

INTELLIGENT AGENTS AND
HIERARCHICAL CONSTRAINT DRIVEN DIAGNOSTIC UNITS
FOR A TELEOPERATED FLUID HANDLING LABORATORY

by

Hessam Seyed Sarjoughian

A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

1989

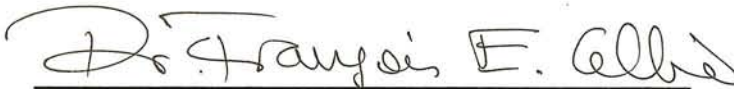
STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under the rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: _____
**APPROVAL BY THESIS DIRECTOR**

This thesis has been approved on the date shown below:



François. E. Cellier
Associate Professor of
Electrical and Computer Engineering

8/18/89
Date

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Professor François E. Cellier, for his invaluable guidance, support, and specially his forbearance and goodwill in shaping this thesis. I gratefully acknowledge all the efforts of Professor Bernard P. Zeigler. This thesis could not have been written without his most valuable guidance and suggestions throughout my research period.

I also wish to express my appreciation to Professor Larry C. Schooley for his careful review of this thesis, helpful comments, and suggestions. The help of Professor Milan Bier and Dr. Ned Egen of the Center for Separation of Science is sincerely appreciated. Acknowledged should also be the help of Eric R. Christensen in providing me with useful suggestions and editing tools.

My special thanks go to my dearest friend Pentti J. Vesanterä for his friendship and helpful discussions during my graduate studies.

I dearly wish to express my sincere feelings of thankfulness to my parents for their constant love, financial support, and encouragement throughout my education.

Finally, the support of this thesis by NASA-Ames Cooperative Agreement No. NCC 2-525 is gratefully acknowledged.

TABLE OF CONTENTS

| | Page |
|---|------|
| LIST OF ILLUSTRATIONS | 6 |
| LIST OF TABLES | 8 |
| ABSTRACT..... | 9 |
| 1. INTRODUCTION | 10 |
| 2. BACKGROUND | 14 |
| 2.0 Outline of the chapter..... | 14 |
| 2.1 System Entity Structure..... | 15 |
| 2.2 General Structure of a Laboratory..... | 16 |
| 2.3 Electrophoresis Experiment..... | 27 |
| 2.4 Event-Based Control..... | 37 |
| 2.5 DEVS Formalism and Discrete-Event Representation of Dynamical Systems..... | 42 |
| 3. MODELING OF INTELLIGENT CONTROLLERS | 48 |
| 3.0 Outline of the Chapter | 48 |
| 3.1 Modeling Methodology..... | 49 |
| 3.2 Modeling the Components of the FHL..... | 56 |
| 3.2.1 Modeling the Camera..... | 57 |
| 3.2.2 Modeling the Rack..... | 59 |
| 3.2.3 Modeling the ITP Device | 62 |
| 3.2.4 Modeling the Robot..... | 69 |
| 3.3 An Overall View of the FHL..... | 79 |
| 4. DIAGNOSTIC UNITS..... | 82 |
| 4.0 Outline of the Chapter | 82 |
| 4.1 The Diagnostic Process | 83 |
| 4.1.1 Conventional Scheme..... | 84 |
| 4.1.2 Artificial Intelligence Scheme..... | 87 |
| 4.2 Hierarchical Diagnostic Agents | 91 |

| | | |
|-------|--|-----|
| 4.3 | Classification Expert System Maker..... | 101 |
| 4.3.1 | Knowledge Representation..... | 107 |
| 4.3.2 | Control Strategies..... | 108 |
| 4.4 | A Diagnostic Unit and M-ITP-DIAG..... | 111 |
| 4.5 | Constraint Driven Diagnostic Units..... | 123 |
| 5. | CONCLUSIONS..... | 141 |
| | APPENDIX—A: Discrete-Event Model of a Robot..... | 144 |
| | REFERENCES..... | 152 |

LIST OF ILLUSTRATIONS

| | page |
|---|------|
| 2.1 Decomposition of the physical structure of a laboratory environment..... | 18 |
| 2.2 Various types of materials necessary in the GPL environment..... | 20 |
| 2.3 Various laboratory settings inside GPL..... | 21 |
| 2.4 Different decompositions of equipment in GPL..... | 23 |
| (a) specialization..... | 23 |
| (b) decomposition..... | 23 |
| 2.5 Various types of Operators available in GPL..... | 25 |
| 2.6 A simplified diagram of the ITP device and a syringe..... | 29 |
| 2.7 Response of a hypothetical ITP experiment..... | 31 |
| 2.8 Decomposition of the FHL into Materials, Workspace, Equipment, and Operator entities..... | 33 |
| 2.9 The pressurized bladder bottle and two types of threshold sensors..... | 34 |
| 2.10 Representation of event-based control..... | 41 |
| (a) block diagram..... | 41 |
| (b) graphical representation of a process..... | 41 |
| 3.1 Model-base, operational, control, and diagnostic models of the ITP device..... | 54 |
| (a) two levels of abstraction..... | 54 |
| (b) three levels of abstraction..... | 54 |
| (c) four levels of abstraction..... | 54 |
| 3.2 Atomic model of the ITP..... | 67 |
| 3.3 External, internal, and output activities for the ITP instrument and robot..... | 68 |

| | | |
|------|--|-----|
| 3.4 | Hierarchy of higher and lower level controllers..... | 70 |
| 3.5 | Decomposition of release and contact tasks into more specialized tasks with respect to filling operation..... | 74 |
| 3.6 | Categorization of the robot's state variables into static and dynamic..... | 76 |
| 3.7 | Block diagram representation of the FHL..... | 80 |
| 4.1 | Block diagram for MBDS..... | 86 |
| 4.2 | Hierarchy of diagnostic units..... | 97 |
| | (a) single diagnoser..... | 97 |
| | (b) high-level and low-level diagnosers..... | 97 |
| 4.3 | Architecture of CESM —components and their interactions..... | 103 |
| 4.4 | Diagnostic structure for the electrophoresis experiment..... | 115 |
| 4.5 | Part of the diagnostic structure given in Figure 4.4..... | 116 |
| 4.6 | A customized diagnostic unit..... | 118 |
| 4.7 | Architecture of the FHL depicted in Figure 3.7 with the addition of a diagnostic unit..... | 120 |
| 4.8 | Graphical representation of Time/Cost..... | 127 |
| 4.9 | A hypothetical diagnostic structure with various measurement devices..... | 130 |
| 4.10 | Steps of a consultation session..... | 136 |
| 4.11 | Partial graphical representation for Tables 4.1-3..... | 139 |

LIST OF TABLES

| | | |
|-----|---|-----|
| 3.1 | Selected state variables chosen to represent the rack..... | 61 |
| 3.2 | Selected state variables chosen to represent the Event-Based controller..... | 75 |
| 4.1 | Numerical results of consultation sessions for cases (a), (b), and (c)..... | 137 |
| 4.2 | Numerical results of consultation sessions for cases (d) and (e)..... | 137 |
| 4.3 | Numerical results of consultation sessions for case (f)..... | 138 |

ABSTRACT

The purpose of this thesis is to study and develop intelligent agents for the forthcoming Space Station Freedom. Relevant intelligent capabilities, which are necessary in a semi-autonomous laboratory environment, are assumed to be built into a robot. An *intelligent controller* based on the DEVS formalism and the event-based approach is considered for an experiment. We shall discuss *multiple model representations*, where each model is tailored toward a specific purpose. Considering the necessity of diagnostic capabilities, we shall discuss the possibility of *hierarchical diagnostic units* for the Space Station. A high-level diagnostic unit is implemented on the basis of an artificial intelligence scheme and a hierarchy of diagnosers. This thesis also discusses the need for real-time diagnostic units and real-time data acquisition. We shall consider a *constraint driven diagnostic unit* which utilizes the *time/cost* (i.e., the actual associated cost or time in inquiring information necessary for a diagnostic process) criterion in an attempt to locate the cause(s) of failures.

CHAPTER 1

INTRODUCTION

The forthcoming Space Station Freedom (SSF) will serve as a platform to conduct long-term scientific experiments with an average duration of 30 days or multiples thereof. This is due to the fact that the SSF will be visited by the Space Shuttle once every 30 days. On that occasion, experiment components can be exchanged/ serviced by the Shuttle crew. Since it is not feasible to request the scientific investigator to spend 30 days in a row at one of the NASA Control Centers, it is important to grant the experimenter access to his instruments from his own laboratory facilities. This mode of operation has been coined *Telescience*. Interaction of the SSF crew should be minimized since crew time is a scarce and expensive resource. The immense cost of human labor in Space (more than \$30,000 per hour) (Hall and Wolbers, 1984) has made the use of *Robot Technology* in Space attractive.

Unfortunately, today's robotic capabilities are insufficient for this task. Robots are currently being used successfully in two different modes of operation. The *autonomous robot* is met in factory automation. These robots are usually sturdy, accurate, and fast, but they are not at all flexible. They can be employed for repetitive precision tasks where the high cost of laboratory set-up for robot manipulation is compensated for by the large production throughput that is achievable in this way. These robots will not be used in Space for some time since robotic tasks in Space are highly

individual and non-repetitive. The *teleoperated robot* is currently met in hazardous environments. A human operator controls the movement of the robot using a mini-master console. In this mode, the robot simply copies the movements of the mini-master. Such robots are very flexible, but currently, they are not sturdy, they are somewhat inaccurate, and they are rather slow. Also this mode of operation is not feasible for the application at hand due to the long communication delay times (the SSF project currently anticipates round-trip delay times of two seconds or longer) which excludes any form of man-in-the-loop control.

