpn-node) (|Cpn-rel|) nil)))
                        (struct (parts cpn-rel concs)
                        (|Cluster Component| |Cpn-node|)
                        cpn-struct-get cpn-struct-make)))
                (form (def-cluster-component ?name ?remarks ?icon ?pict
                        (cpn-nodes >cpnn)
                        (cpn-rel >cpn-rel)
                        (parts >parts)
                        (concs >concs))))

;;;*****************************************
;;;
;;; Editor read/write functions
;;;
;;;

(defun cpn-link-read (pvar)
        (read-list pvar '|Cpn-node| '|Cpn-node| 'remarks '|CpnStateN|
```

```
                '|CpnStateL| '|CpnStateD|))

(defun cpn-link-write (pv)
        (write-list pv '|Cpn-node| 'remarks '|CpnStateN| '|CpnStateL| '|CpnStateD|))

(defun-cpn-struct-get (parts cpn rel concs)
        (nconc (mapcar #'(lambda (part) (list (car part) (cadr part) :item
                                        (cons 'remarks (caddr part)))) parts)
                (mapcar #'(lambda (cpn-rel)
                        ;;; This should GO AWAY!
                        (if (symbolp cpn-rel) (setq cpn-rel (list cpn-rel "")))
                        (let* (
                                (name nth 0 cpn-rel))
                                rem nth 2 cpn-rel)))
                            (list name '|Cpn-rel| :implicit
                            (cons 'remarks rem))))
                        cpn-rel)))

(defun cpn-struct-make (parts conns links convs)
        (list
                (mapcan #'(lambda (p) (cond ((equal (cadr p) '|Cpn-rel|) () )
                                        ((equal (cadr p) '|Cpn-node|) () )
                                        (t (list (cons (car p)
                                                (list (cadr p)
                                                        (list cdr (assoc 'remarks (cddr p)))))))))
                                parts)
                        conns))

;;; ***********************************************
;;;
;;; Model Database:
;;;

(defvar *cpn-dbase-dir* #"")

(defvar *valids* '(def-cluster-component))

(def-database cpn
        (select select-cpn-database)
        (list list-cpn-database)
        (input input-cpn-database)
        (output output-cpn-database))

;;;***********************************************
;;;
;;; Database functions
;;;

;;; function to set the cpn database directory
(defun select-cpn-database (name)
        (if (check-file name)
        (setq *cpn-dbase-dir* (truename name))
        nil))
```

```
;;; function to get the list of cpn model files
(defun list-cpn-database ()
        (let ((list (directory (merge-pathnames *cpn-dbae-dir* ".cpn"))))
        (mapcar #'file-namestring list)))

;;; function to read a cpn model into the editor
(defun input-cpn-dbase (arg)
        (prog (infile)
              (typecase arg
                        (string
                                (setq infile (open (merge-pathnames *cpn-dbase-dir* arg)
                                                   :if-does-not-exist nil))
                                (when (null infile)
                                        (warning "Can't input CPN database:~S" arg)
                                        (return nil)))
                        (stream (setq infile arg))
                        (t (warning "Unknown arg type for ~S." arg)
                                (return nil)))
              (let* ((eof '(())) (valids *valids*))
                (return
                        (do* ((form (read infile nil eof) (read infile nil eof))
                              (eofp (eq form eof) (eq form eof)
                              (good (and (listp form) (member (car form) valids))
                                    (and (listp form) member (car form) valids))))
                             ((or eofp good)
                              (if eofp (progn (close infile) (return nil))
                                (return (values form infile)))))))))

;;; function to write to a cpn model file
(defun output-cpn-database (form &optional file)
   (prog (outfile)
        (typecase file
                (string
                        (setq outfile
                                (open (merge-pathnames file
                                        (merge-pathnames *cpn-dbase-dir* ".cpn"))
                                      :direction :output))
                        (format outfile ";;; CPN database: ~S~%" file)
                        (format outfile "~S~%" (get-icondir)))
                (stream (setq outfile file))
                (t (warning "Unknown arg type for ~S." file)
                   (return nil)))
        (if (null form) (close outfile)
          (progn
                (format outfile ";;; Definition for ~S." (cadr form))
                ((pprint form outfile)
                (terpri outfile)
                (return outfile)))))
```

F.5 CPN Model Translator

The CPN model translator is implemented in two files, a shell script (cpn-compile-script) and a lisp program (tr.lsp) (section 5.4.3).

/home/vandy/Designs/Resp2/Cpn/cpn_compile_script

This script is run by IPCS initialisation to compile the CPN model files for use by xhugin.

```
#
# cpn_compile_script
#
# RNP            16/03/94
#
# This is the glue for "xgem" and "IPCS" to "xhugin".
# It first runs "tr" to translate xgem models (*.cpn)  to node files (*.nod)
# and concats resulting files to create xhugin input format network files
# (*.net) and then runs cpn-comp on cpn-network files to get knowledge-base
# files (*.kb) which are used by IPCS run-time through actor scripts which call
# xhugin engine.
#
##############################################################################

/home/vandy/xlisp/sources/xlisp tr
rm /home/vandy/Hugin/Specs/new.net.old
mv /home/vandy/Hugin/Specs/new.net /home/vandy/Hugin/Specs/new.net.old
cat *.nod > /home/vandy/Hugin/Specs/new.net
rm *.nod *.old
/home/vandy/Bin/cpn-comp resp1
```

/home/vandy/Designs/Resp2/Cpn/tr.lsp

This is a lisp program which converts the CPN model files into an intermediate format which is recognised by the cpn-comp program.

```
;;*************************************************************************
;;
;; tr.lsp
;;
;; RNP            16/03/94
;;
;; This file tr.lsp (translate-toplevel) should be
;; edited so that the expression (trans '|<filename>| 0 0 0 0)
;; contains the name of the toplevel cpn model file as the filename parameter.
;;
;;*************************************************************************

(load "/home/vandy/xlisp/lsp/common.lsp")

;;*************************************************************************
;; UTILITY FUNCTIONS
;;*************************************************************************

(defun get-expr (fn)
        (with-open-file (in-strm (strcat (format nil "~a" fn) ".cpn") :direction :input)
          (dotimes (n 1) (read in-strm))
          (read in-strm)))

(defun  get-list (symb al)
 (rest (find-if #'(lambda (l) (when (listp l) (eq symb (car l)))) al)))

(defun  get-lista (symb al)
 (remove-if-not #'(lambda (l) (when (listp l) (eq symb (car l)))) al))

(defun  get-listb (symb al)
 (find-if #'(lambda (l) (when (listp l) (eq symb (caddr l)))) al))

(defun  get-listc (symb al)
 (find-if #'(lambda (l) (when (listp l) (eq symb (cadr l)))) al))

(defun  get-list-pp (al)
 (remove-if #'(lambda (l) (when (listp l) (null (nth 3 l)))) al))

(defun get-concs (model)
 (get-list '|CPNlink| (get-list 'CONCS model)))

(defun get-rel (al node)
```

```
 (car (find-if #'(lambda (l) (when (listp l) (and (eq node (nth 2 l))
                                                  (eq 'nil (nth 1 l))
                                                  (eq 'nil (nth 3 l))))) al)))

(defun get-par (al rel kw)
 (let ((parl (remove-if-not #'(lambda (l) (when (listp l) (and
                    (eq rel (nth 2 l))
                    (eq 'nil (nth 3 l))))) al)))

   (cond ((eq kw 'num) (length parl))
         ((eq kw 'li)  (mapcar #'(lambda(l) (if (eq 'nil (cadr l)) (car l) (car (cadr l)))) parl)))))


(defun modify-par (elm-list concs-list)

;;; copy file to file.old
(let*  ((*file-name* (format nil "~a" (car (nth 3 elm-list))))
          (in-strm (open (strcat *file-name* ".nod") :direction :input))
          (out-strm (open (strcat *file-name* ".old") :direction :output)))
 (do  ((*str* (read-line in-strm) (read-line in-strm)))
     ((null *str*))
     (format out-strm "~&~a" *str*))
 (close in-strm)
 (close out-strm)

;;; modify parent list and write to file
 (setq in-strm (open (strcat *file-name* ".old") :direction :input))
 (setq out-strm (open (strcat *file-name* ".nod") :direction :output :if-exists :supersede))

(format out-strm "~%")
(format out-strm "~&~a" (read-line in-strm))
 (let ((*str* (read-line in-strm)))
     (format out-strm "~&~a " *str*))


 (do ((l-do (get-par concs-list (car elm-list) 'li ) (cdr l-do))
     (n 1 (1+ n)))
    ((endp l-do))
    (format out-strm "~a " (car l-do)))
 (do  ((*str* (read-line in-strm) (read-line in-strm)))
     ((null *str*))
     (format out-strm "~&~a" *str*))
 (format out-strm "~%~%")
 (close in-strm)
 (close out-strm)))
```

```
;;*************************************************************************
;;
;; trans
;; translates cpn model files (*.cpn), generated by graphical editing (xgem),
;; from the format specified in "Cpn.edf" to xhugin's (the causal probabilistic
;; inference engine) node specification format (*.nod).
;;*************************************************************************
;;

(defun trans (fn-i x-lev x-offs y-lev y-offs)
 (setq x-lev (+ x-lev x-offs))
 (setq y-lev (+ y-lev y-offs))
;;  (format t "~a ~a~&" x-lev y-lev)
 (dolist (part (get-list 'PARTS (get-expr fn-i)))
  (let* ((obj-offs-list (get-lista 'O (get-list 'CPN (nth 4 (get-expr fn-i)))))
         (name (cadr part))
         (x-offs (or (nth 3 (get-listb name obj-offs-list)) 0))
         (y-offs (or (nth 4 (get-listb name obj-offs-list)) 0)))
   (trans name x-lev x-offs y-lev y-offs)))
 (let*  ((model (get-expr fn-i))
         (concs (get-concs model)))
  (dolist (elm (get-list 'CPN-NODES model))
   (let ((rels (get-rel concs (first elm)))
         (node-offs-list (get-lista 'O (get-list 'CPN (nth 4 model))))
         (stream (open (strcat (format nil "~a" (first elm)) ".nod") :direction :output  :if-exists
:append :if-does-not-exist :create)))
    (format stream "~%~a \"~a\" ~a ~a ~&(" (first elm) (cadr elm)
     (+ x-lev (or (nth 3 (get-listc (car elm) node-offs-list)) 0))
     (+ 5000 (* -1 (+ y-lev (or (nth 4 (get-listc (car elm) node-offs-list)) 0)))))
    (dolist (elm1 (nth 3 elm))
     (format stream " ~s" elm1))
    (format stream " )")
    (do ((l-do (get-par concs rels 'li ) (cdr l-do))
         (n 1 (1+ n)))
        ((endp l-do))
        (format stream " ~a  " (car l-do)))
    (format stream "~&~a~%~%" (nth 4 elm))
    (close stream)))
  (dolist (elm2 (get-list-pp concs))
   (modify-par elm2 concs))))

;;*************************************************************************
;;
;;; call trans on top-level model file, here "vent.cpn".
;;*************************************************************************
;;

(trans '|vent| 0 0 0 0)
(exit)
```

F.6 Model Files

The model files implement the respiratory monitoring application as described in section 6.3. Each application is modelled from several aspects, such as the process model, panel, physical, and processor (section 3.6.1). The model files are therefore organised according to these aspects. Within each aspect there is a project file (*.prj) which basically contains a list of the model databases for that aspect.

F.6.1 PML Aspect

This section contains the Process model (PML) files for the prototype application (section 6.3). This includes the project file "resp.prj" and a number of PML database files.

/home/vandy/Designs/Resp2/Pml/resp.prj

This is the project file for the PML aspect.

```
;;; Project file: |resp.prj|
(DEF-PROJECT
   (|resp| "/users/vandy/Designs/Resp2/Pml/"
         "/home/vandy/Icons/Resp2/Pml/")
   "vent.pml" "gasx.pml" "pul_circ.pml" "o2trans.pml" "sys_circ.pml"
   "o2cons.pml" "oxyg.pml" "o2del.pml" "resp.pml")
```

/home/vandy/Designs/Resp2/Pml/vent.pml

The "vent.pml" file contains all the definitions for the class of ventilation processes from the PML aspect.

```
;;; PML database: "vent.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |ventilation|.
(DEF-PROCESS |ventilation| (->) NIL
   ((|mv_lo| NIL) (|mv_hi| NIL) (|fio2_hi| NIL) (|prp_hi| NIL)
    (|vdot_a_lo| NIL))
   ((|pao2| :DOUBLE) (|sao2| :DOUBLE) (|paco2| :DOUBLE)) NIL NIL NIL
   NIL NIL NIL
   ((|Structural| (T |ventilation| 337 6))
    (|Monitoring+Control| (O |pao2| |Signal| 255 282 255 270)
      (O |sao2| |Signal| 255 322 255 310)
      (O |paco2| |Signal| 255 362 255 350)
      (O |vte| |Signal| 85 42 85 30) (O |f| |Signal| 85 82 85 70)
      (O |mv| |Signal| 85 122 85 110)
      (O |peak| |Signal| 85 162 85 150)
      (O |plat| |Signal| 85 202 85 190)
      (O |peep| |Signal| 85 242 85 230)
      (O |pmin| |Signal| 85 282 85 270)
      (O |mean| |Signal| 85 322 85 310)
      (O |fio2| |Signal| 85 362 85 350)
      (O |c| |Signal| 85 402 85 390) (O |r| |Signal| 85 442 85 430)
      (O |mvs| |Signal| 85 482 85 470)
      (O |fs| |Signal| 85 522 85 510)
      (O |etco2| |Signal| 255 242 255 230)
      (O |ute| |Event| 427 81 426 105)
      (O |fio2_t| |Event| 429 206 429 233)
      (O |sao2_t| |Event| 428 166 423 189)
      (O |hit_e| |Event| 393 82 377 105)
      (O |hypo_vent_al| |OnlineAlarm| 255 42 255 30)
      (O |hyper_vent_al| |OnlineAlarm| 255 82 255 70)
      (O |FIO2_hi_al| |OnlineAlarm| 255 122 255 110)
      (O |PRP_hi_al| |OnlineAlarm| 255 162 255 150)
      (O |ALVENT_lo_al| |OnlineAlarm| 255 202 255 190)
      (O |Resp_acid_al| |OfflineAlarm| 255 402 255 390)
      (O |Resp_alka_al| |OfflineAlarm| 255 442 255 430)
      (O |ip1| |InterfacePoint| 15 42 5 30)
      (O |ip2| |InterfacePoint| 15 82 5 70)
      (O |ip3| |InterfacePoint| 15 122 5 110)
      (O |ip4| |InterfacePoint| 15 162 5 150)
      (O |ip5| |InterfacePoint| 15 202 5 190)
      (O |ip6| |InterfacePoint| 15 242 5 230)
      (O |ip7| |InterfacePoint| 15 282 5 270)
```

```
(O |ip8| |InterfacePoint| 15 322 5 310)
(O |ip9| |InterfacePoint| 15 362 5 350)
(O |ip10| |InterfacePoint| 15 402 5 390)
(O |ip11| |InterfacePoint| 15 442 5 430)
(O |ip12| |InterfacePoint| 15 482 5 470)
(O |ip13| |InterfacePoint| 15 522 5 510)
(O |ip14| |InterfacePoint| 195 42 195 30)
(O |ip15| |InterfacePoint| 195 82 195 70)
(O |ip16| |InterfacePoint| 195 122 195 110)
(O |ip17| |InterfacePoint| 195 162 195 150)
(O |ip19| |InterfacePoint| 195 242 195 230)
(O |ip18| |InterfacePoint| 195 202 195 190)
(O |fio2_hi_pro| |Alarm2P| 461 138 430 121)
(O |hi_lo_vnt| |Alarm4P| 458 29 458 19)
(O |ip20| |InterfacePoint| 195 482 195 470)
(O |ip21| |InterfacePoint| 195 522 195 510)
(O |petco2_hi| |OnlineAlarm| 255 482 255 470)
(O |petco2_lo| |OnlineAlarm| 255 522 255 510)
(C |IO connection| (|ip1| NIL NIL NIL) (|vte| NIL NIL NIL)
  |White| 35 50 85 50)
(C |IO connection| (|ip2| NIL NIL NIL) (|f| NIL NIL NIL)
  |White| 35 90 85 90)
(C |IO connection| (|ip3| NIL NIL NIL) (|mv| NIL NIL NIL)
  |White| 35 130 85 130)
(C |IO connection| (|ip4| NIL NIL NIL) (|peak| NIL NIL NIL)
  |White| 35 170 85 170)
(C |IO connection| (|ip5| NIL NIL NIL) (|plat| NIL NIL NIL)
  |White| 35 210 85 210)
(C |IO connection| (|ip6| NIL NIL NIL) (|peep| NIL NIL NIL)
  |White| 35 250 85 250)
(C |IO connection| (|ip7| NIL NIL NIL) (|pmin| NIL NIL NIL)
  |White| 35 290 85 290)
(C |IO connection| (|ip8| NIL NIL NIL) (|mean| NIL NIL NIL)
  |White| 35 330 85 330)
(C |IO connection| (|ip9| NIL NIL NIL) (|fio2| NIL NIL NIL)
  |White| 35 370 85 370)
(C |IO connection| (|ip10| NIL NIL NIL) (|c| NIL NIL NIL)
  |White| 35 410 85 410)
(C |IO connection| (|ip11| NIL NIL NIL) (|r| NIL NIL NIL)
  |White| 35 450 85 450)
(C |IO connection| (|ip12| NIL NIL NIL) (|mvs| NIL NIL NIL)
  |White| 35 490 85 490)
(C |IO connection| (|ip13| NIL NIL NIL) (|fs| NIL NIL NIL)
  |White| 35 530 85 530)
(C |IO connection| (|ip14| NIL NIL NIL)
  (|hypo_vent_al| NIL NIL NIL) |White| 215 50 260 50)
(C |IO connection| (|ip15| NIL NIL NIL)
  (|hyper_vent_al| NIL NIL NIL) |White| 215 90 260 90)
(C |IO connection| (|ip16| NIL NIL NIL)
  (|FIO2_hi_al| NIL NIL NIL) |White| 215 130 260 130)
(C |IO connection| (|ip17| NIL NIL NIL)
  (|PRP_hi_al| NIL NIL NIL) |White| 215 170 260 170)
```

```
(C |IO connection| (|ip19| NIL NIL NIL) (|etco2| NIL NIL NIL)
  |White| 215 250 260 250)
(C |IO connection| (|ip18| NIL NIL NIL)
  (|ALVENT_lo_al| NIL NIL NIL) |White| 215 210 260 210)
(C |Dataflow| (|sao2| NIL NIL NIL)
  (|fio2_hi_pro| |Signal1| INSIGS 0) |White| 273 333 418 333
  418 146 468 146)
(C |Dataflow| (|fio2| NIL NIL NIL)
  (|fio2_hi_pro| |Signal2| INSIGS 1) |White| 101 373 115 373
  115 156 468 156)
(C |Dataflow| (|fio2_hi_pro| |out| OUTALS 0)
  (|FIO2_hi_al| NIL NIL NIL) |White| 512 165 530 165 530 129
  274 129)
(C |Dataflow| (|sao2_t| NIL NIL NIL)
  (|fio2_hi_pro| |Threshold1| INEVS 0) |White| 444 185 464
  185)
(C |Dataflow| (|fio2_t| NIL NIL NIL)
  (|fio2_hi_pro| |Threshold2| INEVS 1) |White| 447 219 464
  219 464 197)
(C |Dataflow| (|hi_lo_vnt| |out1| OUTALS 0)
  (|hypo_vent_al| NIL NIL NIL) |White| 511 36 523 36 523 24
  266 24 266 47)
(C |Dataflow| (|hi_lo_vnt| |out2| OUTALS 1)
  (|hyper_vent_al| NIL NIL NIL) |White| 506 55 291 55 291 91
  276 91)
(C |Dataflow| (|mv| NIL NIL NIL)
  (|hi_lo_vnt| |Signal| INSIGS 0) |White| 102 133 116 133 116
  66 467 66)
(C |Dataflow| (|ute| NIL NIL NIL)
  (|hi_lo_vnt| |Threshold1| INEVS 0) |White| 437 94 437 77
  466 77)
(C |Dataflow| (|hit_e| NIL NIL NIL)
  (|hi_lo_vnt| |Threshold2| INEVS 1) |White| 408 92 462 88)
(C |IO connection| (|ip20| NIL NIL NIL)
  (|petco2_hi| NIL NIL NIL) |White| 215 490 256 490)
(C |IO connection| (|ip21| NIL NIL NIL)
  (|petco2_lo| NIL NIL NIL) |White| 215 530 256 530)
(T |ventilation| 337 6))
(|Simulation| (O |petco2_lo| |OnlineAlarm| 258 521 258 515)
  (O |petco2_hi| |OnlineAlarm| 258 483 257 473)
  (O |hit_e| |Event| 393 82 377 105)
  (O |sao2_t| |Event| 429 186 430 211)
  (O |fio2_t| |Event| 427 142 428 164)
  (O |ute| |Event| 46 180 45 175)
  (O |etco2| |Signal| 259 332 258 318)
  (O |hypo_vent_al| |OnlineAlarm| 255 42 255 30)
  (O |hyper_vent_al| |OnlineAlarm| 255 82 255 70)
  (O |FIO2_hi_al| |OnlineAlarm| 255 122 255 110)
  (O |PRP_hi_al| |OnlineAlarm| 255 162 255 150)
  (O |ALVENT_lo_al| |OnlineAlarm| 256 449 256 439)
  (O |Resp_acid_al| |OfflineAlarm| 255 372 255 360)
  (O |Resp_alka_al| |OfflineAlarm| 255 412 262 400)
```