What is needed is a *telecommanded robot*, i.e., a robot that can be provided with commands at the task level, but that is able to decompose tasks into primitive operations, and execute these primitive operations autonomously. This mode of operation is sometimes called *semi-autonomous*, since low-level operations are executed autonomously, whereas high-level operations are executed under human command. It is the aim of this research to analyze the necessary technology that will enable such a semi-autonomous robot control environment.

We have selected the *electrophoresis* experiment (refer to Section 2.3) as the medium to analyze the required intelligent agents for high-level control and failure analysis. This apparatus has been of much interest to scientists for many years. This is evident from their involvement as well as the number of instances (12 times) that it has been conducted in Space thus far. The knowledge that can be acquired by investigating this experiment provides the basis for other related/similar experiments. Consequently, it

provides an appropriate groundwork in developing some of the required concepts for the analysis and design of intelligent agents.

The motivation and outline of this thesis is presented in this chapter. In Chapter 2, the necessary background material is treated. The necessity of *multiple model representations* with various levels of abstraction is discussed in Chapter 3. The usefulness of the *discrete-event modeling paradigm* (Zeigler, 1984) in the design of an intelligent robot is demonstrated throughout this chapter. An intelligent controller, which is assumed to be an element of a hierarchy of controllers, is considered on the basis of the *event-based control* scheme (Meystel and Luh, 1987) and the *DEVS formalism* (Zeigler, 1984). Other pertinent entities in a laboratory environment are also considered in Chapter 3.

Chapter 4 begins with a discussion of the conventional and the artificial intelligence schemes for developing diagnostic agents. It is proposed that a hierarchical control scheme accompanied by *hierarchical diagnostic units* provides the most appropriate control structure for the SSF. An expert system shell called *CESM* (Zeigler, 1987-b) is considered for the development of a high-level diagnostic agent. A customized diagnostic agent, which obtains the necessary responses from the modeled measurement devices, is constructed on the basis of CESM. This chapter also argues for the necessity of inquiring information from various sensors in real-time. It is suggested that since *real-time data acquisition* can affect the duration as well as the quality of failure analyses, a diagnostic unit that can take advantage of constraints such as *time/cost* may be useful. A *constraint driven diagnostic agent*, an expanded implementation of CESM

which can reduce the consultation time, demonstrates how appropriate hypotheses may be concluded without inquiring all the available knowledge.

Chapter 5 summarizes our work and suggests improvements for some of the deficiencies that emerged in the course of our studies.

CHAPTER 2

BACKGROUND

2.0 Outline of the chapter

This chapter sets the foundation for the chapters to come. We begin in section 2.1 by presenting a mathematical tool, the System Entity Structure, which will facilitate our upcoming modeling efforts. Next, in section 2.2, we present an overview of the structure of a potential laboratory module which is considered to be part of the forthcoming Space Station Freedom (SSF) and its decomposition into subsystems. This laboratory environment hosts among other equipment an electrophoretic apparatus, a process which separates a solution into its charged components. The chemistry and details of this experiment are treated in section 2.3. A control algorithm which is primarily applicable at higher levels of the control system hierarchy is discussed in section 2.4 along with its comparison to a more conventional control algorithm. The type of control implemented in this algorithm is commonly referred to as *Intelligent Control*; however, we prefer to call it *Event-Based Control* anticipating the use of the Discrete-Event paradigm. This chapter ends with section 2.5 which treats the DEVS (Discrete-Event System Specification) formalism, and in particular the Discrete-Event representation of dynamical systems.

2.1 System Entity Structure

In order to mathematically interact with a system, it must be represented in a form which makes it convenient to interact with. The System Entity Structure (SES) is one such form of a system representation. The SES can be considered as a static model of the system. The System Entity Structure formalism is a tool which facilitates organization of the models for a model base. DEVS-Scheme is a realization of the System Entity Structure formalism developed by Zeigler (Zeigler, 1984) in PC-Scheme (Texas Instruments, 1985), a LISP-based, object-oriented programming environment. The SES, a representation scheme of the structural knowledge of a system, is a labeled tree with attached variable types that satisfies the following axioms (Zeigler, 1984):

- (1) *Alternating entity/aspect or entity/specialization* : Each node has a mode that is either entity/aspect or entity/specialization such that a node and its successors are always opposite modes, and the mode of the root is an entity.
- (2) *Uniformity* : Any two nodes with the same names have identical attached variable types and isomorphic subtrees.
- (3) *Strict hierarchy* : No label appears more than once down any path of the tree.
- (4) *Valid brothers* : No two brothers have the same label.
- (5) *Attached variables* : No two variable types attached to the same item have the same name.

It can be noted that there are three different types of nodes associated with the labeled tree or SES. These nodes are *entity*, *aspect*, and *specialization* which represent three different types of knowledge about the structure of the system. The node *entity*, corresponding to a model component that represents a real world object, can have several aspects and/or specializations. The aspect node symbolized by a single vertical line in the labeled tree represents one decomposition of an entity among many existing ones (cf. Figure 2.1). The specialization node represents a way in which a general entity can be categorized into special entities and is symbolized by a double vertical line in the labeled tree (cf. Figure 2.4). An entity can also be a multiple entity which consists of a collection of homogeneous components as evident in Figure 2.5. Consequently, the aspect of such an entity is called multiple aspect, and it is symbolized by a triple vertical line in the labeled tree.

2.2 General Structure of a Laboratory

The forthcoming Space Station Freedom will host a life science laboratory module which can accommodate various types of laboratory experiments related to space medicine, gravitational biology, and biochemistry (Kelly, 1989). While some of the laboratory components may be highly specialized, there will be also provision for a general purpose workbench, probably in the form of a glove box. It has also been proposed that the Life Sciences module contain a rack mounted multi-purpose

instrument to be operated by a Cartesian robot that can move left and right as well as up and down along the rack (Schooley and Cellier, 1988). We will denote this multi-purpose instrument rack, which can support a large set of different experiments in related disciplines, as GPL (General Purpose Laboratory).

It is apparent that the GPL should exhibit certain characteristics in order to minimize the Hardware/Software/Information efforts required in Requirements Development, Concept Development, Full Scale Engineering Development, System Development, System Test and Integration, Operations, Support and Modification, and Retirement and Replacement (Wymore, 1988). Such characteristics include the physical structure of the laboratory and the necessary means for executing required operations. Therefore, the characteristics that we are concerned with are very general and do not pertain to any particular laboratory environment and/or experiment. To present some insight as to what such characteristics are and how they can be realized, we will examine a laboratory environment, GPL, without referring to any particular experiment. The SES described in 2.1 enables us to describe the GPL by providing a medium to represent the physical structure of a laboratory environment in well defined terms. Each laboratory, in general, can be decomposed into the aspects Materials, Equipment, Workspace, and Operators as shown in Figure 2.1. The SES, mentioned in 2.1, decomposes a Laboratory into:

General Laboratory Structure

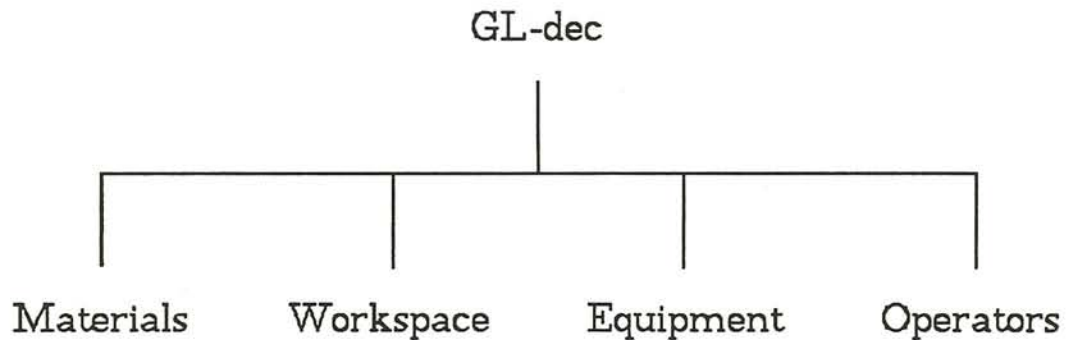


Figure 2.1: Decomposition of the physical structure of a laboratory environment.

Aspects denote decompositions of an entity. A decomposition means that *all* components are parts of the decomposed entity. In our example: the GPL *consists of* Materials, a Workspace, Equipment, and Operators.

Obviously, an ideal laboratory environment, GPL, should support any desired experiment, that is, it should encompass the characteristics of laboratories for e.g. molecular biology, chemical analysis, and material handling to name just a few. The limitations inherent in any space-bound GPL: microgravity, communication obstacles (delay), waste disposal, and contamination control (Schooley and Cellier, 1988) play important roles in the successful completion of any large-scale space experiment. The GPL, therefore, must offer some essential features to suppress some of the limitations mentioned above. Some of the GPL's features are optimization of workspace, incorporation of some of the human capabilities into a robot

or ultimately a group of robots, selection of equipment types, and selection of necessary types of materials for experiments. The description of entities of subsequent decompositions of the GPL will present more and more details, and provide the necessary background required for understanding some aspects of the GPL's operations for a particular experiment.