```
  (O |vte| |Signal| 85 42 85 30) (O |f| |Signal| 85 82 85 70)
  (O |mv| |Signal| 85 122 85 110)
  (O |peak| |Signal| 85 162 85 150)
  (O |plat| |Signal| 85 202 85 190)
  (O |peep| |Signal| 85 242 85 230)
  (O |pmin| |Signal| 85 282 85 270)
  (O |mean| |Signal| 85 322 85 310)
  (O |fio2| |Signal| 85 362 85 350)
  (O |c| |Signal| 85 402 85 390) (O |r| |Signal| 85 442 85 430)
  (O |mvs| |Signal| 85 482 85 470)
  (O |fs| |Signal| 85 522 85 510) (T |ventilation| 337 6))
(|StateTransitions| (O |hit_e| |Event| 393 82 377 105)
  (O |sao2_t| |Event| 429 186 430 211)
  (O |fio2_t| |Event| 427 142 428 164)
  (O |ute| |Event| 46 180 45 175) (T |ventilation| 337 6))
(|FailurePropagation| (O |mv_lo| |FailureMode| 25 44 2 30)
  (O |mv_hi| |FailureMode| 25 84 2 70)
  (O |fio2_hi| |FailureMode| 25 122 2 110)
  (O |prp_hi| |FailureMode| 25 162 2 150)
  (O |vdot_a_lo| |FailureMode| 25 202 2 190)
  (O |hypo_vent_al| |OnlineAlarm| 105 42 105 30)
  (O |hyper_vent_al| |OnlineAlarm| 105 82 105 70)
  (O |FIO2_hi_al| |OnlineAlarm| 105 122 105 110)
  (O |PRP_hi_al| |OnlineAlarm| 105 162 105 150)
  (O |ALVENT_lo_al| |OnlineAlarm| 105 202 105 190)
  (O |Resp_acid_al| |OfflineAlarm| 105 372 105 360)
  (O |Resp_alka_al| |OfflineAlarm| 105 412 105 400)
  (O |petco2_hi| |OnlineAlarm| 106 248 106 237)
  (O |petco2_lo| |OnlineAlarm| 106 294 105 283)
  (C |Failuremode exclusivity| (|mv_lo| NIL NIL NIL)
    (|mv_hi| NIL NIL NIL) |White| 42 62 42 94)
  (C |Failure/Alarm connection| (|mv_lo| NIL NIL NIL)
    (|hypo_vent_al| NIL NIL NIL) |White| 45 55 110 55)
  (C |Failure/Alarm connection| (|mv_hi| NIL NIL NIL)
    (|hyper_vent_al| NIL NIL NIL) |White| 45 95 110 95)
  (C |Failure/Alarm connection| (|fio2_hi| NIL NIL NIL)
    (|FIO2_hi_al| NIL NIL NIL) |White| 45 135 110 135)
  (C |Failure/Alarm connection| (|prp_hi| NIL NIL NIL)
    (|PRP_hi_al| NIL NIL NIL) |White| 45 175 110 175)
  (C |Failure/Alarm connection| (|vdot_a_lo| NIL NIL NIL)
    (|ALVENT_lo_al| NIL NIL NIL) |White| 45 215 110 215)
  (C |Failure/Alarm connection| (|mv_lo| NIL NIL NIL)
    (|Resp_acid_al| NIL NIL NIL) |White| 30 54 14 54 14 383 108
    383)
  (C |Failure/Alarm connection| (|mv_hi| NIL NIL NIL)
    (|Resp_alka_al| NIL NIL NIL) |White| 30 95 6 95 6 423 108
    423)
  (C |Failure/Alarm connection| (|mv_lo| NIL NIL NIL)
    (|petco2_hi| NIL NIL NIL) |White| 30 54 14 54 14 259 108
    259)
  (C |Failure/Alarm connection| (|mv_hi| NIL NIL NIL)
    (|petco2_lo| NIL NIL NIL) |White| 30 95 6 95 6 308 108 308)
```

```
  (T |ventilation| 337 6))
(|OperatorInterface| (O |petco2_lo| |OnlineAlarm| 258 521 258 515)
  (O |petco2_hi| |OnlineAlarm| 258 483 257 473)
  (O |vte| |Signal| 85 42 85 30) (O |f| |Signal| 85 82 85 70)
  (O |mv| |Signal| 85 122 85 110)
  (O |peak| |Signal| 85 162 85 150)
  (O |plat| |Signal| 85 202 85 190)
  (O |peep| |Signal| 85 242 85 230)
  (O |pmin| |Signal| 85 282 85 270)
  (O |mean| |Signal| 85 322 85 310)
  (O |fio2| |Signal| 85 362 85 350)
  (O |c| |Signal| 85 402 85 390) (O |r| |Signal| 85 442 85 430)
  (O |mvs| |Signal| 85 482 85 470)
  (O |fs| |Signal| 85 522 85 510)
  (O |etco2| |Signal| 483 214 482 197)
  (O |ute| |Event| 488 368 486 353)
  (O |fio2_t| |Event| 488 331 489 317)
  (O |sao2_t| |Event| 489 405 489 391)
  (O |hit_e| |Event| 489 295 488 281)
  (O |hypo_vent_al| |OnlineAlarm| 355 42 355 30)
  (O |hyper_vent_al| |OnlineAlarm| 355 82 355 70)
  (O |FIO2_hi_al| |OnlineAlarm| 355 122 355 110)
  (O |PRP_hi_al| |OnlineAlarm| 355 162 355 150)
  (O |ALVENT_lo_al| |OnlineAlarm| 485 41 483 29)
  (O |Resp_acid_al| |OfflineAlarm| 355 382 355 370)
  (O |Resp_alka_al| |OfflineAlarm| 355 422 355 410)
  (O |ventilation| |vent| 299 212 313 354)
  (C |Display connection| (|peak| NIL NIL NIL)
    (|ventilation| |peak| INSIGNALS 3) |White| 100 175 237 175
    237 261 304 261)
  (C |Display connection| (|plat| NIL NIL NIL)
    (|ventilation| |plat| INSIGNALS 4) |White| 100 215 227 215
    227 270 304 270)
  (C |Display connection| (|peep| NIL NIL NIL)
    (|ventilation| |peep| INSIGNALS 5) |White| 100 255 216 255
    216 280 304 280)
  (C |Display connection| (|mean| NIL NIL NIL)
    (|ventilation| |mean| INSIGNALS 7) |White| 100 335 216 335
    216 300 304 300)
  (C |Display connection| (|fio2| NIL NIL NIL)
    (|ventilation| |fio2| INSIGNALS 8) |White| 100 375 227 375
    227 308 304 308)
  (C |Display connection| (|mvs| NIL NIL NIL)
    (|ventilation| |mvs| INSIGNALS 11) |White| 100 495 282 495
    282 330 304 330)
  (C |Display connection| (|fs| NIL NIL NIL)
    (|ventilation| |fs| INSIGNALS 12) |White| 100 535 292 535
    292 342 304 342)
  (C |Display connection| (|r| NIL NIL NIL)
    (|ventilation| |r| INSIGNALS 10) |White| 100 455 276 455
    276 325 304 325)
  (C |Display connection| (|c| NIL NIL NIL)
```

```
    (|ventilation| |c| INSIGNALS 9) |White| 100 415 245 415 245
    316 304 316)
  (C |Display connection| (|mv| NIL NIL NIL)
    (|ventilation| |mv| INSIGNALS 2) |White| 100 135 245 135
    245 241 304 241)
  (C |Display connection| (|f| NIL NIL NIL)
    (|ventilation| |f| INSIGNALS 1) |White| 100 95 264 95 264
    229 304 229)
  (C |Display connection| (|vte| NIL NIL NIL)
    (|ventilation| |vte| INSIGNALS 0) |White| 100 55 278 55 278
    219 304 219)
  (C |Display connection| (|pmin| NIL NIL NIL)
    (|ventilation| |pmin| INSIGNALS 6) |White| 100 290 304 290)
  (C |Alarm connection| (|hypo_vent_al| NIL NIL NIL)
    (|ventilation| |hypo_ven| ALARMS 0) |White| 373 54 434 54
    434 220 414 220)
  (C |Control connection| (|ventilation| |ev1| OUTEVENTS 0)
    (|ute| NIL NIL NIL) |White| 416 323 430 323 430 375 494
    375)
  (C |Alarm connection| (|hyper_vent_al| NIL NIL NIL)
    (|ventilation| |hyper_ven| ALARMS 2) |White| 373 95 437 95
    437 233 414 233)
  (C |Alarm connection| (|FIO2_hi_al| NIL NIL NIL)
    (|ventilation| |fio2_hi| ALARMS 1) |White| 373 133 444 133
    444 226 414 226)
  (C |Alarm connection| (|PRP_hi_al| NIL NIL NIL)
    (|ventilation| |PRP_hi| ALARMS 3) |White| 375 174 456 174
    456 240 414 240)
  (C |Alarm connection| (|ALVENT_lo_al| NIL NIL NIL)
    (|ventilation| |Alv_ven_lo| ALARMS 4) |White| 489 54 450 54
    450 250 414 250)
  (C |Control connection| (|ventilation| |ev3| OUTEVENTS 2)
    (|sao2_t| NIL NIL NIL) |White| 415 338 424 338 424 419 493
    419)
  (C |Control connection| (|ventilation| |ev2| OUTEVENTS 1)
    (|fio2_t| NIL NIL NIL) |White| 415 308 443 308 443 343 491
    343)
  (C |Control connection| (|ventilation| |ev4| OUTEVENTS 3)
    (|hit_e| NIL NIL NIL) |White| 414 294 453 294 453 310 494
    310)
  (C |Alarm connection| (|Resp_acid_al| NIL NIL NIL)
    (|ventilation| |Resp_acid| ALARMS 5) |White| 372 395 444
    395 444 262 414 262)
  (C |Alarm connection| (|Resp_alka_al| NIL NIL NIL)
    (|ventilation| |Resp_alka| ALARMS 6) |White| 375 434 433
    434 433 274 414 274)
  (T |ventilation| 337 6)))
((|Structural| . "vent-str.icon")
 (|Monitoring+Control| . "vent-mon.icon") (|Simulation|)
 (|StateTransitions|) (|FailurePropagation| . "vent-fpg.icon")
 (|OperatorInterface|))
(PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
```

```
(PARAMETERS)
(SIGNALS (|vte| :DOUBLE NIL) (|f| :DOUBLE NIL) (|mv| :DOUBLE NIL)
      (|peak| :DOUBLE NIL) (|plat| :DOUBLE NIL)
      (|peep| :DOUBLE NIL) (|pmin| :DOUBLE NIL)
      (|mean| :DOUBLE NIL) (|fio2| :DOUBLE NIL)
      (|c| :DOUBLE NIL) (|r| :DOUBLE NIL) (|mvs| :DOUBLE NIL)
      (|fs| :DOUBLE NIL) (|etco2| :DOUBLE NIL))
(EVENTS (|ute| :DOUBLE NIL NIL) (|fio2_t| :DOUBLE NIL NIL)
     (|sao2_t| :DOUBLE NIL NIL) (|hit_e| :DOUBLE NIL NIL))
(ALARMS (|hypo_vent_al| :ONLINE) (|hyper_vent_al| :ONLINE)
      (|FIO2_hi_al| :ONLINE) (|PRP_hi_al| :ONLINE)
      (|ALVENT_lo_al| :ONLINE) (|Resp_acid_al| :OFFLINE)
      (|Resp_alka_al| :OFFLINE) (|petco2_hi| :ONLINE)
      (|petco2_lo| :ONLINE))
(INTERFACE ((|ip1| SSIGNALIFP) -> |vte|)
   ((|ip2| SSIGNALIFP) -> |f|) ((|ip3| SSIGNALIFP) -> |mv|)
   ((|ip4| SSIGNALIFP) -> |peak|) ((|ip5| SSIGNALIFP) -> |plat|)
   ((|ip6| SSIGNALIFP) -> |peep|) ((|ip7| SSIGNALIFP) -> |pmin|)
   ((|ip8| SSIGNALIFP) -> |mean|) ((|ip9| SSIGNALIFP) -> |fio2|)
   ((|ip10| SSIGNALIFP) -> |c|) ((|ip11| SSIGNALIFP) -> |r|)
   ((|ip12| SSIGNALIFP) -> |mvs|) ((|ip13| SSIGNALIFP) -> |fs|)
   ((|ip14| ALARMIFP) -> |hypo_vent_al|)
   ((|ip15| ALARMIFP) -> |hyper_vent_al|)
   ((|ip16| ALARMIFP) -> |FIO2_hi_al|)
   ((|ip17| ALARMIFP) -> |PRP_hi_al|)
   ((|ip19| SSIGNALIFP) -> |etco2|)
   ((|ip18| ALARMIFP) -> |ALVENT_lo_al|)
   ((|ip20| ALARMIFP) -> |petco2_hi|)
   ((|ip21| ALARMIFP) -> |petco2_lo|))
(RT-DATABASE)
(PROCESSORS
   (|fio2_hi_pro|
      (|Alarm2P|
         (((|sao2| |fio2| |sao2_t| |fio2_t|) -> (|FIO2_hi_al|))
          NIL NIL NIL)))
   (|hi_lo_vnt|
      (|Alarm4P|
         (((|mv| |ute| |hit_e|) ->
           (|hypo_vent_al| |hyper_vent_al|))
          NIL NIL NIL))))
(CONTROL)
(FAULT-MODES
   (|mv_lo| (|hypo_vent_al| |Resp_acid_al| |petco2_hi|) (|mv_hi|)
         NIL NIL)
   (|mv_hi| (|hyper_vent_al| |Resp_alka_al| |petco2_lo|) NIL NIL
         NIL)
   (|fio2_hi| (|FIO2_hi_al|) NIL NIL NIL)
   (|prp_hi| (|PRP_hi_al|) NIL NIL NIL)
   (|vdot_a_lo| (|ALVENT_lo_al|) NIL NIL NIL))
(FAULT-GRAPH) (SIMULATION)
(PANEL NIL
     (|ventilation|
```

```
        (|vent| (|vte| |f| |mv| |peak| |plat| |peep| |pmin|
                 |mean| |fio2| |c| |r| |mvs| |fs|)
              NIL NIL (|ute| |fio2_t| |sao2_t| |hit_e|)
              (|hypo_vent_al| |hyper_vent_al| |FIO2_hi_al|
                 |PRP_hi_al| |ALVENT_lo_al| |Resp_acid_al|
                 |Resp_alka_al|))))
(SUBPROC))
```

/home/vandy/Designs/Resp2/Pml/gasx.pml

The "gasx.pml" describes the class of processes of type gas-exchange.

```
;;; PML database: "gasx.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |gasx|.
(DEF-PROCESS |gasx| (->) NIL ((|impaired_diff| NIL)) NIL NIL NIL NIL
  NIL NIL NIL
  ((|Structural| (T |gasx| 337 6))
   (|Monitoring+Control| (O |Rdiff_hi| |OnlineAlarm| 125 52 125 40)
     (O |ip1| |InterfacePoint| 40 52 40 40)
     (C |IO connection| (|ip1| NIL NIL NIL)
       (|Rdiff_hi| NIL NIL NIL) |White| 60 65 130 65)
     (T |gasx| 337 6))
   (|Simulation| (O |Rdiff_hi| |OnlineAlarm| 121 53 122 42)
     (T |gasx| 337 6))
   (|StateTransitions| (T |gasx| 337 6))
   (|FailurePropagation| (O |impaired_diff| |FailureMode| 25 52 0 40)
     (O |Rdiff_hi| |OnlineAlarm| 125 52 125 40)
     (C |Failure/Alarm connection| (|impaired_diff| NIL NIL NIL)
       (|Rdiff_hi| NIL NIL NIL) |White| 45 65 130 65)
     (T |gasx| 337 6))
   (|OperatorInterface| (O |Rdiff_hi| |OnlineAlarm| 125 52 125 40)
     (T |gasx| 337 6)))
  ((|Structural| . "gasx-str.icon")
   (|Monitoring+Control| . "gasx-mon.icon") (|Simulation|)
   (|StateTransitions|) (|FailurePropagation| . "gasx-fpg.icon")
   (|OperatorInterface|))
  (PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
  (PARAMETERS) (SIGNALS) (EVENTS) (ALARMS (|Rdiff_hi| :ONLINE))
  (INTERFACE ((|ip1| ALARMIFP) -> |Rdiff_hi|)) (RT-DATABASE)
  (PROCESSORS) (CONTROL)
  (FAULT-MODES (|impaired_diff| (|Rdiff_hi|) NIL NIL NIL))
  (FAULT-GRAPH) (SIMULATION) (PANEL NIL) (SUBPROC))
```

/home/vandy/Designs/Resp2/Pml/pul_circ.pml

The "pul_circ" describes the pulmonary circulation type of process.

```
;;; PML database: "pul_circ.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |pul_circ|.
(DEF-PROCESS |pul_circ| (->) NIL
  ((|pul_hi_tens| NIL) (|shunt_hi| NIL) (|preload_lo| NIL)) NIL NIL
  NIL NIL NIL NIL NIL
  ((|Structural| (T |pul_circ| 337 6))
   (|Monitoring+Control| (O |PAPm| |Signal| 125 135 126 124)
     (O |PAPs| |Signal| 126 209 126 201)
     (O |PAPd| |Signal| 127 246 129 237)
     (O |Qs/Qt_hi_al| |OnlineAlarm| 125 92 125 80)
     (O |pul_hi_t_al| |OnlineAlarm| 125 52 125 40)
     (O |PAPd-PAWP_hi| |OfflineAlarm| 126 176 125 165)
     (O |PAWP_hi| |OfflineAlarm| 128 293 130 278)
     (O |PAWP_lo| |OfflineAlarm| 129 335 129 320)
     (O |ip1| |InterfacePoint| 40 92 40 80)
     (O |ip2| |InterfacePoint| 40 52 40 40)
     (O |ip3| |InterfacePoint| 40 138 41 124)
     (O |ip4| |InterfacePoint| 42 212 41 201)
     (O |ip5| |InterfacePoint| 43 250 42 241)
     (C |IO connection| (|ip1| NIL NIL NIL)
       (|Qs/Qt_hi_al| NIL NIL NIL) |White| 60 102 130 102)
     (C |IO connection| (|ip2| NIL NIL NIL)
       (|pul_hi_t_al| NIL NIL NIL) |White| 60 62 130 62)
     (C |IO connection| (|ip3| NIL NIL NIL) (|PAPm| NIL NIL NIL)
       |White| 58 147 128 148)
     (C |IO connection| (|ip4| NIL NIL NIL) (|PAPs| NIL NIL NIL)
       |White| 61 220 129 221)
     (C |IO connection| (|ip5| NIL NIL NIL) (|PAPd| NIL NIL NIL)
       |White| 63 258 131 258)
     (T |pul_circ| 337 6))
   (|Simulation| (O |PAWP_lo| |OfflineAlarm| 129 335 129 320)
     (O |PAWP_hi| |OfflineAlarm| 128 293 130 278)
     (O |PAPd-PAWP_hi| |OfflineAlarm| 126 176 125 165)
     (O |PAPd| |Signal| 127 246 129 237)
     (O |PAPs| |Signal| 126 209 126 201)
     (O |PAPm| |Signal| 125 135 126 124)
     (O |pul_hi_t_al| |OnlineAlarm| 126 53 124 41)
     (O |Qs/Qt_hi_al| |OnlineAlarm| 125 92 125 80)
     (T |pul_circ| 337 6))
   (|StateTransitions| (T |pul_circ| 337 6))
   (|FailurePropagation| (O |pul_hi_tens| |FailureMode| 25 52 5 40)
     (O |shunt_hi| |FailureMode| 25 92 5 80)
     (O |preload_lo| |FailureMode| 32 334 10 313)
```

```
    (O |Qs/Qt_hi_al| |OnlineAlarm| 125 92 125 80)
    (O |pul_hi_t_al| |OnlineAlarm| 125 52 125 40)
    (O |PAPd-PAWP_hi| |OfflineAlarm| 126 176 125 165)
    (O |PAWP_hi| |OfflineAlarm| 128 293 130 278)
    (O |PAWP_lo| |OfflineAlarm| 129 335 129 320)
    (C |Failure/Alarm connection| (|shunt_hi| NIL NIL NIL)
      (|Qs/Qt_hi_al| NIL NIL NIL) |White| 45 102 130 102)
    (C |Failure/Alarm connection| (|pul_hi_tens| NIL NIL NIL)
      (|pul_hi_t_al| NIL NIL NIL) |White| 45 62 130 62)
    (C |Failure/Alarm connection| (|pul_hi_tens| NIL NIL NIL)
      (|PAWP_hi| NIL NIL NIL) |White| 45 70 92 70 92 306 134 306)
    (C |Failure/Alarm connection| (|preload_lo| NIL NIL NIL)
      (|PAWP_lo| NIL NIL NIL) |White| 52 347 134 348)
    (C |Failure/Alarm connection| (|preload_lo| NIL NIL NIL)
      (|PAPd-PAWP_hi| NIL NIL NIL) |White| 51 341 69 341 69 187
      131 187)
    (T |pul_circ| 337 6))
  (|OperatorInterface| (O |PAWP_lo| |OfflineAlarm| 129 335 129 320)
    (O |PAPm| |Signal| 125 135 126 124)
    (O |PAPs| |Signal| 126 209 126 201)
    (O |PAPd| |Signal| 127 246 129 237)
    (O |Qs/Qt_hi_al| |OnlineAlarm| 125 92 125 80)
    (O |pul_hi_t_al| |OnlineAlarm| 126 53 124 41)
    (O |PAPd-PAWP_hi| |OfflineAlarm| 126 176 125 165)
    (O |PAWP_hi| |OfflineAlarm| 128 293 130 278)
    (O |pul_circ_pnl| |pul_circulation| 336 162 342 272)
    (C |Display connection| (|PAPm| NIL NIL NIL)
      (|pul_circ_pnl| |s1| INSIGNALS 0) |White| 138 146 269 146
      269 171 342 171)
    (C |Display connection| (|PAPs| NIL NIL NIL)
      (|pul_circ_pnl| |s2| INSIGNALS 1) |White| 134 219 273 219
      273 188 340 188)
    (C |Display connection| (|PAPd| NIL NIL NIL)
      (|pul_circ_pnl| |s3| INSIGNALS 2) |White| 141 256 286 256
      286 201 342 201)
    (C |Alarm connection| (|pul_hi_t_al| NIL NIL NIL)
      (|pul_circ_pnl| |pul_hi_t_al| ALARMS 1) |White| 143 65 319
      65 319 230 341 230)
    (C |Alarm connection| (|Qs/Qt_hi_al| NIL NIL NIL)
      (|pul_circ_pnl| |Qs/Qt_hi_al| ALARMS 0) |White| 143 106 314
      106 314 216 339 216)
    (C |Alarm connection| (|PAPd-PAWP_hi| NIL NIL NIL)
      (|pul_circ_pnl| |PAPd-PAWP_hi| ALARMS 2) |White| 143 190
      246 190 246 244 339 244)
    (C |Alarm connection| (|PAWP_hi| NIL NIL NIL)
      (|pul_circ_pnl| |PAWP_hi| ALARMS 3) |White| 146 304 314 304
      314 255 340 255)
    (T |pul_circ| 337 6)))
((|Structural| . "pulcirc-str.icon")
 (|Monitoring+Control| . "pulcirc-mon.icon") (|Simulation|)
 (|StateTransitions|) (|FailurePropagation| . "pulcirc-fpg.icon")
 (|OperatorInterface|))
```

```
(PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
(PARAMETERS)
(SIGNALS (|PAPm| :FLOAT NIL) (|PAPs| :FLOAT NIL)
      (|PAPd| :FLOAT NIL))
(EVENTS)
(ALARMS (|Qs/Qt_hi_al| :ONLINE) (|pul_hi_t_al| :ONLINE)
      (|PAPd-PAWP_hi| :OFFLINE) (|PAWP_hi| :OFFLINE)
      (|PAWP_lo| :OFFLINE))
(INTERFACE ((|ip1| ALARMIFP) -> |Qs/Qt_hi_al|)
   ((|ip2| ALARMIFP) -> |pul_hi_t_al|)
   ((|ip3| FSIGNALIFP) -> |PAPm|) ((|ip4| FSIGNALIFP) -> |PAPs|)
   ((|ip5| FSIGNALIFP) -> |PAPd|))
(RT-DATABASE) (PROCESSORS) (CONTROL)
(FAULT-MODES (|pul_hi_tens| (|pul_hi_t_al| |PAWP_hi|) NIL NIL NIL)
   (|shunt_hi| (|Qs/Qt_hi_al|) NIL NIL NIL)
   (|preload_lo| (|PAWP_lo| |PAPd-PAWP_hi|) NIL NIL NIL))
(FAULT-GRAPH) (SIMULATION)
(PANEL NIL
     (|pul_circ_pnl|
       (|pul_circulation| (|PAPm| |PAPs| |PAPd|) NIL NIL NIL
         (|Qs/Qt_hi_al| |pul_hi_t_al| |PAPd-PAWP_hi|
           |PAWP_hi|))))
(SUBPROC))
```

/home/vandy/Designs/Resp2/Pml/o2trans.pml

The "o2trans" describes the oxygen transport process type.