The entity Materials can have several *specializations* (kind, phase, health hazard) such as shown in Figure 2.2 (Kelly, 1989). The *Kind* of a material determines whether it can be considered as supply, waste, expandable, or whether it is experiment related. The *Phase* of a material is a means of specifying whether it is a liquid, a gas, or a solid body. The *Health Hazard*, on the other hand, characterizes the material as being either pathogenic, radioactive, toxic, caustic, or carcinogenic. The *nil* variant has been added here since a material may not be a health hazard at all.

Characterization of entities by specialization means selection. For each specialization, we need to choose one, i.e., material is either liquid or solid, but it is at the same time of the kind supply or waste, etc. The elements of specializations are always entities, i.e., liquid is an entity, whereas the names of the specializations are attached variables.

Decomposition into components denotes an *all* relationship to the parent, i.e., the laboratory consists of materials *and* a workspace *and* equipment *and* operators; but it denotes a *one-out-of* relationship to the child, i.e., the workspace *is one-out-of* the laboratory, etc.

Specialization denotes a *one-out-of* relationship to the parent, i.e., the (phase of a) material is either liquid *or* solid *or* gaseous; but it denotes an *all* relationship to the child, i.e., *all* liquid_material is material.

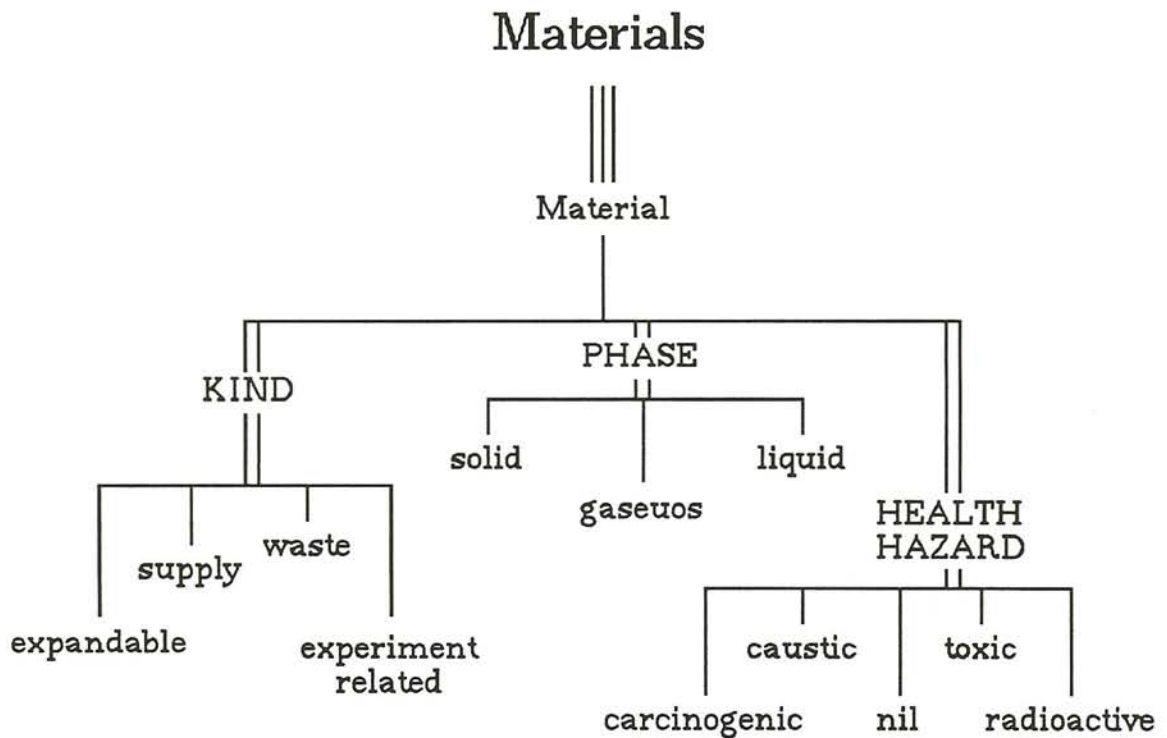


Figure 2.2: Various types of materials necessary in the GPL environment.

This methodology lays the ground work for the concept of *multiple inheritance*. Liquid_Radioactive_Supply_Materials are materials that

inherit all properties of the specialization liquid (phase), the specialization supply (kind), and the specialization radioactive (health hazard).

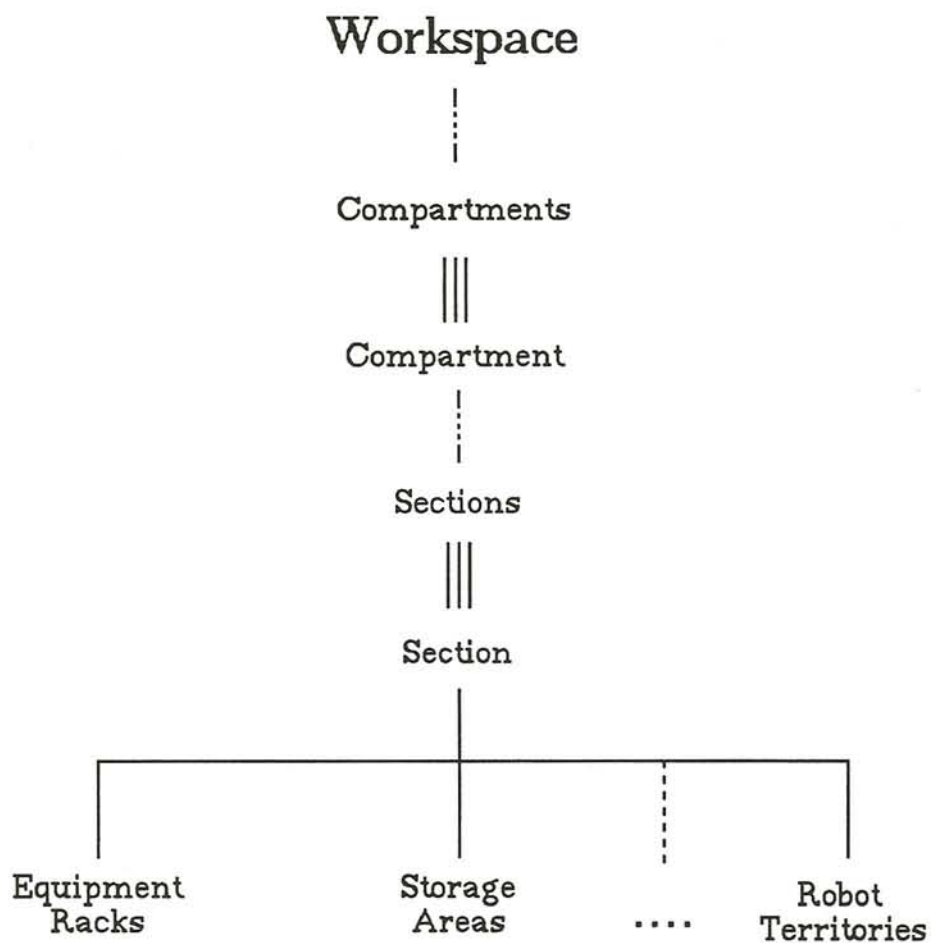


Figure 2.3: Various laboratory setting inside GPL.

The entity Workspace of the GPL is divided into several compartments and subsequently many sections. Each section must be

designed depending on its purpose and other relevant considerations such as permissible size of its containable equipment and its location within a specific compartment. The compartments should have the same physical configurations (Kelly, 1989) whereas the sections must be designed and built in order to accommodate particular specifications enforced by a specific laboratory environment such as those listed earlier. A few possibilities of the sections are presented in Figure 2.3.

Another entity of the GPL is the Equipment that is decomposed/specialized into several types as shown in Figures 2.4a and 2.4b. As these two figures illustrate, the type of decomposition (into aspects or specializations) is not necessarily unique. If we consider *equipment* as a set of different types of apparatus, we can then specialize the generic apparatus into its different types as being a tool *or* a container *or* a transporter, etc. This is shown in Figure 2.4a. Alternatively, we can consider *equipment* as a box full of different utensils, and decompose this box into the various components which are all tools *and* all containers *and* all transporters, etc. Each of the tools can then be decomposed by multiple decomposition into individual tools, and the same holds for the containers, the transporters, and so on. This is illustrated in Figure 2.4b. Each of these entities is then further decomposed into more specific types of equipment. The *Tool*, for example could be a syringe, a pipette, a tube, a gas bottle, a tray, a vacuum sealed plastic bag, or a pressurized bladder bottle. The *Instrument* could be a computer, an input/output device, a transducer, or some other special purpose device.

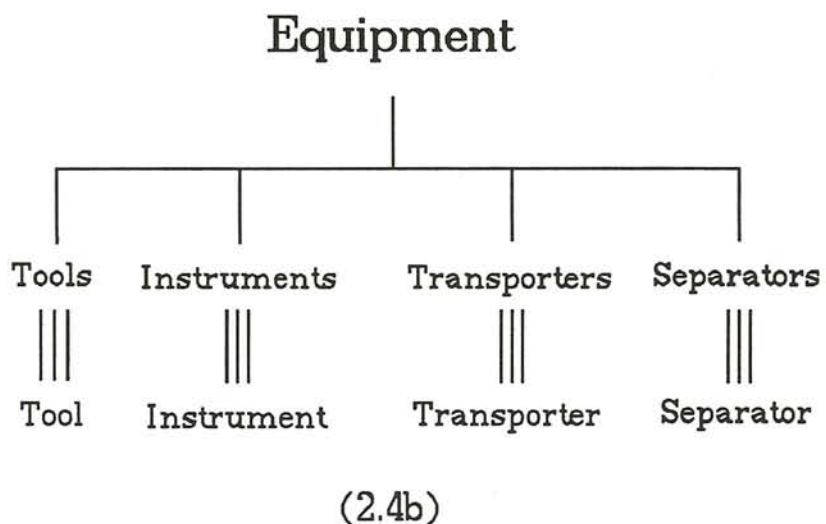
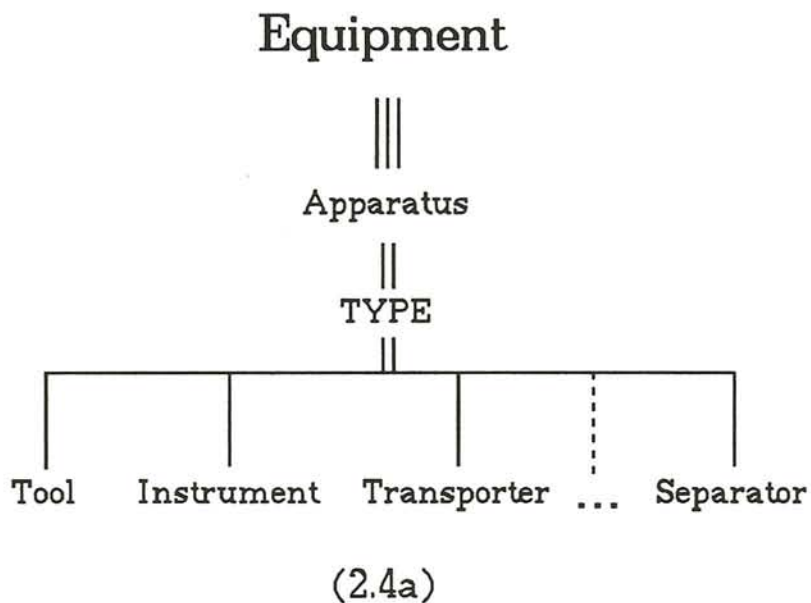


Figure 2.4: Different decompositions of equipment in GPL. (a) specialization; (b) decomposition.

The input devices are of interest to us, and they can be of two types: sensors and video cameras. The sensors can provide information with

respect to temperature, flow, force, pressure, acceleration, velocity, position, voltage, and/or current. The video cameras, on the other hand, provide films or pictures of the entities of Materials, Workspace, Equipment, and the Operators. There are other types of equipment which should be available at the GPL such as *Combiners*, *Separators*, etc.

The last entity Operator which is of primary concern to us is specialized as shown in Figure 2.5. Therefore, the operator can be *either* a Human, *or* an Automated Controller, *or* a Robot. The specialized entity Human, is the most powerful one due to his abilities related to decision making, planning, and learning which are not completely afforded by either of the Automated Controller or the Robot. The presence of humans in space is undesirable due to safety considerations and high cost. Therefore, it is necessary to reduce the number of humans aboard SSF to a minimum by allowing robots and automated controllers to compensate for their absence.

Naturally, it will be necessary to incorporate some of the unavailable human characteristics into automated controllers and/or robots if they are to replace humans. We, however, assume that human capabilities of interest are incorporated only into the robots, rather than building complex or automated controllers that exhibit highly specialized features. This assumption is valid for several reasons. Due to the limited workspace available within the SSF, the number of hard-wired automated apparatus should be minimized due to their limited domain of expertise and brief periods of development. It is impractical and costly to launch equipment/devices for a limited period of time or to service a few

experiments at the GPL. If feasible, it would be better to achieve the same result with a flexible robot which may be already in space, and which can be programmed to perform the desired task.

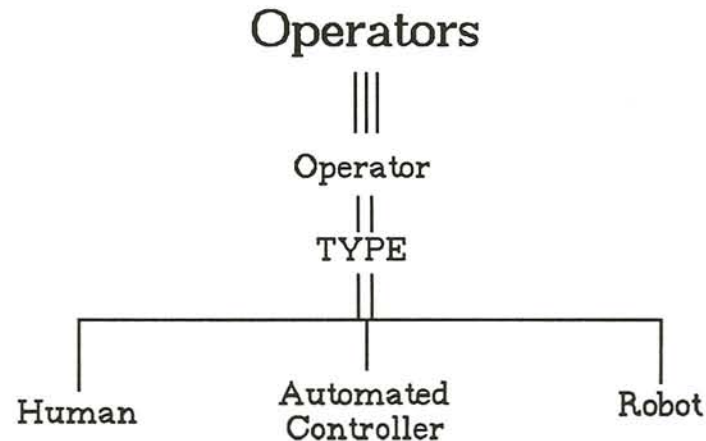


Figure 2.5: Various types of Operators available in GPL.

Moreover, it is impractical to consider the presence of automated equipment in abundance since they would pile up in the limited space available, and consequently, they would leave no space for other entities (see Figure 2.1) which are vital for the operation of the GPL or even the SSF. Thus, it may make sense to perform even tasks that would be easily amenable to hard automation through robot control.

The life cycle of the SSF is considered to be at least 20 years. Robots play a crucial role in this process as they exhibit a crucial property: they are reprogrammable. A robot may be able to perform new/unforeseen tasks

provided that the necessary primitive operations from which these new/unforeseen experiments are composed are provided by it. This is the main disadvantage with automated controllers: they are unable to adapt themselves to new assignments, and therefore, they are handicapped.

What are the characteristics that our laboratory robot must possess to successfully manage a space laboratory? It must be *responsible*, it must possess *decision power*, and it must be able to *recognize its own limitations*. A distributed hierarchical control architecture that possesses these properties can be called *hierarchical control with distributed intelligent agents*. The current status of robot technology is still far from this ultimate goal. It is the aim of this thesis to advance the understanding of what is needed to give a robot the facilities described above.

In the forthcoming chapters, we shall concentrate our efforts on some of the features which must be built into a robot in order to make it responsible for the entire process of preparation of an experiment hosted by the GPL under varying, and maybe even unforeseen, operating conditions. We shall also analyze diagnostic capabilities for fault recovery purposes which must be available to restore the normal operation in the event that something goes wrong.

2.3 Electrophoresis Experiment

We shall consider a particular configuration of the GPL environment for the execution of an electrophoretic experiment, and assume that it is executed under the control and supervision of a single laboratory robot. Before describing this special laboratory environment with respect to the GPL, a brief explanation of the electrophoresis experiment and the steps involved in its preparation is necessary.

Electrophoresis is a process that can separate solutions into their charged components, some of which may be proteins, organic acids, peptides, and metal ions (Hack, 1988).

Electrophoresis experiments can be performed using either Mobility or Isoelectric Point techniques. In particular, the ITP (Isotachopheresis) and the ZE (Zone EPH) methods are based on the Mobility technique, while the IEF (Isoelectric Focusing) employs the Isoelectric Point technique (Thormann, 1984). The device which is being used in our experiment is of the ITP type. It will be referred to as the ITP device, or simply as ITP.

The Isotachopheresis technique applies an electric DC current over the length of a narrow channel, called capillary, that contains the sample to be separated. Figure 2.6 illustrates a simplified schematic of the ITP device which consists of two chambers and a capillary. One of the chambers is filled with the leading electrolyte, and the other contains the terminating electrolyte. The capillary holds the sample solution that is bounded by the two types of electrolytes previously mentioned. These electrolytes are often called reference solutions since their properties must

be known in order to identify various components of a particular sample solution.

Our goal is to study the set-up procedure which consists of filling (or emptying) the chambers, and the capillary. It also includes the examination of each chamber and the capillary for detection of air bubbles after each filling operation. These activities may be referred to as *primitive operations* since other related experiments may require them as well. For the set-up procedure, there exists a predefined sequence of operations which can be described as follows:

1. Filling the Leading Electrolyte chamber,
2. Filling the Terminating Electrolyte chamber, and
3. Injecting the sample solution into the capillary

assuming that the chambers and the capillary are empty and clean prior to the experiment. A successful completion of each of the above sequences requires one or more of the following: Examination of air bubbles in the chambers and/or the capillary, and emptying any of the two chambers or the capillary if air bubbles were indeed detected in them. In setting-up the electrophoresis experiment, it is seldom possible to complete the set-up procedure without repeating one or more of the three operations mentioned above due to the introduction of air bubbles in the chambers or the capillary, often all the other required activities (such as the movement of the robot's gripper from one coordinate to another) are successfully completed. The

examination for the presence of air bubbles is extremely important since their presence invalidates the experiment's results.