```
;;; PML database: "o2trans.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |O2_transport|.
(DEF-PROCESS |O2_transport| (->) NIL
   ((|cto2_lo| NIL) (|tHb_lo| NIL) (|Hb_affinity_lo| NIL)
    (|Dys_Hb_hi| NIL))
   NIL NIL NIL NIL NIL NIL NIL
   ((|Structural| (T |O2_transport| 337 6))
    (|Monitoring+Control|
       (O |tHb_lo_al| |OfflineAlarm| 170 100 170 88)
       (O |p50_hi| |OfflineAlarm| 170 140 170 128)
       (O |CaO2_lo_al| |OfflineAlarm| 170 60 170 48)
       (O |Dys_Hb_hi_al| |OfflineAlarm| 170 180 170 168)
       (T |O2_transport| 337 6))
    (|Simulation| (O |Dys_Hb_hi_al| |OfflineAlarm| 170 176 171 166)
       (O |CaO2_lo_al| |OfflineAlarm| 167 58 170 46)
       (O |p50_hi| |OfflineAlarm| 170 136 170 129)
       (O |tHb_lo_al| |OfflineAlarm| 168 55 168 48)
       (T |O2_transport| 337 6))
    (|StateTransitions| (T |O2_transport| 337 6))
    (|FailurePropagation| (O |cto2_lo| |FailureMode| 125 60 87 48)
       (O |tHb_lo| |FailureMode| 125 100 94 88)
       (O |Hb_affinity_lo| |FailureMode| 125 140 23 128)
       (O |Dys_Hb_hi| |FailureMode| 125 180 69 168)
       (O |tHb_lo_al| |OfflineAlarm| 170 60 170 88)
       (O |p50_hi| |OfflineAlarm| 170 140 170 128)
       (O |CaO2_lo_al| |OfflineAlarm| 170 100 170 48)
       (O |Dys_Hb_hi_al| |OfflineAlarm| 170 180 170 168)
       (C |Failure/Alarm connection| (|cto2_lo| NIL NIL NIL)
          (|tHb_lo_al| NIL NIL NIL) |White| 143 71 175 72)
       (C |Failure/Alarm connection| (|tHb_lo| NIL NIL NIL)
          (|CaO2_lo_al| NIL NIL NIL) |White| 146 112 173 113)
       (C |Failure/Alarm connection| (|Hb_affinity_lo| NIL NIL NIL)
          (|p50_hi| NIL NIL NIL) |White| 145 152 173 152)
       (C |Failure/Alarm connection| (|Dys_Hb_hi| NIL NIL NIL)
          (|Dys_Hb_hi_al| NIL NIL NIL) |White| 145 192 175 192)
       (T |O2_transport| 337 6))
    (|OperatorInterface| (O |tHb_lo_al| |OfflineAlarm| 169 93 168 48)
       (O |p50_hi| |OfflineAlarm| 170 136 170 129)
       (O |CaO2_lo_al| |OfflineAlarm| 167 58 169 85)
       (O |Dys_Hb_hi_al| |OfflineAlarm| 170 176 171 166)
       (T |O2_transport| 337 6)))
   ((|Structural| . "o2trans-str.icon")
    (|Monitoring+Control| . "o2trans-mon.icon") (|Simulation|)
```

```
(|StateTransitions|) (|FailurePropagation| . "o2trans-fpg.icon")
(|OperatorInterface|))
(PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
(PARAMETERS) (SIGNALS) (EVENTS)
(ALARMS (|tHb_lo_al| :OFFLINE) (|p50_hi| :OFFLINE)
      (|CaO2_lo_al| :OFFLINE) (|Dys_Hb_hi_al| :OFFLINE))
(INTERFACE) (RT-DATABASE) (PROCESSORS) (CONTROL)
(FAULT-MODES (|cto2_lo| (|tHb_lo_al|) NIL NIL NIL)
   (|tHb_lo| (|CaO2_lo_al|) NIL NIL NIL)
   (|Hb_affinity_lo| (|p50_hi|) NIL NIL NIL)
   (|Dys_Hb_hi| (|Dys_Hb_hi_al|) NIL NIL NIL))
(FAULT-GRAPH) (SIMULATION) (PANEL NIL) (SUBPROC))
```

/home/vandy/Designs/Resp2/Pml/sys_circ.pml

The "sys_circ" pml model defines the systemic circulation process.

```
;;; PML database: "sys_circ.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |sys_circ|.
(DEF-PROCESS |sys_circ| (->) NIL ((|co_lo| NIL)) NIL NIL NIL NIL NIL
  NIL NIL
  ((|Structural| (T |sys_circ| 337 6))
   (|Monitoring+Control| (O HR |Signal| 100 62 100 50)
     (O |ABPs| |Signal| 100 102 100 90)
     (O |ABPd| |Signal| 100 142 100 130)
     (O CVP |Signal| 101 182 100 170)
     (O |ABPm| |Signal| 101 221 102 208)
     (O |Pulse_P| |Signal| 101 265 104 254)
     (O |CI_lo| |OfflineAlarm| 347 143 347 127)
     (O |Sin_Brady| |OnlineAlarm| 345 54 345 42)
     (O |Sin_Tachy| |OnlineAlarm| 346 92 346 81)
     (O |SI_lo| |OnlineAlarm| 349 183 348 170)
     (O |ABPs_lo| |OnlineAlarm| 351 229 350 216)
     (O |Pulse_P_lo| |OnlineAlarm| 352 274 352 262)
     (O |ip1| |InterfacePoint| 40 62 40 50)
     (O |ip2| |InterfacePoint| 40 102 40 90)
     (O |ip3| |InterfacePoint| 40 142 40 130)
     (O |ip4| |InterfacePoint| 40 182 40 170)
     (O |ip5| |InterfacePoint| 290 56 290 41)
     (O |ip6| |InterfacePoint| 290 92 290 83)
     (O |ip7| |InterfacePoint| 40 221 40 209)
     (O |ip8| |InterfacePoint| 290 183 290 175)
     (O |ip9| |InterfacePoint| 40 265 40 256)
     (O |ip10| |InterfacePoint| 290 229 290 221)
     (O |ip11| |InterfacePoint| 290 275 290 264)
     (O |ip12| |InterfacePoint| 292 313 292 304)
     (O |CVP_lo| |OnlineAlarm| 353 312 353 303)
     (C |IO connection| (|ip1| NIL NIL NIL) (HR NIL NIL NIL)
       |White| 60 75 105 75)
     (C |IO connection| (|ip2| NIL NIL NIL) (|ABPs| NIL NIL NIL)
       |White| 60 115 105 115)
     (C |IO connection| (|ip3| NIL NIL NIL) (|ABPd| NIL NIL NIL)
       |White| 60 155 105 155)
     (C |IO connection| (|ip4| NIL NIL NIL) (CVP NIL NIL NIL)
       |White| 60 195 105 195)
     (C |IO connection| (|ip5| NIL NIL NIL)
       (|Sin_Brady| NIL NIL NIL) |White| 311 65 350 65)
     (C |IO connection| (|ip6| NIL NIL NIL)
       (|Sin_Tachy| NIL NIL NIL) |White| 309 102 349 102)
     (C |IO connection| (|ip7| NIL NIL NIL) (|ABPm| NIL NIL NIL)
```

```
    |White| 60 231 106 231)
   (C |IO connection| (|ip8| NIL NIL NIL) (|SI_lo| NIL NIL NIL)
    |White| 310 194 354 194)
   (C |IO connection| (|ip9| NIL NIL NIL) (|Pulse_P| NIL NIL NIL)
    |White| 60 275 104 275)
   (C |IO connection| (|ip10| NIL NIL NIL)
    (|ABPs_lo| NIL NIL NIL) |White| 311 238 355 238)
   (C |IO connection| (|ip11| NIL NIL NIL)
    (|Pulse_P_lo| NIL NIL NIL) |White| 312 283 354 283)
   (C |IO connection| (|ip12| NIL NIL NIL) (|CVP_lo| NIL NIL NIL)
    |White| 311 321 356 322)
   (T |sys_circ| 337 6))
 (|Simulation| (O |CVP_lo| |OnlineAlarm| 353 312 353 303)
   (O |Pulse_P_lo| |OnlineAlarm| 352 274 352 262)
   (O |ABPs_lo| |OnlineAlarm| 351 229 350 216)
   (O |Pulse_P| |Signal| 101 265 104 254)
   (O CVP |Signal| 101 182 100 170) (O HR |Signal| 100 62 100 50)
   (O |SI_lo| |OnlineAlarm| 349 183 348 170)
   (O |CI_lo| |OfflineAlarm| 347 143 347 127)
   (O |ABPm| |Signal| 101 221 102 208)
   (O |ABPd| |Signal| 100 142 100 130)
   (O |ABPs| |Signal| 100 102 100 90)
   (O |Sin_Tachy| |OnlineAlarm| 346 92 346 81)
   (O |Sin_Brady| |OnlineAlarm| 345 54 345 42)
   (T |sys_circ| 337 6))
 (|StateTransitions| (T |sys_circ| 337 6))
 (|FailurePropagation| (O |co_lo| |FailureMode| 69 59 54 37)
   (O |CI_lo| |OfflineAlarm| 347 143 347 127)
   (O |Sin_Brady| |OnlineAlarm| 345 54 345 42)
   (O |Sin_Tachy| |OnlineAlarm| 346 92 346 81)
   (O |SI_lo| |OnlineAlarm| 349 183 348 170)
   (O |ABPs_lo| |OnlineAlarm| 351 229 350 216)
   (O |Pulse_P_lo| |OnlineAlarm| 352 274 352 262)
   (O |CVP_lo| |OnlineAlarm| 353 312 353 303)
   (C |Failure/Alarm connection| (|co_lo| NIL NIL NIL)
    (|ABPs_lo| NIL NIL NIL) |White| 85 70 278 70 278 241 354
    241)
   (C |Failure/Alarm connection| (|co_lo| NIL NIL NIL)
    (|Pulse_P_lo| NIL NIL NIL) |White| 88 75 264 75 264 286 356
    286)
   (C |Failure/Alarm connection| (|co_lo| NIL NIL NIL)
    (|CI_lo| NIL NIL NIL) |White| 90 65 288 65 288 156 351 156)
   (C |Failure/Alarm connection| (|co_lo| NIL NIL NIL)
    (|CVP_lo| NIL NIL NIL) |White| 89 80 254 80 254 327 355
    327)
   (T |sys_circ| 337 6))
 (|OperatorInterface| (O |CVP_lo| |OnlineAlarm| 353 312 353 303)
   (O |Pulse_P_lo| |OnlineAlarm| 352 274 352 262)
   (O |ABPs_lo| |OnlineAlarm| 351 229 350 216)
   (O |Pulse_P| |Signal| 101 265 104 254)
   (O CVP |Signal| 101 182 100 170) (O HR |Signal| 100 62 100 50)
   (O |ABPs| |Signal| 100 102 100 90)
```

```
    (O |ABPd| |Signal| 100 142 100 130)
    (O |ABPm| |Signal| 101 221 102 208)
    (O |CI_lo| |OfflineAlarm| 347 143 347 127)
    (O |Sin_Brady| |OnlineAlarm| 345 54 345 42)
    (O |Sin_Tachy| |OnlineAlarm| 346 92 346 81)
    (O |SI_lo| |OnlineAlarm| 349 183 348 170)
    (T |sys_circ| 337 6)))
((|Structural| . "syscirc-str.icon")
 (|Monitoring+Control| . "syscirc-mon.icon") (|Simulation|)
 (|StateTransitions|) (|FailurePropagation| . "syscirc-fpg.icon")
 (|OperatorInterface|))
(PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
(PARAMETERS)
(SIGNALS (HR :DOUBLE NIL) (|ABPs| :DOUBLE NIL) (|ABPd| :DOUBLE NIL)
       (CVP :DOUBLE NIL) (|ABPm| :DOUBLE NIL)
       (|Pulse_P| :DOUBLE NIL))
(EVENTS)
(ALARMS (|CI_lo| :OFFLINE) (|Sin_Brady| :ONLINE)
      (|Sin_Tachy| :ONLINE) (|SI_lo| :ONLINE) (|ABPs_lo| :ONLINE)
      (|Pulse_P_lo| :ONLINE) (|CVP_lo| :ONLINE))
(INTERFACE ((|ip1| SIGNALIFP) -> HR) ((|ip2| SIGNALIFP) -> |ABPs|)
   ((|ip3| SIGNALIFP) -> |ABPd|) ((|ip4| SIGNALIFP) -> CVP)
   ((|ip5| ALARMIFP) -> |Sin_Brady|)
   ((|ip6| ALARMIFP) -> |Sin_Tachy|) ((|ip7| SIGNALIFP) -> |ABPm|)
   ((|ip8| ALARMIFP) -> |SI_lo|) ((|ip9| SIGNALIFP) -> |Pulse_P|)
   ((|ip10| ALARMIFP) -> |ABPs_lo|)
   ((|ip11| ALARMIFP) -> |Pulse_P_lo|)
   ((|ip12| ALARMIFP) -> |CVP_lo|))
(RT-DATABASE) (PROCESSORS) (CONTROL)
(FAULT-MODES
   (|co_lo| (|ABPs_lo| |Pulse_P_lo| |CI_lo| |CVP_lo|) NIL NIL NIL))
(FAULT-GRAPH) (SIMULATION) (PANEL NIL) (SUBPROC))
```

/home/vandy/Designs/Resp2/Pml/o2cons.pml

The oxygen consumption process is modelled by the "o2cons" model.

```
;;; PML database: "o2cons.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |O2_consumption|.
(DEF-PROCESS |O2_consumption| (->) NIL
  ((|o2_cons_hi| NIL) (|o2_extr_hi| NIL)) NIL NIL NIL NIL NIL NIL NIL
  ((|Structural| (T |O2_consumption| 337 6))
   (|Monitoring+Control| (O |O2ER_hi| |OnlineAlarm| 255 92 255 80)
     (O |ip1| |InterfacePoint| 190 92 190 80)
     (O |ip2| |InterfacePoint| 190 136 191 125)
     (O |VO2I_hi| |OnlineAlarm| 256 132 257 120)
     (C |IO connection| (|ip1| NIL NIL NIL) (|O2ER_hi| NIL NIL NIL)
       |Green| 205 105 260 105)
     (C |IO connection| (|ip2| NIL NIL NIL) (|VO2I_hi| NIL NIL NIL)
       |Green| 208 145 261 144)
     (T |O2_consumption| 337 6))
   (|Simulation| (O |VO2I_hi| |OnlineAlarm| 256 132 257 120)
     (O |O2ER_hi| |OnlineAlarm| 255 92 255 80)
     (T |O2_consumption| 337 6))
   (|StateTransitions| (T |O2_consumption| 337 6))
   (|FailurePropagation|
     (O |o2_cons_hi| |FailureMode| 154 135 147 119)
     (O |o2_extr_hi| |FailureMode| 156 91 147 76)
     (O |O2ER_hi| |OnlineAlarm| 255 92 255 80)
     (O |VO2I_hi| |OnlineAlarm| 256 132 257 120)
     (C |Failure/Alarm connection| (|o2_extr_hi| NIL NIL NIL)
       (|O2ER_hi| NIL NIL NIL) |Green| 173 103 263 103)
     (C |Failure/Alarm connection| (|o2_cons_hi| NIL NIL NIL)
       (|VO2I_hi| NIL NIL NIL) |Green| 173 145 263 145)
     (T |O2_consumption| 337 6))
   (|OperatorInterface| (O |O2ER_hi| |OnlineAlarm| 255 92 255 80)
     (O |VO2I_hi| |OnlineAlarm| 256 132 257 120)
     (T |O2_consumption| 337 6)))
  ((|Structural| . "o2cons-str.icon")
   (|Monitoring+Control| . "o2cons-mon.icon") (|Simulation|)
   (|StateTransitions|) (|FailurePropagation| . "o2cons-fpg.icon")
   (|OperatorInterface|))
  (PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
  (PARAMETERS) (SIGNALS) (EVENTS)
  (ALARMS (|O2ER_hi| :ONLINE) (|VO2I_hi| :ONLINE))
  (INTERFACE ((|ip1| ALARMIFP) -> |O2ER_hi|)
    ((|ip2| ALARMIFP) -> |VO2I_hi|))
  (RT-DATABASE) (PROCESSORS) (CONTROL)
  (FAULT-MODES (|o2_cons_hi| (|VO2I_hi|) NIL NIL NIL)
    (|o2_extr_hi| (|O2ER_hi|) NIL NIL NIL))
```

(FAULT-GRAPH) (SIMULATION) (PANEL NIL) (SUBPROC))

/home/vandy/Designs/Resp2/Pml/oxyg.pml

The oxygenation process is represented by the "oxyg" model.

```
;;; PML database: "oxyg.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |oxygenation|.
(DEF-PROCESS |oxygenation| (->) NIL
  ((|hypoxemia| NIL) (|oxy_gen_f| NIL)) NIL NIL NIL NIL NIL NIL NIL
  ((|Structural| (O |Pul_circ| |pul_circ| 549 86 567 222)
     (O |Gasx| |gasx| 323 84 358 217)
     (O |Ventilation| |ventilation| 66 86 117 227)
     (T |oxygenation| 320 13))
   (|Monitoring+Control| (O |Pul_circ| |pul_circ| 576 559 593 538)
     (O |Gasx| |gasx| 312 563 342 543)
     (O |Ventilation| |ventilation| 62 572 74 536)
     (O |fio2| |Signal| 84 64 84 52)
     (O |pao2| |Signal| 84 104 84 92)
     (O |paco2| |Signal| 84 184 84 172)
     (O |sao2| |Signal| 84 144 84 132)
     (O |hypoxemia_al| |OnlineAlarm| 255 52 255 40)
     (O |AaDO2_hi_al| |OnlineAlarm| 255 92 255 80)
     (O |ip1| |InterfacePoint| 35 65 35 50)
     (O |ip2| |InterfacePoint| 35 105 35 90)
     (O |ip3| |InterfacePoint| 35 145 35 130)
     (O |ip4| |InterfacePoint| 35 184 35 170)
     (O |ip5| |InterfacePoint| 200 55 200 40)
     (O |ip6| |InterfacePoint| 200 95 200 80)
     (C |IO connection| (|ip1| NIL NIL NIL) (|fio2| NIL NIL NIL)
       |Green| 54 75 85 75)
     (C |IO connection| (|ip2| NIL NIL NIL) (|pao2| NIL NIL NIL)
       |Green| 54 115 85 115)
     (C |IO connection| (|ip3| NIL NIL NIL) (|sao2| NIL NIL NIL)
       |Green| 54 155 85 155)
     (C |IO connection| (|ip4| NIL NIL NIL) (|paco2| NIL NIL NIL)
       |Green| 54 195 85 195)
     (C |Dataflow| (|pao2| NIL NIL NIL)
       (|Ventilation| |pao2| HSIGNALS 0) |White| 103 118 140 118
       140 230 35 230 35 585 65 585 69 586)
     (C |IO connection| (|ip5| NIL NIL NIL)
       (|hypoxemia_al| NIL NIL NIL) |Green| 220 65 260 65)
     (C |Dataflow| (|sao2| NIL NIL NIL)
       (|Ventilation| |sao2| HSIGNALS 1) |White| 103 158 134 158
       134 222 27 222 27 600 65 600 69 600)
     (C |Dataflow| (|paco2| NIL NIL NIL)
       (|Ventilation| |paco2| HSIGNALS 2) |White| 103 198 127 198
       127 215 15 215 15 611 65 611 69 614)
     (C |IO connection| (|ip6| NIL NIL NIL)
```

```
    (|AaDO2_hi_al| NIL NIL NIL) |Green| 220 105 260 105)
  (T |oxygenation| 320 13))
(|Simulation| (O |AaDO2_hi_al| |OnlineAlarm| 255 92 255 80)
  (O |hypoxemia_al| |OnlineAlarm| 255 52 255 40)
  (O |sao2| |Signal| 84 144 84 132)
  (O |paco2| |Signal| 84 184 84 172)
  (O |pao2| |Signal| 84 104 84 92)
  (O |fio2| |Signal| 84 64 84 52) (T |oxygenation| 320 13))
(|StateTransitions| (T |oxygenation| 320 13))
(|FailurePropagation| (O |hypoxemia| |FailureMode| 95 60 95 48)
  (O |oxy_gen_f| |FailureMode| 95 100 95 88)
  (O |cl4| |CausalLink| 58 100 58 88)
  (O |cl3| |CausalLink| 71 292 69 275)
  (O |cl1| |CausalLink| 294 295 292 278)
  (O |cl2| |CausalLink| 705 302 704 285)
  (O |cl5| |CausalLink| 164 226 164 211)
  (O |hypoxemia_al| |OnlineAlarm| 185 60 185 48)
  (O |AaDO2_hi_al| |OnlineAlarm| 185 98 183 89)
  (O |Pul_circ| |pul_circ| 566 287 593 263)
  (O |Gasx| |gasx| 331 285 385 257)
  (O |Ventilation| |ventilation| 118 284 140 265)
  (O |cl6| |CausalLink| 707 338 704 326)
  (C |Failure/Alarm connection| (|hypoxemia| NIL NIL NIL)
    (|hypoxemia_al| NIL NIL NIL) |Green| 115 71 188 71)
  (C |FailurePropagation| (|Pul_circ| |shunt_hi| FAILS 1)
    (|cl2| NIL NIL NIL) |White| 681 316 710 316)
  (C |FailurePropagation| (|Gasx| |impaired_diff| FAILS 0)
    (|cl1| NIL NIL NIL) |White| 335 307 313 307)
  (C |FailurePropagation| (|cl1| NIL NIL NIL)
    (|oxy_gen_f| NIL NIL NIL) |White| 306 299 306 145 110 145
    110 117)
  (C |FailurePropagation| (|cl2| NIL NIL NIL)
    (|oxy_gen_f| NIL NIL NIL) |White| 718 306 718 135 116 135
    116 117)
  (C |FailurePropagation| (|cl3| NIL NIL NIL)
    (|oxy_gen_f| NIL NIL NIL) |White| 82 298 82 145 104 145 104
    117)
  (C |FailurePropagation| (|oxy_gen_f| NIL NIL NIL)
    (|cl4| NIL NIL NIL) |White| 104 110 76 110)
  (C |FailurePropagation| (|cl4| NIL NIL NIL)
    (|hypoxemia| NIL NIL NIL) |White| 70 103 70 72 101 72)
  (C |FailurePropagation| (|Ventilation| |vdot_a_lo| FAILS 4)
    (|cl3| NIL NIL NIL) |White| 127 301 89 301)
  (C |FailurePropagation| (|cl5| NIL NIL NIL)
    (|Ventilation| |vdot_a_lo| FAILS 4) |White| 174 240 105 240
    105 297 127 297)
  (C |FailurePropagation| (|Ventilation| |mv_lo| FAILS 0)
    (|cl5| NIL NIL NIL) |White| 246 299 244 301 262 301 262 240
    183 240)
  (C |Failure/Alarm connection| (|oxy_gen_f| NIL NIL NIL)
    (|AaDO2_hi_al| NIL NIL NIL) |Green| 113 112 191 112)
  (C |FailurePropagation| (|cl6| NIL NIL NIL)
```

```
      (|Pul_circ| |preload_lo| FAILS 2) |White| 714 346 700 346
      700 328 685 328)
    (C |FailurePropagation| (|Ventilation| |prp_hi| FAILS 3)
      (|cl6| NIL NIL NIL) |White| 248 403 267 403 267 471 720 471
      720 358)
    (T |oxygenation| 320 13))
 (|OperatorInterface| (O |fio2| |Signal| 84 64 84 52)
    (O |pao2| |Signal| 84 104 84 92)
    (O |paco2| |Signal| 84 184 84 172)
    (O |sao2| |Signal| 84 144 84 132)
    (O |hypoxemia_al| |OnlineAlarm| 255 52 255 40)
    (O |AaDO2_hi_al| |OnlineAlarm| 255 92 255 80)
    (O |oxyg_pnl| |oxyg| 342 205 355 310)
    (C |Display connection| (|sao2| NIL NIL NIL)
      (|oxyg_pnl| |s1| INSIGNALS 0) |White| 101 155 213 155 213
      217 351 217)
    (C |Alarm connection| (|AaDO2_hi_al| NIL NIL NIL)
      (|oxyg_pnl| |AaDO2_hi_al| ALARMS 1) |White| 274 105 283 105
      283 245 350 245)
    (C |Alarm connection| (|hypoxemia_al| NIL NIL NIL)
      (|oxyg_pnl| |hypoxemia_al| ALARMS 0) |White| 272 67 299 67
      299 228 346 228)
    (T |oxygenation| 320 13)))
((|Structural| . "oxyg_str.icon")
 (|Monitoring+Control| . "oxyg_mon.icon") (|Simulation|)
 (|StateTransitions|) (|FailurePropagation| . "oxyg_fpg.icon")
 (|OperatorInterface|))
(PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
(PARAMETERS)
(SIGNALS (|fio2| :DOUBLE NIL) (|pao2| :DOUBLE NIL)
      (|paco2| :DOUBLE NIL) (|sao2| :DOUBLE NIL))
(EVENTS) (ALARMS (|hypoxemia_al| :ONLINE) (|AaDO2_hi_al| :ONLINE))
(INTERFACE ((|ip1| SIGNALIFP) -> |fio2|)
  ((|ip2| SIGNALIFP) -> |pao2|) ((|ip3| SIGNALIFP) -> |sao2|)
  ((|ip4| SIGNALIFP) -> |paco2|)
  ((|ip5| ALARMIFP) -> |hypoxemia_al|)
  ((|ip6| ALARMIFP) -> |AaDO2_hi_al|))
(RT-DATABASE) (PROCESSORS)
(CONTROL (|Pul_circ| NIL NIL NIL NIL) (|Gasx| NIL NIL NIL NIL)
      (|Ventilation| (|pao2| |sao2| |paco2|) NIL NIL NIL))
(FAULT-MODES (|hypoxemia| (|hypoxemia_al|) NIL NIL NIL)
  (|oxy_gen_f| (|AaDO2_hi_al|) NIL NIL NIL))
(FAULT-GRAPH
  (((|Ventilation| |prp_hi|)) (|cl6| NIL 100 0 0)
   (|Pul_circ| |preload_lo|))
  (((|Ventilation| |mv_lo|)) (|cl5| NIL 100 0 0)
   (|Ventilation| |vdot_a_lo|))
  (((|Pul_circ| |shunt_hi|)) (|cl2| NIL 100 0 0)
   (|oxy_gen_f| NIL))
  (((|Gasx| |impaired_diff|)) (|cl1| NIL 100 0 0)
   (|oxy_gen_f| NIL))
  (((|Ventilation| |vdot_a_lo|)) (|cl3| NIL 100 0 0)
```

```
    (|oxy_gen_f| NIL))
   (((|oxy_gen_f| NIL)) (|cl4| NIL 100 0 0) (|hypoxemia| NIL)))
(SIMULATION)
(PANEL NIL
     (|oxyg_pnl|
       (|oxyg| (|sao2|) NIL NIL NIL
            (|hypoxemia_al| |AaDO2_hi_al|))))
(SUBPROC (|Pul_circ| (|pul_circ| (->) NIL))
       (|Gasx| (|gasx| (->) NIL))
       (|Ventilation| (|ventilation| (->) NIL))))
```

/home/vandy/Designs/Resp2/Pml/o2del.pml

The "o2del" model refers to oxygen delivery process.

```
;;; PML database: "o2del.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |O2_delivery|.
(DEF-PROCESS |O2_delivery| (->) NIL ((|do2_lo| NIL)) NIL NIL NIL NIL
  NIL NIL NIL
  ((|Structural| (O |Sys_circ| |sys_circ| 492 99 523 219)
     (O |O2_transport| |O2_transport| 280 97 287 234)
     (O |Oxygenation| |oxygenation| 69 98 86 235)
     (T |O2_delivery| 318 10))
   (|Monitoring+Control| (O |Sys_circ| |sys_circ| 542 345 552 316)
     (O |O2_transport| |O2_transport| 294 348 307 319)
     (O |Oxygenation| |oxygenation| 68 350 77 322)
     (O |O2AV_lo_al| |OnlineAlarm| 231 64 228 51)
     (O |ip1| |InterfacePoint| 154 63 154 53)
     (C |IO connection| (|ip1| NIL NIL NIL)
       (|O2AV_lo_al| NIL NIL NIL) |White| 174 72 235 73)
     (T |O2_delivery| 318 10))
   (|Simulation| (O |O2AV_lo_al| |OnlineAlarm| 231 64 228 51)
     (T |O2_delivery| 318 10))
   (|StateTransitions| (T |O2_delivery| 318 10))
   (|FailurePropagation| (O |do2_lo| |FailureMode| 115 78 112 55)
     (O |cl6| |CausalLink| 261 407 243 388)
     (O |cl5| |CausalLink| 276 371 263 357)
     (O |cl4| |CausalLink| 296 337 297 321)
     (O |cl3| |CausalLink| 220 304 219 283)
     (O |cl2| |CausalLink| 180 62 179 43)
     (O |cl1| |CausalLink| 181 109 182 101)
     (O |O2AV_lo_al| |OnlineAlarm| 57 78 21 57)
     (O |Sys_circ| |sys_circ| 585 290 623 265)
     (O |O2_transport| |O2_transport| 329 291 356 268)
     (O |Oxygenation| |oxygenation| 35 297 64 275)
     (C |FailurePropagation| (|cl2| NIL NIL NIL)
       (|do2_lo| NIL NIL NIL) |White| 189 75 165 75 165 87 136 87)
     (C |FailurePropagation| (|cl1| NIL NIL NIL)
       (|do2_lo| NIL NIL NIL) |White| 187 119 165 119 165 97 135
       97)
     (C |FailurePropagation| (|O2_transport| |cto2_lo| FAILS 0)
       (|cl1| NIL NIL NIL) |White| 334 316 318 316 318 120 196
       120)
     (C |FailurePropagation| (|Sys_circ| |co_lo| FAILS 0)
       (|cl2| NIL NIL NIL) |White| 594 347 574 347 574 70 198 70)
     (C |FailurePropagation| (|Oxygenation| |hypoxemia| FAILS 0)
       (|cl3| NIL NIL NIL) |White| 200 317 228 317)
     (C |FailurePropagation| (|O2_transport| |Dys_Hb_hi| FAILS 3)
```

```
      (|cl5| NIL NIL NIL) |White| 335 375 294 375)
    (C |FailurePropagation| (|O2_transport| |tHb_lo| FAILS 1)
      (|cl4| NIL NIL NIL) |White| 335 353 312 353)
    (C |FailurePropagation|
      (|O2_transport| |Hb_affinity_lo| FAILS 2)
      (|cl6| NIL NIL NIL) |White| 332 414 277 414)
    (C |FailurePropagation| (|cl3| NIL NIL NIL)
      (|O2_transport| |cto2_lo| FAILS 0) |White| 241 309 320 309
      320 320 334 320)
    (C |FailurePropagation| (|cl6| NIL NIL NIL)
      (|O2_transport| |cto2_lo| FAILS 0) |White| 273 412 272 320
      337 320)
    (C |FailurePropagation| (|cl5| NIL NIL NIL)
      (|O2_transport| |cto2_lo| FAILS 0) |White| 288 374 288 318
      336 318)
    (C |FailurePropagation| (|cl4| NIL NIL NIL)
      (|O2_transport| |cto2_lo| FAILS 0) |White| 307 340 307 314
      333 314)
    (C |Failure/Alarm connection| (|do2_lo| NIL NIL NIL)
      (|O2AV_lo_al| NIL NIL NIL) |White| 120 91 76 90)
    (T |O2_delivery| 318 10))
 (|OperatorInterface| (O |O2AV_lo_al| |OnlineAlarm| 231 64 228 51)
    (T |O2_delivery| 318 10)))
((|Structural| . "o2del_str.icon")
 (|Monitoring+Control| . "o2del_mon.icon") (|Simulation|)
 (|StateTransitions|) (|FailurePropagation| . "o2del_fpg.icon")
 (|OperatorInterface|))
(PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
(PARAMETERS) (SIGNALS) (EVENTS) (ALARMS (|O2AV_lo_al| :ONLINE))
(INTERFACE ((|ip1| ALARMIFP) -> |O2AV_lo_al|)) (RT-DATABASE)
(PROCESSORS)
(CONTROL (|Sys_circ| NIL NIL NIL NIL)
       (|O2_transport| NIL NIL NIL NIL)
       (|Oxygenation| NIL NIL NIL NIL))
(FAULT-MODES (|do2_lo| (|O2AV_lo_al|) NIL NIL NIL))
(FAULT-GRAPH
   (((|O2_transport| |cto2_lo|)) (|cl1| NIL 100 0 0)
    (|do2_lo| NIL))
   (((|Sys_circ| |co_lo|)) (|cl2| NIL 100 0 0) (|do2_lo| NIL))
   (((|Oxygenation| |hypoxemia|)) (|cl3| NIL 100 0 0)
    (|O2_transport| |cto2_lo|))
   (((|O2_transport| |tHb_lo|)) (|cl4| NIL 100 0 0)
    (|O2_transport| |cto2_lo|))
   (((|O2_transport| |Dys_Hb_hi|)) (|cl5| NIL 100 0 0)
    (|O2_transport| |cto2_lo|))
   (((|O2_transport| |Hb_affinity_lo|)) (|cl6| NIL 100 0 0)
    (|O2_transport| |cto2_lo|)))
(SIMULATION) (PANEL NIL)
(SUBPROC (|Sys_circ| (|sys_circ| (->) NIL))
       (|O2_transport| (|O2_transport| (->) NIL))
       (|Oxygenation| (|oxygenation| (->) NIL))))
```

/home/vandy/Designs/Resp2/Pml/resp.pml

The "resp" process model refers to the top level in the process hierarchy.

```
;;; PML database: "resp.pml"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Pml/")
;;; Definition for |Resp|.
(DEF-PROCESS |Resp| (->) NIL ((|hypoxia| NIL)) NIL NIL NIL NIL NIL NIL
  NIL
  ((|Structural|
     (O |O2_consumption| |O2_consumption| 480 291 480 257)
     (O |O2_delivery| |O2_delivery| 145 289 153 253)
     (T |Resp| 324 23))
   (|Monitoring+Control|
     (O |O2_consumption| |O2_consumption| 480 291 480 257)
     (O |O2_delivery| |O2_delivery| 145 289 153 253)
     (O |pvo2| |Signal| 85 42 85 30)
     (O |pvo2_lo| |OnlineAlarm| 235 42 235 30)
     (O |ip1| |InterfacePoint| 15 42 15 30)
     (O |ip2| |InterfacePoint| 165 42 165 30)
     (C |IO connection| (|ip1| NIL NIL NIL) (|pvo2| NIL NIL NIL)
       |Green| 35 52 90 52)
     (C |IO connection| (|ip2| NIL NIL NIL) (|pvo2_lo| NIL NIL NIL)
       |Green| 185 52 240 52)
     (T |Resp| 324 23))
   (|Simulation| (O |pvo2_lo| |OnlineAlarm| 207 63 205 54)
     (O |pvo2| |Signal| 94 61 93 52) (T |Resp| 324 23))
   (|StateTransitions| (T |Resp| 324 23))
   (|FailurePropagation| (O |hypoxia| |FailureMode| 85 62 85 50)
     (O |cl2| |CausalLink| 395 372 395 360)
     (O |cl1| |CausalLink| 324 373 323 360)
     (O |pvo2_lo| |OnlineAlarm| 165 62 165 50)
     (O |O2_consumption| |O2_consumption| 481 370 506 333)
     (O |O2_delivery| |O2_delivery| 145 367 166 342)
     (C |Failure/Alarm connection| (|hypoxia| NIL NIL NIL)
       (|pvo2_lo| NIL NIL NIL) |Green| 103 75 170 75)
     (C |FailurePropagation| (|O2_delivery| |do2_lo| FAILS 0)
       (|cl1| NIL NIL NIL) |Green| 283 385 284 385 326 385)
     (C |FailurePropagation| (|cl1| NIL NIL NIL)
       (|hypoxia| NIL NIL NIL) |Green| 336 379 336 332 95 332 95
       78)
     (C |FailurePropagation|
       (|O2_consumption| |o2_cons_hi| FAILS 0) (|cl2| NIL NIL NIL)
       |Green| 488 381 480 382 412 382)
     (C |FailurePropagation| (|cl2| NIL NIL NIL)
       (|hypoxia| NIL NIL NIL) |Green| 409 377 409 314 100 314 100
       78)
     (T |Resp| 324 23))
```

```
(|OperatorInterface| (O |pvo2| |Signal| 94 61 93 52)
   (O |pvo2_lo| |OnlineAlarm| 207 63 205 54)
   (O |resp| |Respiration| 335 201 341 320)
   (C |Display connection| (|pvo2| NIL NIL NIL)
     (|resp| |s1| INSIGNALS 0) |White| 104 82 104 211 341 211)
   (C |Alarm connection| (|pvo2_lo| NIL NIL NIL)
     (|resp| |pvo2_lo| ALARMS 0) |White| 219 83 219 226 341 226)
   (T |Resp| 324 23)))
((|Structural| . "resp_str.icon")
 (|Monitoring+Control| . "resp_mon.icon") (|Simulation|)
 (|StateTransitions|) (|FailurePropagation| . "resp_fpg.icon")
 (|OperatorInterface|))
(PVARS) (PVAR-ATTRS) (STATES) (STATE-DEPS) (TRANSITIONS)
(PARAMETERS) (SIGNALS (|pvo2| :DOUBLE NIL)) (EVENTS)
(ALARMS (|pvo2_lo| :ONLINE))
(INTERFACE ((|ip1| VSSIGNALIFP) -> |pvo2|)
   ((|ip2| ALARMIFP) -> |pvo2_lo|))
(RT-DATABASE) (PROCESSORS)
(CONTROL (|O2_consumption| NIL NIL NIL NIL)
      (|O2_delivery| NIL NIL NIL NIL))
(FAULT-MODES (|hypoxia| (|pvo2_lo|) NIL NIL NIL))
(FAULT-GRAPH
   ((((|O2_delivery| |do2_lo|)) (|cl1| NIL 100 0 0)
    (|hypoxia| NIL))
   ((((|O2_consumption| |o2_cons_hi|)) (|cl2| NIL 100 0 0)
    (|hypoxia| NIL)))
(SIMULATION)
(PANEL NIL
     (|resp| (|Respiration| (|pvo2|) NIL NIL NIL (|pvo2_lo|))))
(SUBPROC (|O2_consumption| (|O2_consumption| (->) NIL))
      (|O2_delivery| (|O2_delivery| (->) NIL))))
```

F.6.2 Panel Aspect

The models within this section describe the user interface panel for individual processes.

/home/vandy/Designs/Resp2/Panel/resp.prj

The "resp.prj" project file contains a list of the panel databases.

```
;;; Project file: |resp.prj|
(DEF-PROJECT
   (|resp| "/users/vandy/Designs/Resp2/Panel/"
      "/users/vandy/Icons/Resp2/Panel/")
      "resp.pnl" "vent.pnl" "pul_circ.pnl" "oxyg.pnl"  )
```

/home/vandy/Designs/Resp2/Panel/resp.pnl

This file defines the user interface for the process "resp".

```
;;; PANEL database: "resp.pnl"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Panel/")
;;; Definition for |Respiration|.
(DEF-PANEL-TYPE |Respiration| ((|s1| :DOUBLE)) NIL NIL NIL (|PvO2_lo|)
  ((|OperatorInterface| . "resp.icon"))
  (DISPLAY (|gd1| |GraphDisplay| ((|s1| |gdl1| |Blue| SOLID))
             "PvO2 (kPa)" 60 200 0.0 10.0)
       (|ld1| |LogDisplay| |s1| "PvO2 (kPa)"))
  (CONTROL) (ICONS)
  (PICTURE (|OperatorInterface| (O |s1| |Signal| 69 80 71 69)
           (O |PvO2_lo| |Alarm| 774 26 730 5)
           (O |gd1| |GraphDisplay| 144 77 144 68)
           (O |gdl1| |GraphDisplayLink| 107 81 105 72)
           (O |ld1| |LogDisplay| 846 132 847 125)
           (L "PvO2 (kPa)" 144 59 "9x15" |White|)
           (L "PvO2 (kPa)" 846 115 "9x15" |White|)
           (C |Display connection| (|s1| NIL NIL NIL)
             (|gdl1| NIL NIL NIL) |White| 83 91 114 91)
           (C |Display connection| (|gdl1| NIL NIL NIL)
             (|gd1| NIL NIL NIL) |White| 123 91 150 91)
           (C |Display connection| (|s1| NIL NIL NIL)
             (|ld1| NIL NIL NIL) |White| 80 98 80 151 853 151)
           (T |Respiration| 334 20))))
```

/home/vandy/Designs/Resp2/Panel/vent.pnl

This file defines the panel for the ventilation process.