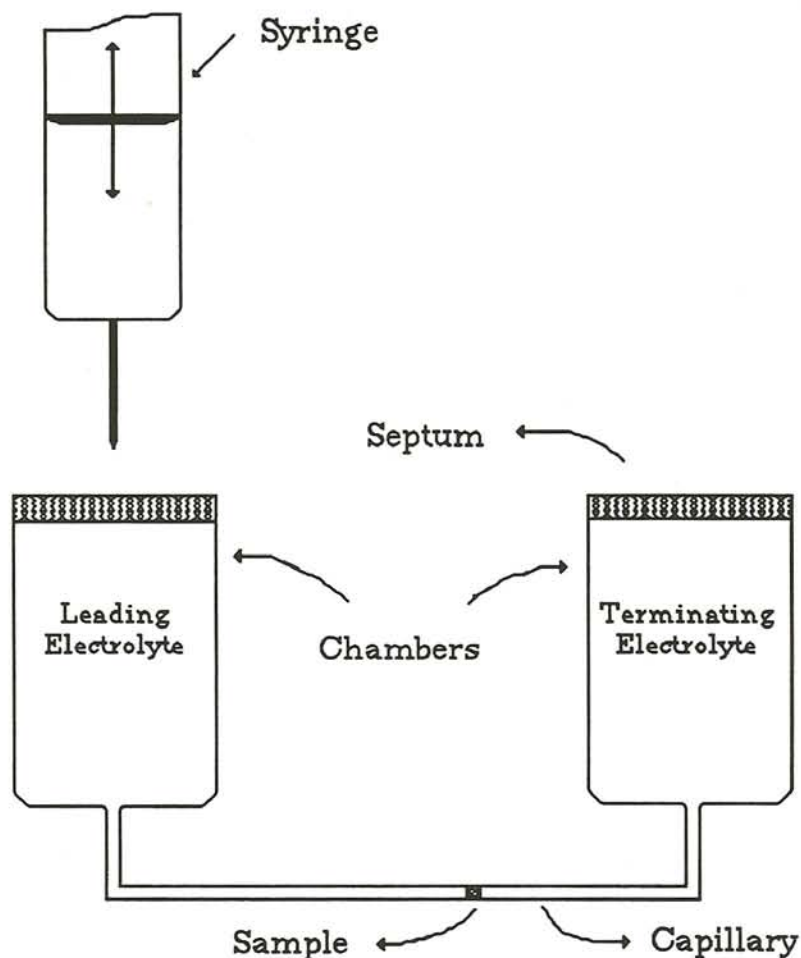


Figure 2.6: A simplified diagram of the ITP device and a syringe.

It can be noted that each chamber and the capillary must be clean prior to the filling process. Hence, if any of the chambers is not clean, it must be emptied, filled with distilled water, and then re-emptied. The

cleansing of the capillary, however, is achieved through cleaning of the chambers since, whenever a sample solution is injected into the capillary, each of the electrolytes exhibits new properties due to its contact with the sample solution.

Upon receiving a request to perform an electrophoresis experiment, a robot must first examine whether the chambers and the capillary are clean. Assuming they are clean, it can proceed with filling a syringe with the leading electrolyte solution and emptying it into the corresponding chamber. Next, the electrolyte solution fills the left portion of the capillary (cf. Figure 2.6) and thereafter it is examined for the presence of air bubbles. If bubbles are detected, the robot must clean it out again and re-fill it, a process which may have to be repeated several times.

However, if no air bubbles are detected, the robot proceeds with filling the other designated chamber (i.e., the terminating electrolyte chamber). Next, it examines the right segment of the capillary for presence of air bubbles after it has been filled with the corresponding electrolyte. Again, if air bubbles are present, it proceeds with cleaning as in the case of the leading electrolyte chamber. In the absence of air bubbles, a micro syringe is filled with the sample solution, and is injected into a designated location in the capillary as sketched in Figure 2.6. A micro syringe is necessary due to the capillary's fine structure which is intended to prevent a mixture of the sample solution and the electrolytes.

The set-up procedure is completed if no air bubbles are detected at the end of the injection process of the sample solution. However, if we detect air bubbles in the capillary, we must start from scratch with a complete set-up

procedure after cleansing each of the chambers and the capillary. It should be noted that the injection of the sample solution into the capillary is the most critical of the three operations and it often forces the entire set-up procedure to be repeated several times.

Once the set-up of the ITP device has been completed, a switch is activated which will start the separation process. At the completion of the experiment, a piecewise constant curve is produced. This curve represents the separated zones which exhibit distinct constant voltages in a staircase fashion starting with the leading solution and ending with the terminating solution. Figure 2.7 displays such a curve for a sample solution.

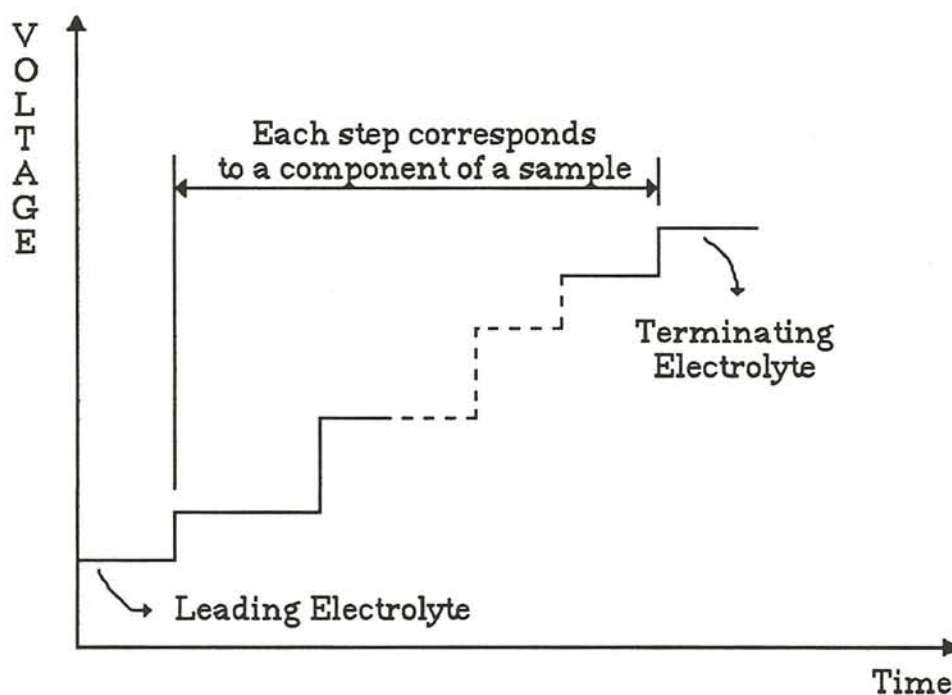


Figure 2.7: Response of a hypothetical ITP experiment.

By observing Figure 2.7, it can be concluded what are different components in a particular sample solution.

One particular laboratory to be installed in the SSF Life Science module is the so-called Fluid Handling Laboratory (FHL) (Schooley and Cellier, 1988). This laboratory, which may exist as a specific configuration of the GPL, hosts the environment desirable for the operation of the electrophoresis experiment. It is desired to teleoperate this laboratory from the ground using a Cartesian laboratory robot. Figure 2.8 represents the structure and related components of each part of the FHL which indeed exhibits the previously defined structure of the GPL.

1. Workspace

- a. Storage sections
- b. Electrophoresis section
- c. Robot territory

2. Materials

a. Liquids

```

Liquid_Carcinogenic_Supply_Leading-Electrolyte
Liquid_Caustic_Supply_Terminating-Electrolyte
Liquid_Toxic_Experiment-Related_Sample
    ..._..._..._..._..._..._..._..._..._..._..._..._..._...
    ..._..._..._..._..._..._..._..._..._..._..._..._..._...
    ..._..._..._..._..._..._..._..._..._..._..._..._..._...
    ..._..._..._..._..._..._..._..._..._..._..._..._..._...
Liquid_Nil_Supply_Distilled-water
Liquid_Toxic_Waste_Waste

```


3. Equipment

- a. Electrophoresis device or ITP
 - Leading Electrolyte chamber
 - Terminating Electrolyte chamber
 - Capillary
 - Others¹
- b. Measuring devices
 - Threshold sensors
 - Vision sensors or cameras
- c. Transporters
 - Micro syringes
 - Regular syringes
- d. Containers
 - Pressurized bladder bottles

4. Operator

- a. Robot

Figure 2.8: Decomposition of the FHL into Materials, Workspace, Equipment, and Operator entities.

The FHL is supposed to contain the leading electrolyte solution, the terminating electrolyte solution, distilled water, one or several sample solutions, the ITP, threshold type sensors, cameras, syringes, and a robot.

¹ Others are parts of an actual ITP device which are not included. An ITP has a power supply and several switches which are necessary to allow only a chamber and part of its attached capillary or just the capillary's segment that is devoted to the sample solution to be filled.

Pressurized Bladder Bottle

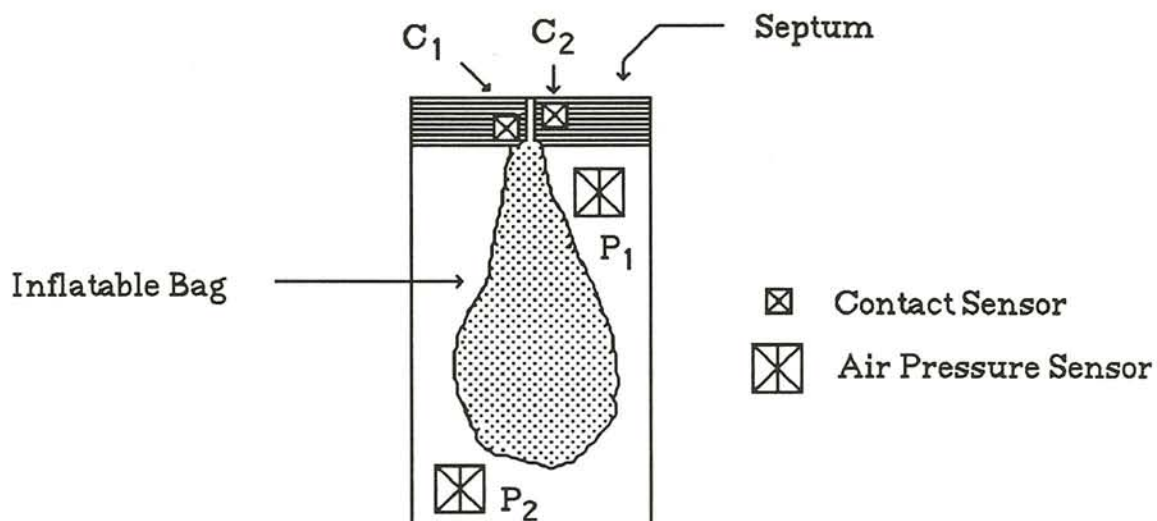


Figure 2.9: The pressurized bladder bottle and two types of threshold sensors.