```
;;; PANEL database: "vent.pnl"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Panel/")
;;; Definition for |vent|.
(DEF-PANEL-TYPE |vent|
   ((|vte| :DOUBLE) (|f| :DOUBLE) (|mv| :DOUBLE) (|peak| :DOUBLE)
    (|plat| :DOUBLE) (|peep| :DOUBLE) (|pmin| :DOUBLE)
    (|mean| :DOUBLE) (|fio2| :DOUBLE) (|c| :DOUBLE) (|r| :DOUBLE)
    (|mvs| :DOUBLE) (|fs| :DOUBLE))
   NIL NIL
   ((|ev1| :DOUBLE) (|ev2| :DOUBLE) (|ev3| :DOUBLE) (|ev4| :DOUBLE))
   (|hypo_ven| |hyper_ven| |FIO2_hi| |PRP_hi| |ALVENT_lo| |Resp_acid|
      |Resp_alka|)
   ((|OperatorInterface| . "vent.icon"))
   (DISPLAY (|gd1| |GraphDisplay| ((|vte| |gdl1| |Blue| SOLID))
               "Vte (l)" 10 200 0.0 2.0)
         (|gd2| |GraphDisplay| ((|f| |gdl2| |Red| SOLID))
               "f (1/min)" 10 200 0.0 60.0)
         (|gd3| |GraphDisplay| ((|mv| |gdl3| |Blue| SOLID))
               "MV (l/min)" 10 200 0.0 41.0)
         (|gd4| |GraphDisplay| ((|peak| |gdl4| |Red| SOLID))
               "Peak (mbar)" 10 200 0.0 99.0)
         (|gd5| |GraphDisplay| ((|plat| |gdl5| |Blue| SOLID))
               "Plat (mbar)" 10 200 0.0 99.0)
         (|gd6| |GraphDisplay| ((|peep| |gdl6| |Blue| SOLID))
               "PEEP (mbar)" 10 200 0.0 99.0)
         (|gd7| |GraphDisplay| ((|pmin| |gdl7| |Blue| SOLID))
               "Pmin (mbar)" 10 200 -20.0 99.0)
         (|gd8| |GraphDisplay| ((|mean| |gdl8| |Blue| SOLID))
               "Mean (mbar)" 10 200 0.0 99.0)
         (|gd9| |GraphDisplay| ((|fio2| |gdl9| |Blue| SOLID))
               "FIO2 (%)" 10 200 15.0 99.0)
         (|gd10| |GraphDisplay| ((|c| |gdl10| |Blue| SOLID))
               "C (ml/mbar)" 10 200 0.0 255.0)
         (|gd11| |GraphDisplay| ((|r| |gdl11| |Blue| SOLID))
               "R (mbar/(l/s))" 10 200 0.0 99.0)
         (|gd12| |GraphDisplay| ((|mvs| |gdl12| |Blue| SOLID))
               "MVs (l/min)" 10 200 0.0 41.0)
         (|gd13| |GraphDisplay| ((|fs| |gdl13| |Blue| SOLID))
               "fs (1/min)" 10 200 0.0 60.0)
         (|bd1| |BarDisplay| |fio2| "FIO2 (%)" 15.0 99.0))
   (CONTROL (|sl1| |Slider| |ev1| "hypo_v" 2.0 10.0 6.0)
         (|sl2| |Slider| |ev2| "fio2_t" 30.0 100.0 38.0)
         (|sl3| |Slider| |ev3| "sao2_t" 85.0 100.0 93.0)
         (|sl4| |Slider| |ev4| "hyper_v" 3.0 15.0 8.0))
```

```
(ICONS)
(PICTURE (|OperatorInterface| (O |vte| |Signal| 150 30 150 10)
        (O |f| |Signal| 150 80 150 60)
        (O |mv| |Signal| 150 130 150 110)
        (O |peak| |Signal| 150 180 150 160)
        (O |plat| |Signal| 150 230 150 210)
        (O |peep| |Signal| 150 280 150 260)
        (O |pmin| |Signal| 150 330 150 310)
        (O |mean| |Signal| 150 380 150 360)
        (O |fio2| |Signal| 150 430 150 410)
        (O |c| |Signal| 150 480 150 460)
        (O |r| |Signal| 150 530 150 510)
        (O |mvs| |Signal| 150 580 150 560)
        (O |fs| |Signal| 150 630 150 610)
        (O |ev1| |Event| 240 39 240 65)
        (O |ev2| |Event| 240 394 240 381)
        (O |ev3| |Event| 240 448 240 431)
        (O |ev4| |Event| 240 98 240 121)
        (O |hypo_ven| |Alarm| 795 30 778 15)
        (O |hyper_ven| |Alarm| 739 31 691 15)
        (O |FIO2_hi| |Alarm| 797 69 789 61)
        (O |PRP_hi| |Alarm| 742 70 702 57)
        (O |ALVENT_lo| |Alarm| 798 114 787 101)
        (O |sl1| |Slider| 190 29 190 15)
        (O |sl2| |Slider| 190 380 190 368)
        (O |sl3| |Slider| 190 433 190 421)
        (O |sl4| |Slider| 189 87 189 77)
        (O |gd1| |GraphDisplay| 20 30 20 10)
        (O |gd2| |GraphDisplay| 20 80 20 60)
        (O |gd3| |GraphDisplay| 20 130 20 110)
        (O |gd4| |GraphDisplay| 20 180 20 160)
        (O |gd5| |GraphDisplay| 20 230 20 210)
        (O |gd6| |GraphDisplay| 20 280 20 260)
        (O |gd7| |GraphDisplay| 20 330 20 310)
        (O |gd8| |GraphDisplay| 20 380 20 360)
        (O |gd9| |GraphDisplay| 20 430 20 410)
        (O |gd10| |GraphDisplay| 20 480 20 460)
        (O |gd11| |GraphDisplay| 20 530 20 510)
        (O |gd12| |GraphDisplay| 20 580 20 560)
        (O |gd13| |GraphDisplay| 20 630 20 610)
        (O |bd1| |BarDisplay| 783 430 755 430)
        (O |gdl13| |GraphDisplayLink| 100 630 100 610)
        (O |gdl12| |GraphDisplayLink| 100 580 100 560)
        (O |gdl11| |GraphDisplayLink| 100 530 100 510)
        (O |gdl10| |GraphDisplayLink| 100 480 100 460)
        (O |gdl9| |GraphDisplayLink| 100 430 100 410)
        (O |gdl8| |GraphDisplayLink| 100 380 100 360)
        (O |gdl7| |GraphDisplayLink| 100 330 100 310)
        (O |gdl6| |GraphDisplayLink| 100 280 100 260)
        (O |gdl5| |GraphDisplayLink| 100 230 100 210)
        (O |gdl4| |GraphDisplayLink| 100 180 100 160)
        (O |gdl3| |GraphDisplayLink| 100 130 100 110)
```

```
(O |gdl2| |GraphDisplayLink| 100 80 100 60)
(O |gdl1| |GraphDisplayLink| 100 30 100 10)
(O |Resp_acid| |Alarm| 801 156 784 141)
(O |Resp_alka| |Alarm| 803 199 782 184)
(L "Vte (l)" 21 10 "9x15" |White|)
(L "f (1/min)" 21 60 "9x15" |White|)
(L "MV (l/min)" 21 110 "9x15" |White|)
(L "Peak (mbar)" 21 160 "9x15" |White|)
(L "Plat (mbar)" 21 210 "9x15" |White|)
(L "PEEP (mbar)" 21 260 "9x15" |White|)
(L "Pmin (mbar)" 21 310 "9x15" |White|)
(L "Mean (mbar)" 21 360 "9x15" |White|)
(L "FIO2 (%)" 21 410 "9x15" |White|)
(L "FIO2 (%)" 783 409 "9x15" |White|)
(L "C (ml/mbar)" 21 460 "9x15" |White|)
(L "R (mbar/(l/s))" 21 510 "9x15" |White|)
(L "MVs (l/min)" 21 560 "9x15" |White|)
(L "fs (1/min)" 21 610 "9x15" |White|)
(C |Display connection| (|vte| NIL NIL NIL)
  (|gdl1| NIL NIL NIL) |White| 155 43 110 43)
(C |Display connection| (|f| NIL NIL NIL)
  (|gdl2| NIL NIL NIL) |White| 155 93 110 93)
(C |Display connection| (|gdl2| NIL NIL NIL)
  (|gd2| NIL NIL NIL) |White| 104 93 60 93)
(C |Display connection| (|gdl1| NIL NIL NIL)
  (|gd1| NIL NIL NIL) |White| 104 43 60 43)
(C |Display connection| (|mv| NIL NIL NIL)
  (|gdl3| NIL NIL NIL) |White| 155 143 110 143)
(C |Display connection| (|gdl3| NIL NIL NIL)
  (|gd3| NIL NIL NIL) |White| 104 143 60 143)
(C |Display connection| (|peak| NIL NIL NIL)
  (|gdl4| NIL NIL NIL) |White| 155 193 110 193)
(C |Display connection| (|gdl4| NIL NIL NIL)
  (|gd4| NIL NIL NIL) |White| 104 193 60 193)
(C |Display connection| (|plat| NIL NIL NIL)
  (|gdl5| NIL NIL NIL) |White| 155 243 110 243)
(C |Display connection| (|gdl5| NIL NIL NIL)
  (|gd5| NIL NIL NIL) |White| 104 243 60 243)
(C |Display connection| (|peep| NIL NIL NIL)
  (|gdl6| NIL NIL NIL) |White| 155 293 110 293)
(C |Display connection| (|gdl6| NIL NIL NIL)
  (|gd6| NIL NIL NIL) |White| 104 293 60 293)
(C |Display connection| (|pmin| NIL NIL NIL)
  (|gdl7| NIL NIL NIL) |White| 155 343 110 343)
(C |Display connection| (|gdl7| NIL NIL NIL)
  (|gd7| NIL NIL NIL) |White| 104 343 60 343)
(C |Display connection| (|mean| NIL NIL NIL)
  (|gdl8| NIL NIL NIL) |White| 155 393 110 393)
(C |Display connection| (|gdl8| NIL NIL NIL)
  (|gd8| NIL NIL NIL) |White| 104 393 60 393)
(C |Display connection| (|fio2| NIL NIL NIL)
  (|gdl9| NIL NIL NIL) |White| 155 443 110 443)
```

```
(C |Display connection| (|gdl9| NIL NIL NIL)
  (|gd9| NIL NIL NIL) |White| 104 443 60 443)
(C |Display connection| (|c| NIL NIL NIL)
  (|gdl10| NIL NIL NIL) |White| 155 493 110 493)
(C |Display connection| (|gdl10| NIL NIL NIL)
  (|gd10| NIL NIL NIL) |White| 104 493 60 493)
(C |Display connection| (|r| NIL NIL NIL)
  (|gdl11| NIL NIL NIL) |White| 155 543 110 543)
(C |Display connection| (|gdl11| NIL NIL NIL)
  (|gd11| NIL NIL NIL) |White| 104 543 60 543)
(C |Display connection| (|mvs| NIL NIL NIL)
  (|gdl12| NIL NIL NIL) |White| 155 593 110 593)
(C |Display connection| (|gdl12| NIL NIL NIL)
  (|gd12| NIL NIL NIL) |White| 104 593 60 593)
(C |Display connection| (|fs| NIL NIL NIL)
  (|gdl13| NIL NIL NIL) |White| 155 643 110 643)
(C |Display connection| (|gdl13| NIL NIL NIL)
  (|gd13| NIL NIL NIL) |White| 104 643 60 643)
(C |Control connection| (|sl1| NIL NIL NIL)
  (|ev1| NIL NIL NIL) |White| 205 50 250 50)
(C |Control connection| (|sl3| NIL NIL NIL)
  (|ev3| NIL NIL NIL) |White| 205 453 250 453)
(C |Control connection| (|sl2| NIL NIL NIL)
  (|ev2| NIL NIL NIL) |White| 205 400 250 400)
(C |Display connection| (|fio2| NIL NIL NIL)
  (|bd1| NIL NIL NIL) |White| 167 443 793 443)
(C |Control connection| (|sl4| NIL NIL NIL)
  (|ev4| NIL NIL NIL) |White| 206 111 244 111)
(T |vent| 383 5))))
```

/home/vandy/Designs/Resp2/Panel/pul_circ.pnl

This file specifies the user interface panel for the pulmonary circulation process.

```
;;; PANEL database: "pul_circ.pnl"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Panel/")
;;; Definition for |pul_circulation|.
(DEF-PANEL-TYPE |pul_circulation|
   ((|s1| :FLOAT) (|s2| :FLOAT) (|s3| :FLOAT)) NIL NIL NIL
   (|Qs/Qt_hi| |pul_hi_t| |PAPd-PAWP_hi| |PAWP_hi|)
   ((|OperatorInterface| . "pul_circ.icon"))
   (DISPLAY (|gd1| |GraphDisplay| ((|s1| |gdl1| |Blue| SOLID))
            "PAPm (mmHg)" 2 200 0.0 100.0))
   (CONTROL) (ICONS)
   (PICTURE (|OperatorInterface| (O |s1| |Signal| 69 80 71 69)
           (O |s2| |Signal| 70 131 70 118)
           (O |s3| |Signal| 70 175 73 166)
           (O |Qs/Qt_hi| |Alarm| 774 26 730 5)
           (O |pul_hi_t| |Alarm| 776 78 737 59)
           (O |PAPd-PAWP_hi| |Alarm| 778 122 738 105)
           (O |gd1| |GraphDisplay| 144 77 144 68)
           (O |gdl1| |GraphDisplayLink| 107 81 105 72)
           (O |PAWP_hi| |Alarm| 782 167 744 150)
           (L "PAPm (mmHg)" 144 59 "9x15" |White|)
           (C |Display connection| (|s1| NIL NIL NIL)
             (|gdl1| NIL NIL NIL) |White| 83 91 114 91)
           (C |Display connection| (|gdl1| NIL NIL NIL)
             (|gd1| NIL NIL NIL) |White| 123 91 150 91)
           (T |pul_circulation| 334 20))))
```

/home/vandy/Designs/Resp2/Panel/oxyg.pnl

This file defines the user interface panel for the oxygenation process.

```
;;; PANEL database: "oxyg.pnl"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Panel/")
;;; Definition for |oxyg|.
(DEF-PANEL-TYPE |oxyg| ((|s1| :DOUBLE)) NIL NIL NIL
  (|hypoxemia| |AaDO2_hi|) ((|OperatorInterface| . "oxyg.icon"))
  (DISPLAY (|gd1| |GraphDisplay| ((|s1| |gdl1| |Blue| SOLID)) "Sao2 (%)" 2
           200 60.0 100.0))
  (CONTROL) (ICONS)
  (PICTURE (|OperatorInterface| (O |s1| |Signal| 69 80 71 69)
           (O |hypoxemia| |Alarm| 774 26 730 5)
           (O |AaDO2_hi| |Alarm| 776 78 737 59)
           (O |gd1| |GraphDisplay| 144 77 144 68)
           (O |gdl1| |GraphDisplayLink| 107 81 105 72)
           (L "SaO2 (%)" 144 59 "9x15" |White|)
           (C |Display connection| (|s1| NIL NIL NIL)
             (|gdl1| NIL NIL NIL) |White| 83 91 114 91)
           (C |Display connection| (|gdl1| NIL NIL NIL)
             (|gd1| NIL NIL NIL) |White| 123 91 150 91)
           (T |oxyg| 334 20))))
```

F.6.3 Physical Aspect

The files in this section contain the information about the monitored system from a physical perspective.

/home/vandy/Designs/Resp2/Physical/resp.prj

This is the project file for the Physical aspect.

```
;;; Project file: |resp.prj|
(DEF-PROJECT
    (|resp| "/users/vandy/Designs/Resp2/Physical/"
        "/users/vandy/Icons/Resp2/Physical/")
    "cardio.phy" "ventilator.phy" "resp.phy"
)
```

/home/vandy/Designs/Resp2/Physical/cardio.phy

This file describes the "cardio" physical system referring to the cardiovascular system.

```
;;; PHY database: "cardio.phy"
(DEF-ICONLIB "/hdskgb/home/vandy/Icons/Resp2/Physical/")
;;; Definition for |cardio|.
(DEF-PHYSICAL-COMPONENT |cardio| NIL NIL
  ((|Structural| . "cardio.icon"))
  ((|Structural| (O |general failure| |FailureState| 219 324 175 384)
     (T |cardio| 351 473)))
  (FAILURE-STATES (|general failure| NIL)) (PARTS))
```

/home/vandy/Designs/Resp2/Physical/ventilator.phy

This file describes the ventilator physical component.

```
;;; PHY database: "ventilator.phy"
(DEF-ICONLIB "/hdskgb/home/vandy/Icons/Resp2/Physical/")
;;; Definition for |ventilator|.
(DEF-PHYSICAL-COMPONENT |ventilator| NIL NIL
  ((|Structural| . "ventilator.icon"))
  ((|Structural| (O |general failure| |FailureState| 338 259 300 299)
     (T |ventilator| 376 431)))
(FAILURE-STATES (|general failure| NIL)) (PARTS))
```

/home/vandy/Designs/Resp2/Physical/resp.phy

This file describes "resp", the top level physical system.

```
;;; PHY database: "resp.phy"
(DEF-ICONLIB "/hdskgb/home/vandy/Icons/Resp2/Physical/")
;;; Definition for |resp-phy|.
(DEF-PHYSICAL-COMPONENT |resp-phy| NIL NIL ((|Structural| . "resp.icon"))
  ((|Structural| (O |cardio| |cardio| 222 374 210 431)
     (O |vent| |ventilator| 100 370 95 426) (T |resp| 407 564)))
  (FAILURE-STATES)
  (PARTS (|cardio| |cardio|) (|vent| |ventilator|)))
```

F.6.4 Processors Aspect

The model files in this section define the processing blocks used in the application. The processors wrap the processing blocks defined in the Hierarchical Description Language (HDL) for use by the IPCS. The "resp.prj" file contains the name of processors databases (*.prc). Each database contains at least one processor (DEF-PROCESSOR) and the corresponding HDL structures as primitives (DEFPRIMITIVE) or compounds (DEFCOMPOUND). The script for the HDL structures are in the file "resp.c", which is compiled using the "makefile" and the corresponding objects are loaded into multigraph kernel using the "load.lsp" lisp file, during the IPCS initialisations.

/home/vandy/Designs/Resp2/Processors/resp.prj

The first file in this section is the "resp.prj" project file.

```
;;; Project file: |resp.prj|
(DEF-PROJECT
   (|resp| "/users/vandy/Designs/Resp2/Processors/"
      "/users/vandy/Icons/Resp2/Processors/")
      "alarm2.prc" "alarm3.prc" "alarm4.prc")
```

/home/vandy/Designs/Resp2/Processors/alarm2.prc

This is the processor database file for the alarm generator which uses two signals with two "hi" thresholds and the logical AND combination of the two (section 6.3.1).

```
;;; Procs database: "alarm2.prc"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Processors/")
;;; Definition for |Alarm2P|.
(DEF-PROCESSOR |Alarm2P|
  ((((|Signal1| :DOUBLE) (|Signal2| :DOUBLE))
   ((|Threshold1| :DOUBLE) (|Threshold2| :DOUBLE)) NIL -> NIL NIL
   ((|out| :ONLINE)))
  NIL NIL NIL ((|Monitoring+Control| . "a2procmon.icon"))
  ((|Monitoring+Control| (O |Signal1| |Signal| 260 200 193 200)
     (O |Signal2| |Signal| 260 245 193 253)
     (O |Threshold1| |Event| 260 279 166 280)
     (O |Threshold2| |Event| 260 315 161 317)
     (O |out| |Alarm| 605 238 598 218)
     (O |Alarm2| |@Alarm2| 434 222 440 294)
     (C |Dataflow| (|Signal1| NIL NIL NIL)
       (|Alarm2| |Signal1| INS 0) |White| 268 211 403 211 403 230
        445 230)
     (C |Dataflow| (|Signal2| NIL NIL NIL)
       (|Alarm2| |Signal2| INS 1) |White| 275 258 406 258 406 240
       445 240)
     (C |Dataflow| (|Threshold1| NIL NIL NIL)
       (|Alarm2| |Threshold1| INS 2) |White| 276 289 402 289 402
       271 441 271)
     (C |Dataflow| (|Threshold2| NIL NIL NIL)
       (|Alarm2| |Threshold2| INS 3) |White| 278 324 414 324 414
       281 443 281)
     (C |Dataflow| (|Alarm2| |Output| OUTS 0) (|out| NIL NIL NIL)
       |White| 485 249 612 249)
     (T |Alarm2P| 398 2)))
  (|Alarm2|
    (|@Alarm2|
      (|Signal1| |Signal2| |Threshold1| |Threshold2| -> |out|)
      NIL NIL NIL NIL)))
;;; Definition for |Alarm2|.
(DEFPRIMITIVE |Alarm2| :IFANY
  ((|signal1| :STREAM) (|signal2| :STREAM) (|threshold1| :STREAM)
   (|threshold2| :STREAM) -> (|outalarm| :STREAM))
  ((|T1Context| 0.0 :DOUBLE) (|T2Context| 0.0 :DOUBLE)
   (|ASContext| 0 :INT))
  NIL NIL NIL ((|Monitoring+Control| . "a2pmon.icon"))
```

```
   ((|Monitoring+Control| (O |signal1| |Stream| 164 230 100 230)
      (O |signal2| |Stream| 164 280 100 280)
      (O |threshold1| |Stream| 164 330 68 330)
      (O |threshold2| |Stream| 164 380 70 380)
      (O |outalarm| |Stream| 404 260 432 262)
      (O |T1Context| |StaticParameter| 164 80 71 80)
      (O |T2Context| |StaticParameter| 164 130 66 130)
      (O |ASContext| |StaticParameter| 164 180 66 180)
      (T |Alarm1| 413 5)))
   "alarm2es")
;;; Definition for |@Alarm2|.
(DEFCOMPOUND |@Alarm2|
   ((|Signal1| :STREAM) (|Signal2| :STREAM) (|Threshold1| :STREAM)
    (|Threshold2| :STREAM) -> (|Output| :STREAM))
   NIL NIL NIL NIL ((|Monitoring+Control| . "a2compmon.icon"))
   ((|Monitoring+Control| (O |Signal1| |Stream| 200 200 113 200)
      (O |Signal2| |Stream| 200 240 113 240)
      (O |Threshold1| |Stream| 200 280 110 280)
      (O |Threshold2| |Stream| 200 320 107 320)
      (O |Output| |Stream| 606 239 634 241)
      (O |T1Context| |StaticParameter| 200 80 120 80)
      (O |T2Context| |StaticParameter| 200 120 121 120)
      (O |ASContext| |StaticParameter| 200 160 121 160)
      (O |Alarm2| |Alarm2| 417 219 414 290)
      (C |Signal connection| (|Signal1| NIL NIL NIL)
        (|Alarm2| |signal1| INS 0) |White| 209 209 366 209 366 227
        426 227)
      (C |Signal connection| (|Signal2| NIL NIL NIL)
        (|Alarm2| |signal2| INS 1) |White| 206 247 367 247 367 236
        424 236)
      (C |Signal connection| (|Threshold1| NIL NIL NIL)
        (|Alarm2| |threshold1| INS 2) |White| 207 287 363 287 363
        267 423 267)
      (C |Signal connection| (|Threshold2| NIL NIL NIL)
        (|Alarm2| |threshold2| INS 3) |White| 205 328 376 328 376
        278 424 278)
      (C |Par connection| (|T1Context| NIL NIL NIL)
        (|Alarm2| |T1Context| PARS 0) |White| 220 89 458 89 458
        224)
      (C |Par connection| (|T2Context| NIL NIL NIL)
        (|Alarm2| |T2Context| PARS 1) |White| 215 132 465 132 465
        224)
      (C |Par connection| (|ASContext| NIL NIL NIL)
        (|Alarm2| |ASContext| PARS 2) |White| 217 172 473 172 473
        224)
      (C |Signal connection| (|Alarm2| |outalarm| OUTS 0)
        (|Output| NIL NIL NIL) |White| 470 246 612 246)
      (T |@Alarm2| 403 9)))
   (SIGNALS)
   (PARAMS (|T1Context| 95.0 :DOUBLE) (|T2Context| 50.0 :DOUBLE)
      (|ASContext| 0 :INT))
   (SHARED) (VARS) (COMPUTE)
```

```
(STRUCT (|Alarm2|
        (|Alarm2|
          (|Signal1| |Signal2| |Threshold1| |Threshold2| ->
            |Output|)
          (|T1Context| |T2Context| |ASContext|) NIL NIL NIL))))
```

/home/vandy/Designs/Resp2/Processors/alarm3.prc

The database file for alarm generator "alarm3", which is a single signal low threshold alarm.