It should be recalled that this experiment is to take place in the microgravity environment (i.e., in Space), and as a result, there must be no air/liquid interfaces that are not controlled by surface tension. A device called *Pressurized Bladder Bottle* (Zeigler *et al.*, 1988) is used to replace containers, beakers, and alike. The pressurized bladder bottle is outfitted with various sensors as depicted in Figure 2.9. The pressurized bladder bottles and their sensors will be detailed in 3.23 together with a description of their usage.

The chambers of the ITP device as depicted in Figure 2.6 must be redesigned in order to avoid the problem of air/liquid interface presented by

the chambers. This is an ongoing research activity investigated at the Center of Separation Science (CSS) of the University of Arizona.

As mentioned in Section 2.1, we are not concerned with the Workspace entity of the FHL. The Materials are considered to be *either* of the kind 'supply' or 'waste' or 'experiment_related' with their phase being 'liquid' combined with various types of health_hazards as outlined in Figure 2.8. We have considered only the minimum number of solutions required for an experiment; a realistic laboratory, however, hosts various kinds of sample solutions as well as leading and terminating electrolytes and rinsing solutions.

The third member of the FHL is the entity *Equipment* which consists of the ITP, the measuring devices, the transporters (i.e., the syringes or motorized pipettes), and the pressurized bladder bottles. The measuring devices are either threshold type sensors or vision sensors. The threshold sensors, shown in Figure 2.9, are activated upon reaching their threshold (i.e., binary), and can be used for examining the amount of a solution present in any of the containers. The output of the vision sensors is provided in the form of image frames; these are used for more critical tasks such as examining the presence of air bubbles in the capillary or to make sure that the micro syringe is properly located at the interface to the capillary where a sample solution must be injected.

The specific number of measuring devices varies. Considering the control operation of the electrophoresis experiment, we should assume that there exist at least eight threshold type sensors and one vision sensor. One threshold type sensor is used for each of the chambers and the capillary,

three more are used for the containers with the leading and the terminal electrolytes and the sample solution, one is used for the distilled water container, and finally one such sensor is used for the liquid waste container, while the vision sensor is used for the examination of air bubbles.

The threshold type sensors monitor the amount of the leading electrolyte, the terminating electrolyte, and the Sample solutions injected into the chambers and the capillary. They are special purpose sensors with a predetermined denomination. The vision sensor, such as a video camera, provides various types of information, and can be utilized (reassigned) for many purposes.

To include diagnosis and subsequent recovery procedures, more sensors of the threshold type are required, as well as possibly more advanced vision sensors.

The last component of the laboratory is the *robot* which, to a certain degree, must exhibit human expertise in completing its tasks as mentioned earlier. The ultimate goal, of course, is to introduce cognitive capabilities into a robot which will be responsible for conducting a subset of experiments which will be available at the GPL.

2.4 Event-Based Control

In the previous section, we described the set-up process of the electrophoresis experiment as a sequence of operations, each depending on the previously completed operation. Equivalently, it can be said that the FHL needs a control scheme (for both *static* and *dynamic* aspects) to carry out a series of operations. Hence, we must decide what type(s) of control are necessary and adequate for the experiments which will be hosted by the GPL.

Traditionally, *classical control theory* offered control schemes which have been used for several decades in isolation, and which are currently also being used as part of a methodology called *Intelligent Control* (Saridis, 1977, 1979, 1983). An intelligent control scheme is an intersection of *Control Theory*, *Artificial Intelligent*, and *Operations Research*. The control intelligence is hierarchically distributed according to the principle of decreasing precision with increasing intelligence. An intelligent controller is composed of the following three basic levels of controls :

- (i) *The organizational level* : accepts and interprets the input commands and related feedback from the system, defines the task to be executed, and divides it into subtasks in their appropriate order of execution.
- (ii) *The coordination level* : receives instructions from (i) and feedback information from the process for each subtask to be executed, and coordinates the execution at the lowest level.

- (iii) *The hardware control level* : usually involves the execution of a certain motion, it requires the mathematical model of the process, the assignment of end conditions, and a performance criterion or cost function defined by the coordinator.

This type of control scheme provides the basis upon which development of systems such as intelligent machines, space technologies (such as the SSF), integrated manufacturing linked with robotics, and c³I (Command, Communication, Control, and Intelligence) (McKinley, 1989) are possible. It is clear that all these systems are very complex, and conventional controllers alone are not able to control/manage them. Thus, there arise the need for intelligent controllers to control any of the above systems in various degrees of detail.

It is beyond the scope of this work to engage in a complete analysis of intelligent controllers. Nonetheless, it is necessary to consider two aspects of such controllers: what form of representation is necessary?, and what type of control strategy is pertinent? The question of representation is postponed until section 2.5. The other question, control strategy, is examined and explained in some detail next.

Generally, a control problem should be examined with respect to the available representation schemes (i.e., *Continuous, discrete-time, discrete-event, and qualitative*). Presently, we assume that the dynamics of an electrophoresis experiment will be represented as a discrete-event model.

Classical and intelligent types of control are usually applicable at various levels; however, some control algorithms are not usually useful

and/or justifiable for certain applications. For example, conventional control algorithms are applicable at the lowest levels of control while intelligent controllers are mostly used at the higher/highest levels of the control hierarchies². An intelligent controller is considered to be superior to a conventional controller for hierarchical levels of control since it relieves unnecessary queries to the data acquisition subsystem to sample a process at regular intervals of time. Types of control that are based upon the discrete-event representation of systems are called *event-based (Eventistic)* control (Meystel and Luh, 1987).

In the event-based control approach, a process can be partitioned into several phases such that they are distinguishable from each other (Zeigler, 1989). Thus, each phase can be considered as a simple process which is scheduled (predicted) to take *some* time, where the time can be either measured or obtained analytically. Each of these phases is associated with a minimum time, t_{min} , and a time window, t_{wind} , which specifies the acceptable variability of the execution time. Therefore, the lower limit of an acceptable time for completion of a process is t_{min} while the upper limit is t_{max} which is defined as follows:

$$t_{max} = t_{min} + t_{wind}$$

² These types of control (classical and intelligent) are not mutually exclusive. Indeed, usually intelligent controllers depend on conventional controllers for their inputs.

Thus, an acceptable time for an event, t_{event} , to occur can be specified as follows:

$$t_{event} \in t_{min} + [0, t_{wind}] \quad \text{or}$$

$$t_{min} \leq t_{event} \leq t_{max}$$

A higher-level model can move through its phases as long as it receives the sensors' outputs (from its lower level controllers) within the time interval, t_{event} . Hence, the higher-level model starts in an assumed phase and stays in it for t_{min} . If the sensor's state³ (i.e., a state which is responsible to move the process from one phase to another) is changed during this time interval, t_{min} , it is recognized as a *too-early* signal, an error, and relevant actions should be taken. However, if a sensor's response arrives during t_{event} , it is interpreted as a successful signal. Therefore, receiving sensory outputs within proper time intervals, t_{event} , causes subsequent processes or operations to continue.

The only other possibility is that the model does not receive an input from the corresponding sensor within the time interval bounded by t_{max} . This failure to respond from the appropriate sensor is interpreted as an error implying a *too-late* error signal. Hence, upon recognizing that the process is not completed within t_{event} , the process should be stopped and appropriate error messages be directed to a fault diagnoser or any other appropriate agent. A graphical representation of an event-based control

³ Here, we have assumed that only changes in one state-variable are sufficient to move a process from one phase to another. In general, it may be required to consider several such state-variables to correctly represent a process.

logic, along with the block diagram of its corresponding process, is shown in Figures 2.10a and 2.10b.