```
;;; Procs database: "alarm3.prc"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Processors/")
;;; Definition for |Alarm3|.
(DEFPRIMITIVE |Alarm3| :IFANY
   ((|signal| :STREAM) (|threshold| :STREAM) -> (|outalarm| :STREAM))
   ((|T1Context| 0.0 :DOUBLE) (|ASContext| 0 :INT)) NIL NIL NIL
   ((|Monitoring+Control| . "a3pmon.icon"))
   ((|Monitoring+Control| (O |signal| |Stream| 164 273 104 274)
      (O |threshold| |Stream| 163 314 68 315)
      (O |outalarm| |Stream| 404 260 432 262)
      (O |T1Context| |StaticParameter| 159 175 71 178)
      (O |ASContext| |StaticParameter| 159 219 66 222)
      (T |Alarm3| 413 5)))
   "alarm3es")
;;; Definition for |@Alarm3|.
(DEFCOMPOUND |@Alarm3|
   ((|signal| :STREAM) (|threshold| :STREAM) -> (|Output| :STREAM))
   NIL NIL NIL NIL ((|Monitoring+Control| . "a3compmon.icon"))
   ((|Monitoring+Control| (O |signal| |Stream| 204 244 113 245)
      (O |threshold| |Stream| 200 322 107 322)
      (O |Output| |Stream| 606 239 634 241)
      (O |ASContext| |StaticParameter| 199 152 120 163)
      (O |T1Context| |StaticParameter| 199 108 120 111)
      (O |Alarm3| |Alarm3| 417 219 414 290)
      (C |Signal connection| (|threshold| NIL NIL NIL)
        (|Alarm3| |threshold| INS 1) |White| 211 329 375 329 375
        277 423 277)
      (C |Signal connection| (|signal| NIL NIL NIL)
        (|Alarm3| |signal| INS 0) |Orange| 215 256 425 256)
      (C |Par connection| (|ASContext| NIL NIL NIL)
        (|Alarm3| |ASContext| PARS 1) |Green| 215 165 325 165 325
        233 427 233)
      (C |Par connection| (|T1Context| NIL NIL NIL)
        (|Alarm3| |T1Context| PARS 0) |Green| 216 116 337 116 337
        226 420 226)
      (C |Signal connection| (|Alarm3| |outalarm| OUTS 0)
        (|Output| NIL NIL NIL) |Green| 472 247 611 247)
      (T |@Alarm3| 403 9)))
   (SIGNALS) (PARAMS (|T1Context| 8.0 :DOUBLE) (|ASContext| 0 :INT))
   (SHARED) (VARS) (COMPUTE)
   (STRUCT (|Alarm3|
         (|Alarm3| (|signal| |threshold| -> |Output|)
```

```
            (|T1Context| |ASContext|) NIL NIL NIL))))
;;; Definition for |Alarm3P|.
(DEF-PROCESSOR |Alarm3P|
  ((((|Signal| :DOUBLE)) ((|Threshold| :DOUBLE)) NIL -> NIL NIL
   ((|out| :ONLINE)))
  NIL NIL NIL ((|Monitoring+Control| . "a3procmon.icon"))
  ((|Monitoring+Control| (O |Signal| |Signal| 260 245 193 253)
     (O |Threshold| |Event| 260 311 168 316)
     (O |out| |Alarm| 605 238 598 218)
     (O |Alarm3| |@Alarm3| 434 222 440 294)
     (C |Dataflow| (|Alarm3| |Output| OUTS 0) (|out| NIL NIL NIL)
       |White| 491 249 611 249)
     (C |Dataflow| (|Signal| NIL NIL NIL) (|Alarm3| |signal| INS 0)
       |White| 276 258 439 258)
     (C |Dataflow| (|Threshold| NIL NIL NIL)
       (|Alarm3| |threshold| INS 1) |White| 279 322 420 322 420
       282 439 282)
     (T |Alarm3P| 398 2)))
  (|Alarm3|
    (|@Alarm3| (|Signal| |Threshold| -> |out|) NIL NIL NIL NIL)))
```

/home/vandy/Designs/Resp2/Processors/alarm4.prc

The "alarm4" database which generates both high and low alarms for a single signal, using a high and a low threshold. Both of these thresholds are adaptable, *i.e.* can be modified during runtime.

```
;;; Procs database: "alarm4.prc"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Processors/")
;;; Definition for |Alarm4P|.
(DEF-PROCESSOR |Alarm4P|
   ((( |Signal| :DOUBLE))
    (( |Threshold1| :DOUBLE) ( |Threshold2| :DOUBLE)) NIL -> NIL NIL
    (( |out1| :ONLINE) ( |out2| :ONLINE)))
   NIL NIL NIL (( |Monitoring+Control| . "a4procmon.icon"))
   (( |Monitoring+Control| (O |Signal| |Signal| 260 245 193 253)
      (O |Threshold1| |Event| 260 311 168 316)
      (O |Threshold2| |Event| 260 350 167 354)
      (O |out1| |Alarm| 604 221 602 202)
      (O |out2| |Alarm| 607 268 599 254)
      (O |Alarm4| |@Alarm4| 434 222 440 294)
      (C |Dataflow| (|Signal| NIL NIL NIL) (|Alarm4| |signal| INS 0)
        |White| 279 260 442 260)
      (C |Dataflow| (|Threshold1| NIL NIL NIL)
        (|Alarm4| |threshold1| INS 1) |White| 271 324 396 324 396
        271 442 271)
      (C |Dataflow| (|Threshold2| NIL NIL NIL)
        (|Alarm4| |threshold2| INS 2) |White| 272 362 422 362 422
        280 442 280)
      (C |Dataflow| (|Alarm4| |Output1| OUTS 0) (|out1| NIL NIL NIL)
        |White| 487 231 609 231)
      (C |Dataflow| (|Alarm4| |Output2| OUTS 1) (|out2| NIL NIL NIL)
        |White| 489 250 512 250 512 280 611 280)
      (T |Alarm4P| 398 2)))
   (|Alarm4|
     (|@Alarm4|
       (|Signal| |Threshold1| |Threshold2| -> |out1| |out2|) NIL
       NIL NIL NIL)))
;;; Definition for |@Alarm4|.
(DEFCOMPOUND |@Alarm4|
   ((|signal| :STREAM) (|threshold1| :STREAM) (|threshold2| :STREAM)
    -> (|Output1| :STREAM) (|Output2| :STREAM))
   NIL NIL NIL NIL (( |Monitoring+Control| . "a4compmon.icon"))
   (( |Monitoring+Control| (O |signal| |Stream| 204 244 113 245)
      (O |threshold1| |Stream| 200 282 101 282)
      (O |threshold2| |Stream| 200 322 107 322)
      (O |Output1| |Stream| 606 239 634 241)
```

```
      (O |Output2| |Stream| 606 269 634 269)
      (O |AS2Context| |StaticParameter| 199 192 120 203)
      (O |AS1Context| |StaticParameter| 199 152 120 163)
      (O |T2Context| |StaticParameter| 199 108 120 111)
      (O |T1Context| |StaticParameter| 199 68 120 51)
      (O |Alarm4| |Alarm4| 417 219 414 290)
      (C |Signal connection| (|Alarm4| |outalarm1| OUTS 0)
        (|Output1| NIL NIL NIL) |White| 466 245 611 245)
      (C |Signal connection| (|Alarm4| |outalarm2| OUTS 1)
        (|Output2| NIL NIL NIL) |White| 470 261 613 261 613 275)
      (C |Signal connection| (|threshold2| NIL NIL NIL)
        (|Alarm4| |threshold2| INS 2) |White| 208 331 406 331 406
        279 422 279)
      (C |Signal connection| (|threshold1| NIL NIL NIL)
        (|Alarm4| |threshold1| INS 1) |White| 213 290 375 290 375
        267 428 267)
      (C |Signal connection| (|signal| NIL NIL NIL)
        (|Alarm4| |signal| INS 0) |White| 211 255 429 255)
      (C |Par connection| (|T1Context| NIL NIL NIL)
        (|Alarm4| |T1Context| PARS 0) |White| 218 78 377 78 377 225
        424 225)
      (C |Par connection| (|T2Context| NIL NIL NIL)
        (|Alarm4| |T2Context| PARS 1) |White| 212 115 372 115 372
        236 430 236)
      (C |Par connection| (|AS1Context| NIL NIL NIL)
        (|Alarm4| |AS1Context| PARS 2) |White| 214 160 461 160 461
        224)
      (C |Par connection| (|AS2Context| NIL NIL NIL)
        (|Alarm4| |AS2Context| PARS 3) |White| 215 203 471 203 471
        224)
      (T |@Alarm4| 403 9)))
  (SIGNALS)
  (PARAMS (|T1Context| 6.0 :DOUBLE) (|T2Context| 8.0 :DOUBLE)
      (|AS1Context| 0 :INT) (|AS2Context| 0 :INT))
  (SHARED) (VARS) (COMPUTE)
  (STRUCT (|Alarm4|
          (|Alarm4|
            (|signal| |threshold1| |threshold2| -> |Output1|
              |Output2|)
            (|T1Context| |T2Context| |AS1Context| |AS2Context|)
            NIL NIL NIL))))
;;; Definition for |Alarm4|.
(DEFPRIMITIVE |Alarm4| :IFANY
  ((|signal| :STREAM) (|threshold1| :STREAM) (|threshold2| :STREAM)
   -> (|outalarm1| :STREAM) (|outalarm2| :STREAM))
  ((|T1Context| 0.0 :DOUBLE) (|T2Context| 0.0 :DOUBLE)
   (|AS1Context| 0 :INT) (|AS2Context| 0 :INT))
  NIL NIL NIL ((|Monitoring+Control| . "a4pmon.icon"))
  ((|Monitoring+Control| (O |signal| |Stream| 163 273 104 274)
     (O |threshold1| |Stream| 163 314 68 315)
     (O |threshold2| |Stream| 163 354 68 355)
     (O |outalarm1| |Stream| 404 243 429 242)
```

```
    (O |outalarm2| |Stream| 405 284 434 280)
    (O |T1Context| |StaticParameter| 159 95 71 95)
    (O |T2Context| |StaticParameter| 159 135 71 138)
    (O |AS1Context| |StaticParameter| 159 175 66 175)
    (O |AS2Context| |StaticParameter| 159 219 66 222)
    (T |Alarm4| 413 5)))
"alarm4es")
```

/home/vandy/Designs/Resp2/Processors/cpn.prc

This database file contains a cpn processing block with two inputs and one output.

```
;;; Procs database: "cpn.prc"
(DEF-ICONLIB "/home/vandy/Icons/Resp2/Processors/")
;;; Definition for |cpnp|.
(DEFPRIMITIVE |cpn| :IFANY
   ((|in1| :SCALAR) (|in2| :SCALAR) -> (|out| :SCALAR)) NIL NIL NIL
   NIL ((|Monitoring+Control| . "cpn.icon"))
   ((|Monitoring+Control| (O |in1| |Scalar| 164 159 158 136)
      (O |in2| |Scalar| 168 239 165 224)
      (O |out| |Scalar| 472 183 471 160) (T |cpn| 368 27)))
   "cpn_script")
;;; Definition for |cpn-pro|.
(DEF-PROCESSOR |cpn-pro|
   (NIL ((|in1| :DOUBLE) (|in2| :DOUBLE)) NIL -> NIL ((|out| :FLOAT))
      NIL)
   NIL NIL NIL ((|Monitoring+Control| . "cpn-pro.icon"))
   ((|Monitoring+Control| (O |in1| |Event| 157 189 155 170)
      (O |in2| |Event| 158 279 156 261)
      (O |out| |Event| 519 234 516 208)
      (O |cpn-inst1| |cpn| 271 175 317 153)
      (C |Dataflow| (|in1| NIL NIL NIL) (|cpn-inst1| |in1| INS 0)
        |White| 176 202 275 243)
      (C |Dataflow| (|in2| NIL NIL NIL) (|cpn-inst1| |in2| INS 1)
        |White| 174 288 276 278)
      (C |Dataflow| (|cpn-inst1| |out| OUTS 0) (|out| NIL NIL NIL)
        |White| 448 254 527 245)
      (T |cpn-pro| 373 39)))
   (|cpn-inst1| (|cpn| (|in1| |in2| -> |out|) NIL NIL NIL NIL)))
```

/home/vandy/Designs/Resp2/Processors/load.lsp

This is a lisp file which is used by the IPCS initialisation to load the processors' precompiled object files, which implement the processing blocks (actors) computations, into the multigraph kernel.

```
;;*************************************************************************
;;
;; load.lsp
;;
;; RNP              16/03/94
;;
;; Loads the following object and libraries in this order:
;; resp multigraph scripts object
;; k&r hugin users, hugin , maths , standard c libraries.
;;
;;*************************************************************************

(si:faslink "resp.o" "-lkrhu -lhugin -lm -lc")
```

/home/vandy/Designs/Resp2/Processors/makefile

This makefile is used to compile the multigraph actor scripts used in the prototype application.

```
##########################################
#
#  makefile for resp signal processing
#  actor scripts
#
#  RNP              14/03/94
#
##########################################

.SUFFIXES: .c .o .lsp

CSRC = resp.c
COBJ = resp.o
BASE = ../../..
CC   = cc
INCLUDES= -I$(BASE)/H -I$(HUGINHOME)/include
CFLAGS= $(INCLUDES)
SHELL= /bin/csh
ECHO = ~/Bin/echo

all:        $(COBJ)


resp.o: resp.c
        $(CC) $(CFLAGS) -c resp.c
        $(ECHO) '\000\000\004\020\000\000\000\000#()\c' | cat >>resp.o
```

/home/vandy/Designs/Resp2/Processors/resp.c

The "resp.c" file contains all the scripts for the processors used in the prototype application. They include the alarm generators and a cpn processing block.

```
/*************************************************************************
*
*  resp.c
*
*  RNP            16/03/95
*
*  contains all signal processing actor scripts and initial mgk binding
*
*************************************************************************/

#include <stdio.h>
#include <sys/time.h>
#include "cmgkdefs.h"
#include "cmgkfunc.h"
#include "hugin.h"
#include "krhu.h"

#define domainpath "/home/vandy/Hugin/Domains/resp1/resp1"


void  alarm1es();
void  alarm2es();
void  alarm3es();
void  alarm4es();
void  cpn_script();


make_binder(init_code)
 bind(alarm1es)
 bind(alarm2es)
 bind(alarm3es)
 bind(alarm4es)
bind(cpn_script)
end_bind(;)

void cpn_script(context)
int context;
{
        /********* VAR ******************/
        domain_reference domain;
        node_reference node;
        char node_name[12];
```

```
        float find,middel,var;
        /*********************************/

domain = Load_Domain(domainpath);
Select_Domain(domain);

        switch(mgk_trigger_port()) {  /* 1 */
      case 0:
                strcpy(node_name,"Vt");
                break;
         case 1:
                strcpy(node_name,"f");
                break;
         default:break;          } /* 1 */
        node = Get_Node_By_Name(node_name);
        find=mgk_d_receive(mgk_trigger_port());
        Compute_Beliefs(find,node);
     strcpy(node_name,"O2-alv-cap");
        node = Get_Node_By_Name(node_name);
        middel = Mean_Value_Of_Node(node);
        var = Varians_Of_Node(node);
Free_Domain();
        mgk_f_propagate(0, middel);
/*      mgk_f_propagate(1, var);  */

}



struct Context {
        double sp;
        int *dp;
};

void alarm1es(context)     /* single channel low alarm generator with dynamic threshold*/
struct Context *context;
{
        double data,dummy;
        int trigger;
        int curtime = time((long *)0);


        trigger = mgk_trigger_port();
        switch(trigger)
         {
          case 0:
           data=mgk_d_receive(0);
           if ((data < context->sp)          /* if signal value less than threshold      */
              &&
          (*(context->dp) == 0))             /* and alarm is off                         */
      {
            *(context->dp) = 1;              /* turn the alarm on                        */
```

```
            mgk_i_propagate(0,curtime);
           }
           else
           {
            if ((*(context->dp) == 1) && (data >= context->sp))
                                /* if alarm is on and value is higher than threshold   */
            {
             *(context->dp) = 0;                    /* turn alarm off                  */
             mgk_i_propagate(0,-curtime);
            }
           }
           break;
          case 1:
        dummy=mgk_d_receive(1);
           break;
          case 2:
           context->sp=mgk_d_receive(2);            /* set the threshold to new value  */
           break;
         }

}

struct Context1 {
        double sp1;
        double sp2;
        int sp3;
};

 static double data2=0.0;

void alarm2es(context)                  /* two channel (a&b) hi alarm generator    */
                                        /* triggered on the slower data channel (a) */
struct Context1 *context;
{
        double data1;
        int trigger;
        int curtime = time((long *)0);


        trigger = mgk_trigger_port();
        switch(trigger)
         {
          case 0:
           data1=mgk_d_receive(0);
           if ((data1 > context->sp1)
               &&
          (data2 > context->sp2)
                   &&
              (context->sp3 == 0))
       {
           context->sp3 = 1;
           mgk_i_propagate(0,curtime);
```

```
            }
            else
            {
             if ((context->sp3 == 1) && ((data1 <= context->sp1) || (data2 <= context->sp2)))
             {
               context->sp3 = 0;
              mgk_i_propagate(0,-curtime);
             }
            }
            break;
           case 1:
        data2=mgk_d_receive(1);
            break;
           case 2:
        context->sp1=mgk_d_receive(2);
            break;
           case 3:
            context->sp2=mgk_d_receive(3);
            break;
          }

}


struct Context2 {
          double sp1;
          int sp2;
};

void alarm3es(context)     /* single channel hi alarm generator with dynamic threshold  */
struct Context2 *context;
{
          double data,dummy;
          int trigger;
          int curtime = time((long *)0);


          trigger = mgk_trigger_port();
          switch(trigger)
           {
           case 0:
            data=mgk_d_receive(0);
            if ((data > context->sp1)
                &&
          (context->sp2 == 0))
        {
            context->sp2 = 1;
            mgk_i_propagate(0,curtime);
           }
           else
           {
            if ((context->sp2 == 1) && (data <= context->sp1))
```

```
            {
             context->sp2 = 0;
             mgk_i_propagate(0,-curtime);
            }
           }
           break;
          case 1:
        context->sp1=mgk_d_receive(1);
           break;
         }

}

struct Context3 {
        double sp1;
        double sp2;
        int sp3;
        int sp4;
};

void alarm4es(context)              /* two independent channel hi and lo alarm generator */
struct Context3 *context;
{
        double data;
        int trigger;
        int curtime = time((long *)0);


        trigger = mgk_trigger_port();
        switch(trigger)
         {
          case 0:
           data=mgk_d_receive(0);
           if ((data < context->sp1)
               &&
          (context->sp3 == 0))
       {
            context->sp3 = 1;
            mgk_i_propagate(0,curtime);
           }
          else
           {
           if ((context->sp3 == 1) && (data >= context->sp1))
           {
            context->sp3 = 0;
            mgk_i_propagate(0,-curtime);
           }
          }
          if ((data > context->sp2)
              &&
         (context->sp4 == 0))
       {
```

```
            context->sp4 = 1;
            mgk_i_propagate(1,curtime);
           }
           else
           {
            if ((context->sp4 == 1) && (data <= context->sp2))
            {
             context->sp4 = 0;
             mgk_i_propagate(1,-curtime);
            }
           }
           break;
          case 1:
        context->sp1=mgk_d_receive(1);
           break;
          case 2:
        context->sp2=mgk_d_receive(2);
           break;

         }

}
```

F.7 System Files

The system files contain the initialisations for the multigraph kernel and the IPCS system.

/home/vandy/System/Resp2/init.lsp

The "init.lsp" contains the kernel initialisations.

```
;;;
;;; Initialization file for KCL
;;;

;;; ed, edl, edsl

(defvar *edit-file* "")

(defun int-ed (fname load-p verb-p)
  (setq *edit-file* fname)
  (system (format nil "vi ~A" fname))
  (if load-p
      (load fname :print verb-p)))

(defmacro ed  (&optional (x *edit-file*))
  `(int-ed ',x nil nil))

(defmacro edl (&optional (x *edit-file*))
  `(int-ed ',x t nil))

(defmacro edsl (&optional (x *edit-file*))
  `(int-ed ',x t t))

;;; pp

(defmacro pp (arg)
  (if (symbolp arg)
      `(progn
            (if (fboundp ',arg)
                (pprint (symbol-function ',arg)))
            (if (boundp ',arg)
```

```
            (pprint (list 'setq ',arg ,arg))))
    `(pprint ',arg)))

;;; unix Cshell
(defmacro shell ()
  `(system "csh"))

;;; undefun
(defmacro undefun (name args)
   `(defun ,name ,args
          (format t "Undefined function ~S called with args ~S.~%"
                  ',name (list ,@args))))


;;; signals
(si::catch-bad-signals)

;;; optional args
(if (> (si:argc) 2) (load (string (si:argv 2))))
```

/home/vandy/System/Resp2/resp.lsp

The "resp.lsp" file contains the initialisations of the IPCS for the application prototype.