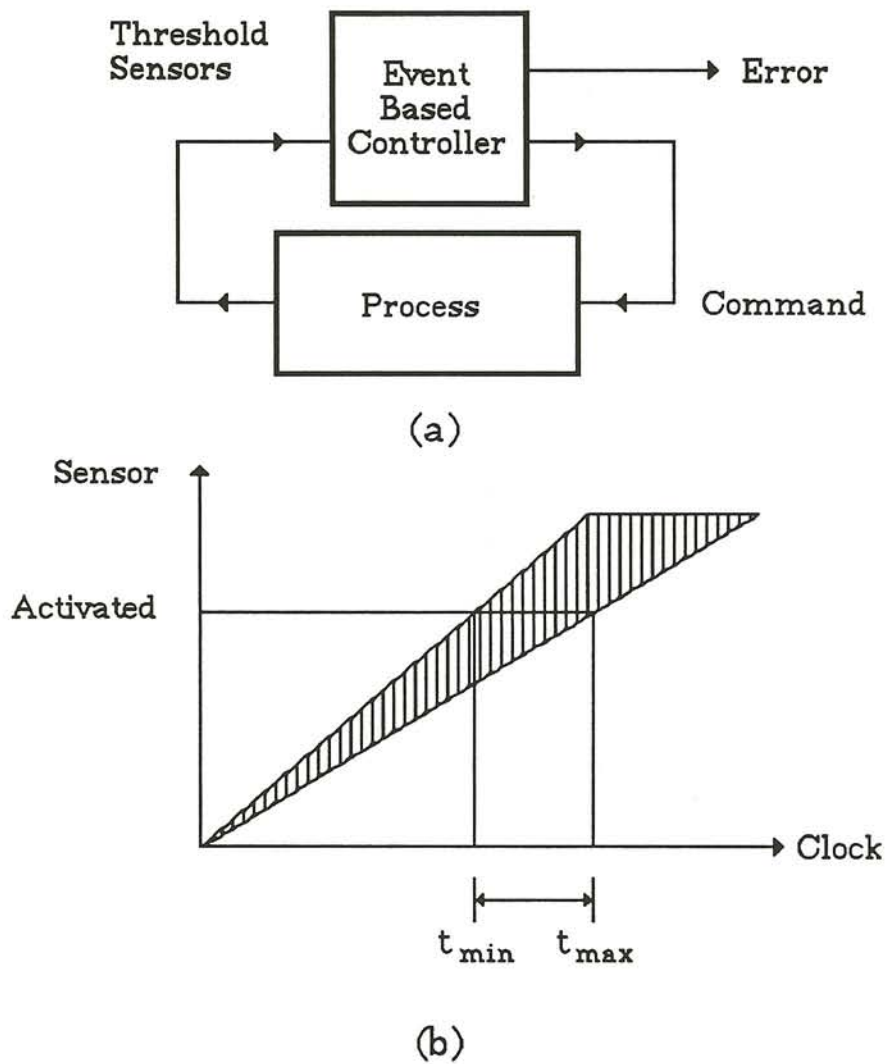


Figure 2.10: Representation of event-based control. (a) block diagram; (b) graphical representation of a process.

By examining Figure 2.10b, the event-based control strategy shows that the produced sensory outputs do not have to be very precise. In other words, sensors exhibit threshold-like characteristics, and precision is the responsibility of the clock and not the sensor. Nonetheless, it must be noted that generating time windows requires accurate correlation of sensory outputs to significant variables of the model (Zeigler, 1989).

One very important advantage afforded by an eventistic type of control is the ability of generating *nonnumeric* error messages, such as *too-late* and *too-early*, that can carry important information for a diagnostic unit. The reasoning procedures afforded by these forms of error messages in comparison to error signals containing computed values is very important for efficient and justifiable higher levels of control. In the following chapter, an example of an event-based controller as applied to certain operations related to the electrophoresis experiment is presented. Next, a mathematical formalism is presented which forms the basis for the design of event-based controllers.

2.5 DEVS Formalism and Discrete-Event Representation of Dynamical Systems

The *Discrete Event System Specification* (DEVS⁴) is a formalism introduced by Zeigler (Zeigler, 1986) that provides a formal basis for specifying models expressible within discrete-event simulation languages

⁴ A complete description of the DEVS formalism is given in (Zeigler, 1976, 1984)

such as SIMSCRIPT, and SIMULA. The DEVS formalism, much like DESS (the Differential Equation System Specification) or the automaton formalism, attempts to represent real world relationships within the constraints of its formalism. Obviously, the objective of each formalism is to faithfully and adequately represent a world view representation of systems.

These formalisms, however, differ in how easily they can accomplish their tasks. One very basic criterion for selecting a formalism is its relative *expressive power*. By the expressive power of a formalism is meant the class of systems it is able to faithfully represent. Hence, formalisms can be compared to each other by asking whether every system specified, or more precisely simulated by a model, in one can also be represented by a model in another. The *simulation efficiency*, defined as time and space required to simulate systems in a specified formalism, is another criterion in selecting a formalism. There exist other aspects, such as *convenience* and *versatility*, which contribute to the selection process of a formalism which are not addressed here. An in-depth discussion of the world view of formalism is given by (Zeigler, 1984).

Considering the brief definitions of the expressive power and the simulation efficiency, the DEVS formalism provides the necessary tools for the design of a large-scale system such as the SSF and consequently the GPL (Zeigler, 1984).

DEVS-Scheme (Zeigler, 1986), is a lisp-based environment for modeling and simulation of discrete-event systems based on the DEVS formalism and its associated abstract simulator concepts in PC-Scheme

(Texas Instruments, 1985). This modeling language provides an environment which supports building models in a *hierarchical*, modular manner since it is closed under composition (Wymore, 1988). Thus, the development of large-scale models is made possible by using classes *atomic-models*⁵ and the *coupled-models*⁶.

Atomic-models are constructed by using the following structural frame:

$$M = \langle X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a \rangle$$

Where

| | |
|-----------------------|------------------------------|
| X | set of input ports |
| Y | set of output ports |
| S | set of states |
| δ_{int} | Internal Transition Function |
| δ_{ext} | External Transition Function |
| λ | Output Function |
| t_a | Time Advance Function |

The above seven-tuple description is completely described in (Zeigler, 1984). However, a brief explanation of the elements of the seven-tuple, M , is necessary for understanding the discrete-event representation of dynamical models, and subsequently, event-based controllers.

⁵ Atomic-models are primitive components which are described in DEVS-Scheme.

⁶ Coupled-models describe how various models, which could be atomic-models or otherwise, are connected to each other to form more complex models. In other words, it is a class which embodies the hierarchical model composition.

The set X contains all possible external events (i.e., inputs) that are received by a model. The set of outputs that are to be sent out is contained in Y . The state variables are represented by the set S . In DEVS-Scheme, there exist two state variables (i.e., *sigma* and *phase*) which are usually present. The time an event (external or internal) is to occur is specified by *sigma*, while the state variable *phase* specifies the current phase of the model. The Internal Transition Function, δ_{int} , specifies the next state that the model transits to after the time *sigma* has elapsed, provided that no external event has arrived at an input port. The External Transition Function, δ_{ext} , specifies how the model should change its state when an external event is received. Once the time assigned by *sigma* has elapsed, the model is scheduled for its next internal transition. The Output Function, λ , generates an output just before an Internal Transition Function takes place. The last element of M is the Time Advance Function, t_a , and it is responsible for timing the internal transitions. Whenever the state variable *sigma* is present, this function simply returns its value.

Atomic-models are coupled to each other by utilizing the class coupled-models as mentioned earlier. Coupled-models specify how to couple several component models whether they are atomic-models, coupled-models, or a mixture of both, to form a larger model. Having the freedom of connecting atomic-models and coupled-models together translates to hierarchical model constructions.

Throughout this work, only discrete-event models of systems will be considered. Thus, it is appropriate to ask the question to what type of systems this formalism can be applied.

A model, usually, is specified using some type of formalism depending on its dynamics and/or its structure. The essence of the DEVS representation of systems lies in the construction of External Transition (δ_{ext}), Internal Transition (δ_{int}), and Time Advance (t_a) Functions. Sometimes, it may be necessary to extract these functions out of another model expressed in another formalisms such as DESS. It has been suggested by (Zeigler, 1984) that there exist several techniques⁷ for the construction of δ_{ext} , δ_{int} , and t_a depending on the available knowledge about the original system. These are:

- i. The original system is analytically tractable.
- ii. The original system is described by a set of differential equations which can be simulated in advance.
- iii. No models exist, but experimental studies with the real system are permitted.
- iv. The simulation system learns the DEVS model structure on-line.

All these methods require previous knowledge of the system whether through simulation or by any other means. The objection may be raised

⁷ There exist appropriate measures which reduce the complexity of efforts in obtaining these functions (Zeigler, 1984)

that, once a system has been analyzed, its behavior is completely known. Then, what is the the purpose of representing it in the DEVS or any other formalism? This view may seem legitimate at first glance; however the justification and reasoning lies in the use of models as components in a *multicomponent* DEVS modeling effort (Zeigler, 1984). The benefit of transforming DESS or other models into DEVS models becomes even more evident when all components of a large-scale system are represented in such a form, since this permits extensive studies of large-scale/complex systems to be performed.

CHAPTER 3

MODELING OF INTELLIGENT CONTROLLERS

3.0 Outline of the Chapter

In this chapter, we will discuss the modeling of the Fluid Handling Laboratory (FHL) components, as well as how a robot should act in order to successfully carry out its responsibilities as they relate to the control of the FHL components in an intelligent manner. Section 3.1 presents a *modeling methodology* for systems that must support intelligent capabilities. In Sections 3.2.1, 3.2.2, and 3.2.3, models of the camera, the rack, and the ITP device are described in detail. The modeling of the robot, with its static and dynamical decision making powers which depend on the models of the camera, the rack, and the ITP device, is treated in Section 3.2.4. Furthermore, the filling process of one of the chambers is examined in detail to illustrate the coordination between the *operational* and the *controlled* models of the ITP device. The generation of meaningful *error messages* by an event-based controller is also discussed. The last section of this chapter examines the interactions among the robot, the rack, the camera, and the ITP device.

3.1 Modeling Methodology

It was conveyed in the previous chapter that a modeling methodology applicable to intelligent controllers of large-scale and complex systems can be realized through the discrete-event formalism and the event-based control approach. There was, however, no precise indication of how a system and its intelligent controller should be represented.

In representing a process, the underlying control objectives play an important role in the model construction (Zeigler, 1984, 1987-a). Therefore, there can exist various types of models of the same system, each devoted to a specific purpose. That is, if operations pertinent to a system are being studied, there must exist an *operational model* which captures such behaviors. Likewise, there must exist another model, a *control model*, of the same system used by its controller which provides knowledge not available in the *operational model*. Thus, the models which are intended for 'operation' and 'control' of a system must provide different types of information, some of which may be provided by either model. It should, therefore, be evident that the number of models associated with a system is related to the *number of objectives* at hand, each requiring a class of information not available in any other model¹.