```
;;*************************************************************
;;
;;  resp.lsp
;;
;;  RNP             17/03/95
;;
;;  Contains IPCS initialisations
;;
;;*************************************************************

(use-fast-links nil)

;;; translating cpn model files from *.cpn to *.net file and
;;; compiling from *.net to *.kb for xhugin
;;;(let ((this (truename ".")))
;;;  (si:chdir "/users/vandy/Designs/Resp2/Cpn")
;;;  (system  "csh cpn_compile_script")
;;;  (si:chdir this))

(load "trace")
(trace actor-create)
;(trace make-process-model)
;(trace make-processor)

;;;
;;;
;;; Configuration file for the Respogg models
;;;
;;; ************************************************************

;;; Declare main task
(dcl-task Task
        (node           "")
        (user           "")
        (passwd         "")
        (type           :foreign)
)

;;; Declare main environment
(dcl-environment Env Task
        (priority 128)
)
```

```
;;; Define signal,event, and alarm types
(def-ifp-type AlarmIFP
 (dir :in)
 (data :alarm))

(def-ifp-type SignalIFP
  (dir :in)
  (data :double)
  2)

(def-ifp-type FSignalIFP
  (dir :in)
  (data :float)
  2)

(def-ifp-type SSignalIFP
  (dir :in)
  (data :double)
  10)

(def-ifp-type VSSignalIFP
  (dir :in)
  (data :double)
  30)

(def-ifp-type OutSignalIFP
  (dir :out)
  (data :double))

;;; not needed here ;;;; (gdcl-signal |Analog Signal| :double Task Env)

(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip14|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip15|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip16|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip17|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip18|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip20|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip21|   AlarmIFP)

(gdcl-ifp |Resp/O2_delivery/Oxygenation/Gasx/ip1|   AlarmIFP)

(gdcl-ifp |Resp/O2_delivery/Oxygenation/Pul_circ/ip1|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Pul_circ/ip2|   AlarmIFP)

(gdcl-ifp |Resp/O2_delivery/Oxygenation/ip5|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/ip6|   AlarmIFP)

(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip5|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip6|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip8|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip10|   AlarmIFP)
```

```
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip11|   AlarmIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip12|   AlarmIFP)

(gdcl-ifp |Resp/O2_delivery/ip1|   AlarmIFP)

(gdcl-ifp |Resp/O2_consumption/ip1|   AlarmIFP)
(gdcl-ifp |Resp/O2_consumption/ip2|   AlarmIFP)

(gdcl-ifp |Resp/ip2|   ALARMIFP)

(gdcl-alarm |Resp/O2_delivery/Oxygenation/Ventilation/hypo_vent_al| :online TASK ENV)
(gdcl-alarm |Resp/O2_delivery/Oxygenation/Ventilation/hyper_vent_al| :online TASK ENV)
(gdcl-alarm |Resp/O2_delivery/Oxygenation/Ventilation/fio2_hi_al| :online TASK ENV)

(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip1|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip2|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip3|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip4|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip5|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip6|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip7|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip8|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip9|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip10|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip11|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip12|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip13|   SSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Ventilation/ip19|   SSignalIFP)

(gdcl-ifp |Resp/O2_delivery/Oxygenation/Pul_circ/ip3|   FSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Pul_circ/ip4|   FSignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/Pul_circ/ip5|   FSignalIFP)

(gdcl-ifp |Resp/O2_delivery/Oxygenation/ip1|   SignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/ip2|   SignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/ip3|   SignalIFP)
(gdcl-ifp |Resp/O2_delivery/Oxygenation/ip4|   SignalIFP)

(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip1|   SignalIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip2|   SignalIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip3|   SignalIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip4|   SignalIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip7|   SignalIFP)
(gdcl-ifp |Resp/O2_delivery/Sys_circ/ip9|   SignalIFP)

(gdcl-ifp |Resp/ip1|   VSSIGNALIFP)

;;; This is black magic
(si::allocate-contiguous-pages 1000)
(si::allocate-relocatable-pages 310)
(allocate 'cons 300)
(allocate 'string 50)
```

```
;;; Define design directory
(def-design-directory "/users/vandy/Designs/Resp2/Pml")

;;; Define all the project files
(def-project-files RESPPML "resp.prj")

;;; Load the files of the project
(load-project 'RESPPML)

;;; Define design directory
(def-design-directory "/users/vandy/Designs/Resp2/Physical")

;;; Define all the project files
(def-project-files RESPPHY "resp.prj")

;;; Load the files of the project
(load-project 'RESPPHY)

;;; Define design directory
(def-design-directory "/users/vandy/Designs/Resp2/PML-Physical")

;;; Define all the project files
(def-project-files RESPPPL "resp.prj")

;;; Load the files of the project
(load-project 'RESPPPL)

;;; Define design directory
(def-design-directory "/users/vandy/Designs/Resp2/Panel")
(def-icondir "/users/vandy/Icons/Resp2/Panel")

;;; Define all the project files

(def-project-files RESPPNL "resp.prj")

;;; Load the files of the project
(load-project 'RESPPNL)

;;; Define design directory
(def-design-directory "/users/vandy/Designs/Resp2/Processors")

;;; Loading signal processing actor script objects and relavant libraries
(let ((this (truename ".")))
 (si:chdir "/users/vandy/Designs/Resp2/Processors")
 (load "load")
 (si:chdir this))

;;; Define all the project files
(def-project-files RESPPROC "resp.prj")

;;; Load the files of the project
```

```
(load-project 'RESPPROC)

;;; Define the simulation program
(def-ifp-program "Respogg" "" "RespTsim" 2)

;;; Define toplevel process with parameter values, enviroments and tasks
(def-toplevel-process |Resp| () (Env) (Task))

;;; Define the convenience function START
(defun start () (make-toplevel-process-model) (run-nodes)
  (format t "Don't type anything in this window!~%~%~%"))
```

F.8 Data Acquisition Program

This section contains the program listings for the data acquisition program (see section 6.4) written in visual basic for windows.

VENT_DA/DOC.TXT

The first file contains the basic documentation for the data acquisition program.

Data Acquisition Program
Version 1.0

This application is intended for use with Drager Evita Intensive Care Ventilators only. The program in no way interferes with the normal operation of the ventilator. The primary purpose is to record data for offline analysis, therefore little or no processing is carried out and all received data is saved. Trend graphs are plotted for some variables and status messages are displayed in a log.

Before starting the program ensure the RS232 cable is connected with the connector marked "PC" at the PC side (serial port 1, COM1) and the unmarked connector at the Ventilator side. Slide the earth switch at the rear of ventilator upwards and make sure the PC is NOT connected to earth.(See page 12 of Evita manual.) The Evita RS232 connection socket is electrically isolated from the equipment electronics. This together with above earthing arrangement ensure electrical safety from the data acquisition hardware (ie PC).

For each data request the ventilator generates about 0.1Kbyte of data. There is a simple data reduction that will result in not recording identical data telegrams. If two or more identical data telegrams are received only the time stamp is recorded. Despite this, you need to ensure that there is enough free disk space to record the data. For example a sample rate of 10 samples per minute and a recording session of 10 hours requires about 0.6Mbyte free disk space on the desired drive.

Start the program by double_clicking on the icon. To start the data collection select "Start" from the "File" menu. This will prompt for a file name for the recorded data, the sampling interval and the number of points in the trend graphs.

If you close some graph windows they can be reopened by selecting the "Show all graphs" option in the "View" menu. The graphs may be resized by resizeing the graph window first and then left mouse click on the graph.

To end the data collection select "Exit" from file menu.

RNP 22/07/1994

Update
Version 1.0a

This is a more intelligent version. It can be started when the ventilator is off or cable not connected. It will detect when the ventilator has been switched on or cable connection has been re_established. Similarly if for any reason the ventilator is switched off during a data collection session or cable is disconnected, the program will restart as soon as the ventilator is switched on again or cable reconnected. The result will be appended to same file as before.

The point of this is that the program can be started when convenient and the data collection starts automatically when the ventilator is switched on. Also if data collection is interrupted it will restart itself once the ventilator is on and cable connection is ok.

RNP 28/10/1994

VENT_DA/VENT_DA.VBZ

The template file used by the visual basic set-up wizard to create master distribution diskettes for the data acquisition application.

```
[Files]
File1=C:\WINDOWS\SYSTEM\GRID.VBX
File2=C:\WINDOWS\SYSTEM\MSOLE2.VBX
File3=C:\WINDOWS\SYSTEM\STORAGE.DLL
File4=C:\WINDOWS\SYSTEM\COMPOBJ.DLL
File5=C:\WINDOWS\SYSTEM\OLE2PROX.DLL
File6=C:\WINDOWS\SYSTEM\OLE2NLS.DLL
File7=C:\WINDOWS\SYSTEM\OLE2DISP.DLL
File8=C:\WINDOWS\SYSTEM\OLE2.DLL
File9=C:\DOS\SHARE.EXE
File10=C:\WINDOWS\SYSTEM\OLE2.REG
File11=C:\WINDOWS\SYSTEM\MSOLEVBX.DLL
File12=C:\WINDOWS\SYSTEM\ANIBUTON.VBX
File13=C:\WINDOWS\SYSTEM\CMDIALOG.VBX
File14=C:\WINDOWS\SYSTEM\COMMDLG.DLL
File15=C:\WINDOWS\SYSTEM\CRYSTAL.VBX
File16=C:\WINDOWS\SYSTEM\CRPE.DLL
File17=C:\WINDOWS\SYSTEM\CRXLATE.DLL
File18=C:\WINDOWS\SYSTEM\GAUGE.VBX
File19=C:\WINDOWS\SYSTEM\GRAPH.VBX
File20=C:\WINDOWS\SYSTEM\GSWDLL.DLL
File21=C:\WINDOWS\SYSTEM\GSW.EXE
File22=C:\WINDOWS\SYSTEM\KEYSTAT.VBX
File23=C:\WINDOWS\SYSTEM\MSCOMM.VBX
File24=C:\WINDOWS\SYSTEM\MSMASKED.VBX
File25=C:\WINDOWS\SYSTEM\MSOUTLIN.VBX
File26=C:\WINDOWS\SYSTEM\PICCLIP.VBX
File27=C:\WINDOWS\SYSTEM\SPIN.VBX
File28=C:\WINDOWS\SYSTEM\THREED.VBX
File29=C:\VENT_DA\VENT_DA.EXE
File30=C:\VENT_DA\DOC.TXT

[WinFiles]
File1=SHARE.EXE
File2=OLE2.REG
File3=GSW.EXE

[Flags]
AppEXEName=C:\VENT_DA\VENT_DA.EXE
APPTITLE=VENT_DA
DataControl=0
OLE=-1
```

```
Btrieve=0
ODBC=0
DataAccess=0
VB.Mak Full Path=C:\VENT_DA\VENT_DA.MAK
VB.Mak Path Only=C:\VENT_DA\
Disk Drive=a:
Disk Type=1
```

VENT_DA/VENT_DA.MAK

The project file for the data acquisition program.

```
VENT_DA.FRM
C:\WINDOWS\SYSTEM\GRID.VBX
C:\WINDOWS\SYSTEM\MSOLE2.VBX
C:\WINDOWS\SYSTEM\ANIBUTON.VBX
C:\WINDOWS\SYSTEM\CMDIALOG.VBX
C:\WINDOWS\SYSTEM\CRYSTAL.VBX
C:\WINDOWS\SYSTEM\GAUGE.VBX
C:\WINDOWS\SYSTEM\GRAPH.VBX
C:\WINDOWS\SYSTEM\KEYSTAT.VBX
C:\WINDOWS\SYSTEM\MSCOMM.VBX
C:\WINDOWS\SYSTEM\MSMASKED.VBX
C:\WINDOWS\SYSTEM\MSOUTLIN.VBX
C:\WINDOWS\SYSTEM\PICCLIP.VBX
C:\WINDOWS\SYSTEM\SPIN.VBX
C:\WINDOWS\SYSTEM\THREED.VBX
VENT-DA.BAS
VD_F2.FRM
VD_F1.FRM
VD_F4.FRM
VD_F3.FRM
VD_F5.FRM
VD_F6.FRM
ProjWinSize=71,557,248,215
ProjWinShow=2
IconForm="vent_da"
Title="vent_da"
ExeName="VENT_DA.EXE"
```

VENT_DA/VENT_DA.BAS

The file containing the global declarations for the data acquisition program.

```
'----------------------------------------
'WINDOW MESSAGE FACILITY
'----------------------------------------
Global Const WM_USER = &H400
Global Const EM_SETREADONLY = (WM_USER + 31)
Declare Function SendMessage Lib "User" (ByVal hWnd As Integer, ByVal wMsg As Integer, ByVal
wParam As Integer, lParam As Any) As Long

'----------------------------------------
'Comm Control
'----------------------------------------
'Handshaking
Global Const MSCOMM_HANDSHAKE_NONE = 0
Global Const MSCOMM_HANDSHAKE_XONXOFF = 1
Global Const MSCOMM_HANDSHAKE_RTS = 2
Global Const MSCOMM_HANDSHAKE_RTSXONXOFF = 3

'Event constants
Global Const MSCOMM_EV_SEND = 1
Global Const MSCOMM_EV_RECEIVE = 2
Global Const MSCOMM_EV_CTS = 3
Global Const MSCOMM_EV_DSR = 4
Global Const MSCOMM_EV_CD = 5
Global Const MSCOMM_EV_RING = 6
Global Const MSCOMM_EV_EOF = 7

'Error code constants
Global Const MSCOMM_ER_BREAK = 1001
Global Const MSCOMM_ER_CTSTO = 1002
Global Const MSCOMM_ER_DSRTO = 1003
Global Const MSCOMM_ER_FRAME = 1004
Global Const MSCOMM_ER_OVERRUN = 1006
Global Const MSCOMM_ER_CDTO = 1007
Global Const MSCOMM_ER_RXOVER = 1008
Global Const MSCOMM_ER_RXPARITY = 1009
Global Const MSCOMM_ER_TXFULL = 1010

'------------------------------------------------------------
'message in error log
Global ef_msg As String
```

VENT_DA/VENT_DA.FRM

This form (vent_da) implements the main window that is opened when data acquisition application is first started by clicking on its icon.

```
VERSION 2.00
Begin Form vent_da

'-------------------------------------------------------
' The attributes of the main application window
'-------------------------------------------------------
  BackColor      = &H00FFFF80&
  Caption        = "Draeger  Data"
  ClientHeight   = 6195
  ClientLeft     = 150
  ClientTop      = 750
  ClientWidth    = 4470
  ControlBox     = 0  'False
  Height         = 6885
  Icon           = VENT_DA.FRX:0000
  Left           = 90
  LinkTopic      = "Form1"
  MaxButton      = 0  'False
  ScaleHeight    = 6195
  ScaleWidth     = 4470
  Top            = 120
  Width          = 4590

'-------------------------------------------------------
' The timer object for port reinitialisation
'-------------------------------------------------------
  Begin Timer Timer2
    Enabled      = 0  'False
    Interval     = 30000
    Left         = 480
    Top          = 360
  End

'-------------------------------------------------------
' The timer object for RS232 timed enquiry
'-------------------------------------------------------
  Begin Timer Timer1
    Enabled      = 0  'False
    Interval     = 6000
    Left         = 0
    Top          = 360
  End
```

```
'-----------------------------------------------------------
' The Grid object that displays received data
'-----------------------------------------------------------
  Begin Grid Grid1
    Cols          =  3
    FixedRows     =  0
    Height        =  4095
    HighLight     =  0  'False
    Left          =  960
    Rows          =  16
    ScrollBars    =  1  'Horizontal
    TabIndex      =  4
    Top           =  2040
    Width         =  3490
  End

'-----------------------------------------------------------
' Text boxes to display data telegram and length
'-----------------------------------------------------------
  Begin TextBox Text2
    Height        =  300
    Left          =  960
    TabIndex      =  1
    Top           =  1716
    Width         =  1450
  End
  Begin TextBox Text1
    Height        =  1668
    Left          =  960
    MultiLine     =  -1  'True
    TabIndex      =  0
    Top           =  24
    Width         =  3490
  End

'-----------------------------------------------------------
' Commms object
'-----------------------------------------------------------
  Begin MSComm Comm1
    Interval      =  1000
    Left          =  0
    NullDiscard   =  -1  'True
    RThreshold    =  1
    SThreshold    =  10
    Top           =  720
  End

'-----------------------------------------------------------
' Label objects to identify telegram and length
'-----------------------------------------------------------
  Begin Label Label3
```

```
    BackStyle      =  0 'Transparent
    Caption        =  "Telegram"
    Height         =  252
    Left           =  120
    TabIndex       =  5
    Top            =  120
    Width          =  852
  End
  Begin Label Label2
    BackStyle      =  0 'Transparent
    Caption        =  "Length"
    Height         =  240
    Left           =  120
    TabIndex       =  3
    Top            =  1680
    Width          =  636
  End

'------------------------------------------------------
' Label object for Data
'------------------------------------------------------
  Begin Label Label1
    BackStyle      =  0 'Transparent
    Caption        =  "Data"
    Height         =  288
    Left           =  120
    TabIndex       =  2
    Top            =  2040
    Width          =  492
  End

'------------------------------------------------------
' Main form's Menu items
'------------------------------------------------------
  Begin Menu mnufile
    Caption        =  "&File"
    Begin Menu mnustart
      Caption      =  "&Start"
    End
    Begin Menu mnuexit
      Caption      =  "E&xit"
    End
  End
  Begin Menu mnuview
    Caption        =  "&View"
    Begin Menu mnugraph
      Caption      =  "Show all &Graphs"
    End
    Begin Menu mnustatuslog
      Caption      =  "Show &Status Log"
    End
    Begin Menu mnusherr
```

```
    Caption       =  "Show  &Error Log"
  End
 End
 Begin Menu mnuhelp
  Caption       =  "&Help"
  Begin Menu mnudoc
   Caption      =  "&Documentation"
  End
  Begin Menu mnuabout
   Caption      =  "&About Data Acquisition ..."
  End
 End
End

Const port = 1              'communication port
Const num_sets = 3          'number of pressure data sets in form1 graph1
Const num_sets1 = 1         'number of data sets in resistance compliance grah1 form6
Const dt_len = 93           'data telegram length
Const sig_id_len = 2        'signal identification length
Const max_ts_len = 8        'maximum length of telegram section in data telegram
Const vars = 15             'number of variables (rows in grid1 of form1)
Const mis_val = -1          'missing value

Const ack_len = 534         'identification telegram length    new
'Const nak_len = 400        'status telegram length           new
Const enq_len = 93          'data telegram length             new

Dim arr2(num_sets) As Integer
Dim arr4(num_sets1) As Integer
Dim start As Integer
Dim initdone As Integer             ' new
Dim Filename As String
Dim num_points As Integer           ' number of points in graphs
Dim arr() As Double
Dim arr1() As Double
Dim arr3() As Double

'
'-------------------------------------------------------------------------------
Sub Comm1_OnComm ()

  Static ee_msg  As String
  Dim er_msg   As String
  Dim time_stamp As String

  er_msg = ""

  Select Case comm1.CommEvent
   Case MSCOMM_ER_BREAK
   Case MSCOMM_ER_CDTO
   Case MSCOMM_ER_CTSTO
   Case MSCOMM_ER_DSRTO
   Case MSCOMM_ER_FRAME          'Framing Error.
```

```
        er_msg = "Framing Error."
    Case MSCOMM_ER_OVERRUN        'Data Lost.
        er_msg = "Data Lost."
    Case MSCOMM_ER_RXOVER         'Receive Buffer Over Flow.
        er_msg = "Receive Buffer Overflow."
    Case MSCOMM_ER_RXPARITY       'Parity Error.
        er_msg = "Parity Error."
    Case MSCOMM_ER_TXFULL         'Transmit Buffer Full.
        er_msg = "Transmit Buffer Full."

    Case MSCOMM_EV_CD             'Change in Carrier detect
    Case MSCOMM_EV_CTS            'Change in clear to send
    Case MSCOMM_EV_DSR            'Change in data set ready
   ' er_msg = "data set ready"
    Case MSCOMM_EV_RING           'Ring detect
    Case MSCOMM_EV_RECEIVE        'Received RThreshold number of characters
    Case MSCOMM_EV_SEND           'There are SThreshold number of characters in the
transmit buffer.

  End Select

        If Len(er_msg) > 0 Then                    ' if there is an error message
        time_stamp = Format(Now, "ttttt") + "  "   ' get a time stamp
        ee_msg = ee_msg + time_stamp + er_msg + Chr(13) + Chr(10)
        vd_f5.Text1.Text = ef_msg + ee_msg         ' stick itin the error log new
        End If

End Sub
'
-------------------------------------------------------------------------------
Sub Form_Load ()

Static varn(vars) As String

  varn(0) = "Evita ID"
  varn(1) = "t       (hh,mm)"      ' Time
  varn(2) = "VTe          (l)"     ' Exp. Tidal Vol
  varn(3) = "f         (/min)"     ' Breathing frequency
  varn(4) = "MV       (l/min)"     ' Minute Vol
  varn(5) = "Peak     (mbar)"      ' Peak Pressure
  varn(6) = "Plat     (mbar)"      ' Plateau Pressure
  varn(7) = "PEEP     (mbar)"      ' PEEP
  varn(8) = "Pmin     (mbar)"      ' Minimum Pressure
  varn(9) = "Mean     (mbar)"      ' Mean Pressure
  varn(10) = "FIO2         (%)"    ' Insp. O2-concentration
  varn(11) = "C     (ml/mbar)"     ' Compliance
  varn(12) = "R    (mbar*s/l)"     ' Resistance
  varn(13) = "MVs     (l/min)"     ' Spont. Minute Vol"
  varn(14) = "f s      (/min)"     ' Spont. Frequency
  varn(15) = "Temp   (deg C)"      ' Airway Temperature

  arr2(0) = 5                      ' Peak Pressure
  arr2(1) = 6                      ' Plateau Pressure
```

```
    arr2(2) = 9             ' Mean Pressure
    arr2(3) = 7             ' PEEP

    arr4(0) = 11            ' Compliance
    arr4(1) = 12            ' Resistance

    start = 0
    initdone = 0                    'new
    lastline = ""                   'static defined in timer1
    ss_msg = ""                     'static defined in timer1
    ee_msg = ""                     'static defined in comm1

    'Filename = "c:\zan.txt"

    'Make text boxes readonly.
    dummy = SendMessage(Text1.hWnd, EM_SETREADONLY, 1, 0)
    dummy = SendMessage(Text2.hWnd, EM_SETREADONLY, 1, 0)


' Set grid column widths
    grid1.ColWidth(0) = 1450
    For i% = 1 To grid1.Cols - 1
    grid1.ColWidth(i%) = 1010
    Next i%

' Fill grid with variable names
    grid1.Col = 0
    For i% = 0 To vars
    grid1.Row = i%
    grid1.Text = varn(i%)
    Next i%

End Sub
'
-------------------------------------------------------------------------------
Sub initial ()

    Dim latest_char As String
    Dim sta_tel As String           ' Length of Status telegram
    Dim txtime As Single            ' Transmition time
    Dim duration As Single
    Dim dline As String             ' Data line
    Dim pos1, pos2
    Dim etime_stamp As String       ' Error time stamp
    ' Dim loopcntr As Integer

    latest_char = ""        ' new
    dline = ""
    etime_stamp = ""        ' new

    etime_stamp = Format(Now, "ttttt") + "  "                                   ' new
    ef_msg = ef_msg + etime_stamp + "initializing" + Chr(13) + Chr(10)          ' new
    vd_f5.Text1.Text = ef_msg                                                   ' new
```

```
    'clear port here
    comm1.CommPort = port
    comm1.Settings = "1200,E,7,2"
    comm1.PortOpen = True
    comm1.Output = Chr(17)              ' DC1
    txtime! = Timer + .01               ' wait
    Do
    Loop Until Timer > txtime!
    comm1.PortOpen = False

    'clear port here
    comm1.PortOpen = True
    comm1.Output = Chr(20)              ' DC4
    txtime! = Timer + .01               ' wait
    Do
    Loop Until Timer > txtime!
    comm1.PortOpen = False


'clear port
comm1.InputLen = 1
comm1.PortOpen = True
comm1.Output = Chr(6)                   ' ACK
sta_tel = ""
latest_char = ""                        ' new
duration! = Timer + 6
'loopcntr = 0                           ' new
Do
     latest_char = comm1.Input
      If Len(latest_char) <> 0 Then
       sta_tel = sta_tel + latest_char
      End If
    ' loopcntr = loopcntr + 1
Loop Until (Timer > duration! Or latest_char = Chr(4))
comm1.PortOpen = False
'this should go
'Print loopcntr
'Save to file
If latest_char = Chr(4) And Len(sta_tel) = ack_len Then  ' if message ok then continue
    Open Filename For Append As #1
    pos1 = 1            ' chop the identification telegram into lenghts of about 80 characters
    While Len(Mid$(sta_tel, pos1)) > 80
      pos2 = InStr(pos1 + 79, sta_tel, Chr(27))
      If pos2 = 0 Then
            pos2 = pos1 + 80
      End If
      dline = Mid$(sta_tel, pos1, pos2 - pos1) + Chr(13) + Chr(10)
      pos1 = pos2
      Print #1, dline;
    Wend
    dline = Mid$(sta_tel, pos1) + Chr(13) + Chr(10)
```

```
            Print #1, dline;
            Close #1
            Text1.Text = sta_tel
            Text2.Text = Str$(Len(sta_tel))

        'clear port
        comm1.InputLen = 1
        comm1.PortOpen = True
        comm1.Output = Chr(21)        ' NAK
        sta_tel = ""
        latest_char = ""              ' new
        'loopcntr = 0                 ' new
        duration! = Timer + 10
        Do
            latest_char = comm1.Input
            If Len(latest_char) <> 0 Then
               sta_tel = sta_tel + latest_char
            End If
        ' loopcntr = loopcntr + 1              ' new
        Loop Until (Timer > duration! Or latest_char = Chr(4))
        comm1.PortOpen = False
        ' this should go
        'Print loopcntr
        'Print "out of NAK loop"
        'Save to file
    If latest_char = Chr(4) Then              ' if message ok continue
            Open Filename For Append As #1
            pos1 = 1                          ' chop the status telegram
            While Len(Mid$(sta_tel, pos1)) > 80
               pos2 = InStr(pos1 + 79, sta_tel, Chr(29))
               If pos2 = 0 Then
                     pos2 = pos1 + 80
               End If
               dline = Mid$(sta_tel, pos1, pos2 - pos1) + Chr(13) + Chr(10)
               pos1 = pos2
               Print #1, dline;
            Wend
            dline = Mid$(sta_tel, pos1) + Chr(13) + Chr(10)
            Print #1, dline;
            Close #1
            Text1.Text = sta_tel        ' stick the status telegram in the text box
            Text2.Text = Str$(Len(sta_tel))

      initdone = 1
      timer2.Enabled = False
      timer1.Enabled = True
      etime_stamp = Format(Now, "ttttt") + "  "
      ef_msg = ef_msg + etime_stamp + "Done initializing" + Chr(13) + Chr(10)
      vd_f5.Text1.Text = ef_msg
      End If
    End If
```

```
End Sub
'
--------------------------------------------------------------------------------
Sub mnuabout_Click ()

    vd_f3.Show

End Sub
'
--------------------------------------------------------------------------------
Sub mnudoc_Click ()

    dummy = Shell("notepad.exe c:\vent_da\doc.txt", 1)

End Sub
'
--------------------------------------------------------------------------------
Sub mnuexit_Click ()

    Close
    End

End Sub
'
--------------------------------------------------------------------------------
Sub mnugraph_Click ()

    vd_f1.Show
    vd_f2.Show
    vd_f6.Show

End Sub
'
--------------------------------------------------------------------------------
Sub mnusherr_Click ()

    vd_f5.Show

End Sub
'
--------------------------------------------------------------------------------
Sub mnustart_Click ()

   Dim sample_t As Single

   If start <> 1 Then
    Filename = InputBox$("Enter  output filename:", "File Open")
     If Len(Filename) > 0 Then                ' if valid filename then initilaize
          Do                                  ' get data rate from user input
          sample_t = Val(InputBox$("Enter sample interval in seconds (>=3)", "Data rate"))
          Loop Until (sample_t >= 3 And sample_t < 120)
          timer1.Interval = 1000 * sample_t
          Do                                  ' get number of points to plot
          num_points = Val(InputBox$("Enter number of points for trend graphs (>=10)", "Points in
Graphs"))
          Loop Until (num_points > 10 And num_points < 100)
          initial                             ' new
          start = 1
```

```
        If initdone = 0 Then timer2.Enabled = True          ' new


        ReDim arr(num_points) As Double                     ' data of graph1 of form2
        ReDim arr1(num_sets, num_points) As Double          ' data of graph1 of form1
        ReDim arr3(num_sets1, num_points) As Double

        vd_f1.Show
        vd_f2.Show
        vd_f6.Show


    End If
  End If

End Sub
'
-------------------------------------------------------------------------------
Sub mnustatuslog_Click ()

   vd_f4.Show

End Sub
'
-------------------------------------------------------------------------------
Sub Timer1_Timer ()
  ReDim dataval(vars) As Double
  Dim duration As Single
  Dim data_tel As String
  Dim tel_sec As String
  Dim dline As String
  Dim st_msg As String
  Dim time_stamp As String
  Dim etime_stamp As String
  Dim latest_char As String

  Static lastline As String
  Static ss_msg As String
  Static errcount As Integer                ' new

  If start = 1 And initdone = 1 Then        ' new

        st_msg = ""
        data_tel = ""
        tel_sec = ""
        time_stamp = ""
        etime_stamp = ""                    ' new
        dline = ""
        latest_char = ""

        'if possible clear port here
        comm1.InputLen = 1
        comm1.PortOpen = True
        comm1.Output = Chr(5)               ' ENQ  Request data
```

```
        var_num = 0
        duration! = Timer + 2
        Do                                  ' Receive and parse data
        latest_char = comm1.Input
        If Len(latest_char) <> 0 Then
          Select Case latest_char
          Case ","                          ' Convert the comma character to decimal point
           tel_sec = tel_sec + "."
           data_tel = data_tel + "."
          Case Chr(27)                      ' ESC data separator
                If Len(tel_sec) > sig_id_len And Len(tel_sec) < max_ts_len And var_num < vars +
1 Then
                   grid1.Row = var_num
                   grid1.Col = 1
                   dataval(var_num) = Val(Mid$(tel_sec, 3))
                   grid1.Text = Mid$(tel_sec, 3)
                   var_num = var_num + 1
                End If
                tel_sec = ""
                data_tel = data_tel + ","
          Case Chr(4)               ' EOT
                If Len(tel_sec) > sig_id_len And Len(tel_sec) < max_ts_len And var_num < vars +
1 Then
                   grid1.Row = var_num
                   grid1.Col = 1
                   dataval(var_num) = Val(Mid$(tel_sec, 3))
                   grid1.Text = Mid$(tel_sec, 3)
                   var_num = var_num + 1
                End If
                tel_sec = ""
                data_tel = data_tel + " "
          Case Chr(29)              ' status separator
                If Len(tel_sec) > sig_id_len And Len(tel_sec) < max_ts_len And var_num < vars +
1 Then
                   grid1.Row = var_num
                   grid1.Col = 1
                   dataval(var_num) = Val(Mid$(tel_sec, 3))
                   grid1.Text = Mid$(tel_sec, 3)
                   var_num = var_num + 1
                End If
                tel_sec = ""
                data_tel = data_tel + latest_char
          Case Else
                tel_sec = tel_sec + latest_char
                data_tel = data_tel + latest_char
          End Select
        End If
        Loop Until (Timer > duration! Or latest_char = Chr(4))
        comm1.PortOpen = False
    If latest_char = Chr(4) And Len(data_tel) = enq_len Then          ' new
        errcount = 0                                                 ' new
        dline = data_tel + Chr(13) + Chr(10)
```

```
        comp_res = StrComp(Mid$(dline, 14), Mid$(lastline, 14))   'compare ignoring time stamp
        lastline = dline
        ' If same data just keep time stamp
        If comp_res <> 1 And comp_res <> -1 Then
        dline = Mid$(data_tel, 1, 14) + Chr(13) + Chr(10)
        End If
        ' Save to file
        Open Filename For Append As #1
        Print #1, dline;
        Close #1

        Text1.Text = data_tel
        Text2.Text = Str$(Len(data_tel))
        'this should go
        'data_tel = "hello world" + Chr(29) + "status message 1" + Chr(29) + "status message 2" +
Chr(29) + "end of message"
        ' Chopping the status message
        pos1 = InStr(1, data_tel, Chr(29))
        If pos1 <> 0 Then
           pos2 = InStr(pos1 + 1, data_tel, Chr(29))
           grid1.Row = 1
           grid1.Col = 1
           time_stamp = grid1.Text + "  "
        ' this should go
        '   time_stamp = Format(Now, "ttttt") + "  "

           While pos2 <> 0
           st_msg = st_msg + time_stamp + Mid$(data_tel, pos1 + 1, pos2 - pos1 - 1) + Chr(13) +
Chr(10)
           pos1 = pos2
           pos2 = InStr(pos1 + 1, data_tel, Chr(29))
           Wend
           st_msg = st_msg + time_stamp + Mid$(data_tel, pos1 + 1) + Chr(13) + Chr(10) +
"********************" + Chr(13) + Chr(10)
           ss_msg = ss_msg + st_msg
           vd_f4.Text1.Text = ss_msg                    ' Stick it in the status log
        End If

  ' Graphing

        vd_f2.Graph1.AutoInc = 1
        vd_f2.Graph1.NumPoints = num_points + 1

        ' shift register to get new data and shift out old
        For i% = 0 To num_points - 1
        arr(i%) = arr(i% + 1)
        Next i%

        If Len(data_tel) < dt_len Then
           arr(num_points) = mis_val                    ' Missing value
        Else
           arr(num_points) = dataval(4)                 ' Tidal Volume
```

```
        End If
        For i% = 0 To num_points
        vd_f2.Graph1.GraphData = arr(i%)
        Next i%

        For i% = 0 To num_points
           vd_f2.Graph1.XPosData = vd_f2.Graph1.ThisPoint - num_points - 1
        Next i%

        vd_f2.Graph1.GraphType = 6
        vd_f2.Graph1.DrawMode = 2

' Graphing  Resistance and Compliance

        vd_f6.Graph1.AutoInc = 1
        vd_f6.Graph1.NumPoints = num_points + 1
        vd_f6.Graph1.NumSets = num_sets1 + 1

' shift register to get new data and shift out old

        For j% = 0 To num_sets1
         For i% = 0 To num_points - 1
           arr3(j%, i%) = arr3(j%, i% + 1)
         Next i%
        Next j%

  If Len(data_tel) < dt_len Then
  For j% = 0 To num_sets1
  arr1(j%, num_points) = mis_val                    ' Missing value
  Next j%
  Else

'this should go
'   dataval(11) = 11
'   dataval(12) = 3

        For j% = 0 To num_sets1
        arr3(j%, num_points) = dataval(arr4(j%))    ' pressures
        Next j%
  End If

        For j% = 0 To num_sets1
          For i% = 0 To num_points
            vd_f6.Graph1.GraphData = arr3(j%, i%)
          Next i%
        Next j%

        For j% = 0 To num_sets1
           For i% = 0 To num_points
                 vd_f6.Graph1.XPosData = vd_f6.Graph1.ThisPoint - num_points - 1
           Next i%
        Next j%
```

```
        vd_f6.Graph1.GraphType = 6
        vd_f6.Graph1.DrawMode = 2

' Graphing  pressures

        vd_f1.Graph1.AutoInc = 1
        vd_f1.Graph1.NumPoints = num_points + 1
        vd_f1.Graph1.NumSets = num_sets + 1

' shift register to get new data and shift out old

        For j% = 0 To num_sets
         For i% = 0 To num_points - 1
          arr1(j%, i%) = arr1(j%, i% + 1)
         Next i%
        Next j%

  If Len(data_tel) < dt_len Then
  For j% = 0 To num_sets
  arr1(j%, num_points) = mis_val                    ' Missing value
  Next j%
  Else
'this should go
'   dataval(5) = 5
'   dataval(6) = 4
'   dataval(9) = 3
'   dataval(7) = 1

        For j% = 0 To num_sets
        arr1(j%, num_points) = dataval(arr2(j%))    ' pressures
        Next j%
        End If

        For j% = 0 To num_sets
         For i% = 0 To num_points
           vd_f1.Graph1.GraphData = arr1(j%, i%)
         Next i%
        Next j%

        For j% = 0 To num_sets
         For i% = 0 To num_points
           vd_f1.Graph1.XPosData = vd_f1.Graph1.ThisPoint - num_points - 1
         Next i%
        Next j%

        vd_f1.Graph1.GraphType = 6
        vd_f1.Graph1.DrawMode = 2
 Else
        errcount = errcount + 1
        If errcount >= 4 Then               ' If garbage or nothing so many consecutive times
        initdone = 0                        ' then must reinitilaise
```

```
        timer1.Enabled = False
        timer2.Enabled = True
        etime_stamp = Format(Now, "ttttt") + "  "
        ef_msg = ef_msg + etime_stamp + "Ventilator not responding or cable disconnected!" +
Chr(13) + Chr(10) + "Trying to reinitialize" + Chr(13) + Chr(10)        ' Let sombody know
        vd_f5.Text1.Text = ef_msg
    End If
   End If
  End If

End Sub
'
-------------------------------------------------------------------------------
Sub Timer2_Timer ()

   If initdone = 0 Then initial                    ' Do the reinitialisation every timer tick

End Sub
```

VENT_DA/VD_F1.FRM

This form (vd_f1) contains the window for plotting the graphs of the airway pressure variables.

```
VERSION 2.00
Begin Form vd_f1
  Caption        =  "Airway pressures (mbar)"
  ClientHeight   =  2996
  ClientLeft     =  4170
  ClientTop      =  2282
  ClientWidth    =  5295
  Height         =  3374
  Icon           =  VD_F1.FRX:0000
  Left           =  4110
  LinkTopic      =  "Form1"
  ScaleHeight    =  2996
  ScaleWidth     =  5295
  Top            =  1960
  Width          =  5415
  Begin GRAPH Graph1
    AsciiColor     =  "12~1~2"
    AsciiFSize     =  "80~150~100~80"
    AsciiLegend    =  "Peak~Plateau~Mean~PEEP"
    Background     =  7 'Light Gray
    BottomTitle    =  "Samples"
    GraphType      =  6 'Line
    Height         =  3015
    Left           =  0
    LegendStyle    =  1 'Color
    NumPoints      =  21
    NumSets        =  4
    RandomData     =  0 'Off
    TabIndex       =  0
    ThickLines     =  0 'Off
    Top            =  0
    Width          =  5295
  End
End
'
-------------------------------------------------------------------------------
Sub Graph1_Click ()

    'fit graph to window
    graph1.Height = Height - 100
    graph1.Width = Width - 100

End Sub
```

VENT_DA/VD_F2.FRM

This form (vd_f2) contains the window for plotting the volume graphs.

```
VERSION 2.00
Begin Form vd_f2
  Caption        =  "Minute Volume (l/min)"
  ClientHeight   =  3150
  ClientLeft     =  4065
  ClientTop      =  630
  ClientWidth    =  5325
  Height         =  3528
  Icon           =  VD_F2.FRX:0000
  Left           =  4005
  LinkTopic      =  "Form2"
  ScaleHeight    =  3150
  ScaleWidth     =  5325
  Top            =  308
  Width          =  5445
  Begin GRAPH Graph1
    AsciiColor     =  "1~2~2~2~1"
    AsciiFSize     =  "80~150~100~80"
    AsciiLegend    =  "MV (l/min)"
    AsciiSymbol    =  "1~1~9~1~1"
    Background     =  7  'Light Gray
    BottomTitle    =  "Samples"
    GraphType      =  6  'Line
    Height         =  3165
    Left           =  0
    LegendStyle    =  1  'Color
    NumPoints      =  21
    RandomData     =  0  'Off
    TabIndex       =  0
    ThickLines     =  0  'Off
    Top            =  0
    Width          =  5340
  End
End

Sub Graph1_Click ()

   'fit graph to window
   graph1.Height = Height - 100
   graph1.Width = Width - 100

End Sub
```

VENT_DA/VD_F3.FRM

This form (vd_f3) specifies the window that is opened by the "HELP" menu item.

```
VERSION 2.00
Begin Form vd_f3
  Caption        =  "Ventilator Data Acquisition Help"
  ClientHeight   =  2025
  ClientLeft     =  2145
  ClientTop      =  1560
  ClientWidth    =  5730
  Height         =  2430
  Icon           =  VD_F3.FRX:0000
  Left           =  2085
  LinkTopic      =  "Form1"
  MaxButton      =  0  'False
  ScaleHeight    =  2025
  ScaleWidth     =  5730
  Top            =  1215
  Width          =  5850
  Begin PictureBox Picture1
    BorderStyle    =  0  'None
    Height         =  476
    Left           =  240
    Picture        =  VD_F3.FRX:0302
    ScaleHeight    =  480
    ScaleWidth     =  510
    TabIndex       =  2
    Top            =  378
    Width          =  510
  End
  Begin CommandButton Command1
    Caption        =  "OK"
    Height         =  378
    Left           =  4680
    TabIndex       =  1
    Top            =  252
    Width          =  975
  End
  Begin Label Label1
    AutoSize       =  -1  'True
    Caption        =  "Label1"
    Height         =  2030
    Left           =  960
    TabIndex       =  0
    Top            =  0
    Width          =  3975
    WordWrap       =  -1  'True
```

```
  End
End

Sub Command1_Click ()

    vd_f3.Hide

End Sub

Sub Form_Load ()

    Dim abt_msg

    abt_msg = Chr(13) + Chr(10) + Chr(10) + "Ventilator Data Acquisition Program" + Chr(13) +
Chr(10) + "Version 1.0a"
    abt_msg = abt_msg + Chr(13) + Chr(10) + Chr(10) + "Copyright " + Chr(169) + "  1994
R.N.Pirjamali"
    abt_msg = abt_msg + Chr(13) + Chr(10) + Chr(10) + "Developed for  City University  &&
RBNHLH"
    vd_f3.Label1.Caption = abt_msg

End Sub
```

VENT_DA/VD_F4.FRM

This form (vd_f4) implements the window for displaying the status log.

```
VERSION 2.00
Begin Form vd_f4
  Caption       =  "Status Log"
  ClientHeight  =  3090
  ClientLeft    =  4935
  ClientTop     =  4965
  ClientWidth   =  4335
  Height        =  3495
  Icon          =  VD_F4.FRX:0000
  Left          =  4875
  LinkTopic     =  "vd_f4"
  MaxButton     =  0  'False
  ScaleHeight   =  3090
  ScaleWidth    =  4335
  Top           =  4620
  Width         =  4455
  Begin TextBox Text1
    Height      =  4212
    Left        =  0
    MultiLine   =  -1  'True
    ScrollBars  =  2  'Vertical
    TabIndex    =  0
    Top         =  0
    Width       =  4932
  End
End

Sub Form_Load ()

   dummy = SendMessage(Text1.hWnd, EM_SETREADONLY, 1, 0)

End Sub

Sub Text1_Change ()

   vd_f4.Show

End Sub
```

VENT_DA/VD_F5.FRM

This form (vd_f5) implements the window that contains the communication error log.

```
VERSION 2.00
Begin Form vd_f5
  Caption       =  "Communication  Errors "
  ClientHeight  =  2295
  ClientLeft    =  2955
  ClientTop     =  2280
  ClientWidth   =  4320
  Height        =  2700
  Icon          =  VD_F5.FRX:0000
  Left          =  2895
  LinkTopic     =  "Form1"
  MaxButton     =  0  'False
  ScaleHeight   =  2295
  ScaleWidth    =  4320
  Top           =  1935
  Width         =  4440
  Begin TextBox Text1
    Height      =  2295
    Left        =  0
    MultiLine   =  -1  'True
    ScrollBars  =  2  'Vertical
    TabIndex    =  0
    Text        =  "No Comms Errors!"
    Top         =  0
    Width       =  4335
  End
End

Sub Form_Load ()

   dummy = SendMessage(Text1.hWnd, EM_SETREADONLY, 1, 0)

End Sub

Sub Text1_Change ()

   vd_f5.Show

End Sub
```

VENT_DA/VD_F6.FRM

This form (vd_f6) implements the window that contains the graphs of the resistance and compliance variables.

```
VERSION 2.00
Begin Form vd_f6
  Caption       =  "Resistance & Compliance"
  ClientHeight  =  3766
  ClientLeft    =  4695
  ClientTop     =  4004
  ClientWidth   =  6075
  Height        =  4144
  Icon          =  VD_F6.FRX:0000
  Left          =  4635
  LinkTopic     =  "Form1"
  ScaleHeight   =  3766
  ScaleWidth    =  6075
  Top           =  3682
  Width         =  6195
  Begin GRAPH Graph1
    AsciiColor    =  "1~4~1~1"
    AsciiFSize    =  "80~150~100~80"
    AsciiLegend   =  "C (ml/mbar)~R (mbar*s/l)~~R (l/min*kPa)"
    Background    =  7  'Light Gray
    BottomTitle   =  "Samples"
    GraphType     =  6  'Line
    Height        =  3794
    Left          =  0
    LegendStyle   =  1  'Color
    NumPoints     =  21
    NumSets       =  2
    RandomData    =  0  'Off
    TabIndex      =  0
    ThickLines    =  0  'Off
    Top           =  0
    Width         =  6135
  End
End

Sub Graph1_Click ()

    ' Fit graph to window size
    graph1.Height = Height - 100
    graph1.Width = Width - 100

End Sub
```