Although it is possible to map all the desired/required features present in every model into a single model, the crucial underlying objectives of intelligent agents, which depend upon the hierarchical

¹ Note that in modeling by means of differential and integral calculi, models of the system and its controller are usually merged together. Therefore, the description of the goals and the control concepts are usually represented implicitly.

approach, is thereby ignored. There are also other issues to consider such as excess amount of knowledge and unavoidable increase in the complexity of the model. When there exists a large amount of stored information, the retrieval cost/efficiency of it can be quite high or even unacceptable. The tradeoff between the amount of stored information and the rate at which it can be retrieved is usually resolved by considering parameters such as cost, desired retrieval speed, and available hardware.

The available approaches to store information lie between completely centralized and completely decentralized schemes. Certainly, in designing intelligent agents, an adequate level of decentralization is necessary. The level of detail of the information, ranging from fine granularity to coarse granularity, determines the level of the complexity of a system. An adequate information granularity must be determined in order to represent the necessary level of detail for a particular purpose, not considering the proper level of detail in designing a system, can adversely affect its performance. If not enough information is stored locally, the requested task cannot be executed without asking frequently some other agent for information. This will reduce the system's performance. On the other hand, if too much information is stored locally, the access time will grow which again will slow the system down. Obviously, there must be an optimum amount of information or optimal information granularity which lies in between the two extremes.

To demonstrate and clarify the necessity of constructing various models of a system, the FHL is considered next. It was stated in Chapter 2 that our main focus is to study the FHL environment and in particular the

process of setting-up an electrophoretic experiment. Consequently, we must model a robot to conduct the electrophoresis experiment with the aid of an intelligent controller to monitor some desired operations as explained in Section 2.4. The capabilities required from an intelligent agent are assigned to the robot. However, the immediate controllers of equipment that is directly related to the ITP device, e.g., the controllers for the camera and the rack will not be assigned to the robot. In this study, these controllers are simply ignored. The exclusion of such controllers will cause no loss of generality since no significant additional insight is gained by including similar class of control capabilities.

It should be apparent from the above discussion that we will have *different models* of the ITP device: one to administrate the required control assignments, and another to reflect the actual dynamical operations. We may attempt to represent the ITP device by considering four models three of which will be modeled in this and the following chapters:

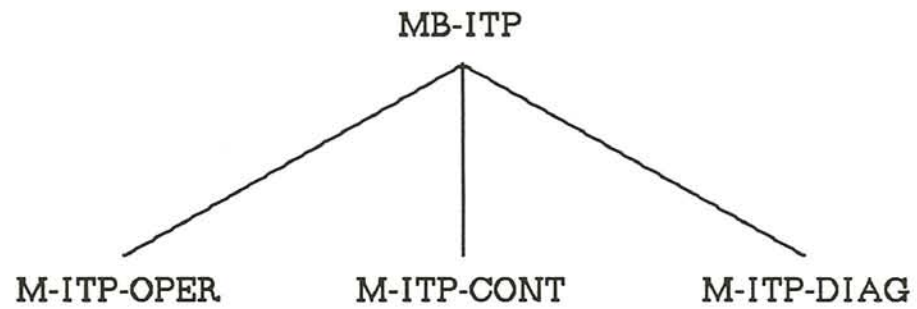
1. The first model is a reference model, MB-ITP, for the model base which provides the *most refined characteristics* of the real system. This model is usually characterized by a set of integro/differential equations (using the DESS formalism). It is this MB-ITP which will not be modeled in this Thesis. This is the task of another parallel Thesis (Wang, 1989). The detailed model of a real system, MB-ITP, can be used to generate (extract) models with either equivalent or less complexity (at a more coarse granularity). A higher aggregated model may be better suited for the intelligent controller, or other agents such as the diagnostic unit, due to its simplicity.

2. The second model, M-ITP-OPER, is the model of the system, ITP, that is being controlled, and it is devoted to the *operational tasks*. This *operational model* is considered to be *external* to the intelligent controller. It is this model that may eventually be replaced by the real system during a true implementation of the intelligent control strategy. It can provide external information not known internally to the controller. This model will be constructed as a discrete-event model which may eventually be derived from the MB-ITP in an automated manner as demonstrated by (Wang, 1989).
3. The third model, M-ITP-CONT, is another model of the ITP device that is *internal* to the intelligent controller. This *control model* is *internal* to the controller. It provides the type of knowledge that permits higher levels of control. The M-ITP-CONT is also represented as a discrete-event model.
4. The last model, which is also considered to be *external*, is used for diagnostic purposes. It is named M-ITP-DIAG. It provides the information that is necessary for a diagnostic unit (e.g., an expert system) in the event of an error. This model should be replaced by the real system during a true implementation of a diagnostic unit. We shall elaborate on this *diagnostic model*, M-ITP-DIAG, in Section 4.4.

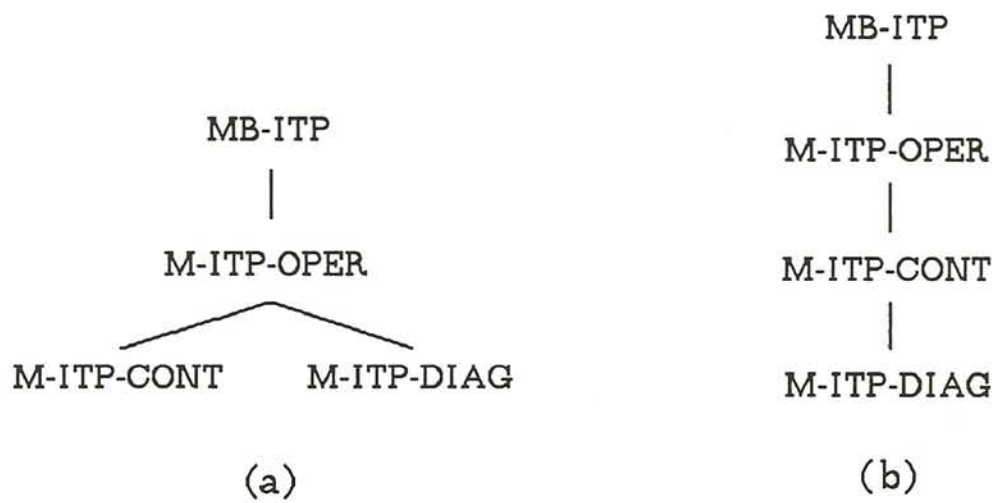
Note that the *internal* and *external* models of the ITP device must reflect various levels of detail; i.e., the M-ITP-CONT contains less detail than the M-ITP-OPER, which must represent the more detailed characteristics required at the operational level.

It should be apparent that the number of models of a real system generated from the reference model, MB-ITP, in the model base is directly proportional to the number of objectives necessary to represent the various desired characteristics of the real system. Therefore, a logical consequence is that, as the complexity and the number of capabilities of a system increases, we must expect that more models will be necessary to validly represent all the facets of the real system that are being considered in the design.

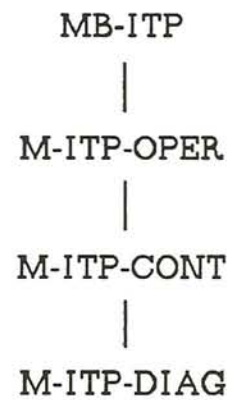
In Figure 3.1, we have illustrated that the M-ITP-OPER, the M-ITP-CONT and the M-ITP-DIAG are to be extracted from the reference model, MB-ITP, in the model base. It is also quite common to start with a less complex model, and elaborate on it to generate a more detailed model through the process of *step-wise refinement*. This is the approach that we take when we generate the initial entity structure of the model. However, this route is not amenable to automation. More on elaboration (disaggregation) and simplification (aggregation) of models will be presented in due course.



(a)



(a)



(b)

Figure 3.1: Model-base, operational, control, and diagnostic models of the ITP device. (a) two levels of abstraction; (b) three levels of abstraction; (c) four levels of abstraction.

It is not mandatory to always refer to the reference model in the model base to generate all the other models from it as depicted in Figure 3.1a. For example, it may be advantageous to generate the M-ITP-OPER model first, and then deduce the M-ITP-CONT and the M-ITP-DIAG models from it (cf. Figure 3.1b). The *control* and the *diagnostic* models, as shown in Figure 3.1b, may be constructed from the *operational model* such that they belong to the same level in the hierarchy. Another possibility is to develop the M-ITP-OPER model from the MB-ITP, then obtain the M-ITP-CONT model from it, which in turn is used for the realization of the M-ITP-DIAG model (cf. Figure 3.1c). This is possible by successively simplifying the models from the top to the bottom as shown in Figure 3.1c, indicating four levels of complexity of the ITP device. The three discrete-event models mentioned above can be realized through abstractions, i.e., by means of *homomorphic* and/or *isomorphic* relations from the reference model or any other model (Wymore, 1988).

The concept of homomorphism states that we can have many different models of the same reality, a real system, where one is a consistent simplification of another, or one is a consistent elaboration of another. Nevertheless, through homomorphism we can assure that all these models represent the same reality with various degrees of complexity. In order to assure homomorphism, three mapping functions must be defined. These mapping functions relate the *states*, *inputs*, and *outputs* of any two systems to each other. Moreover, the next state and output functions, similar to those defined for the DEVS formalism, must also be preserved by the defined sets of mapping functions (Wymore, 1988).

The isomorphism concept is a specialized form of homomorphism. In order for two models or systems to be isomorphic in addition to being homomorphic, the relations specified in the three mapping functions must all be one-to-one relations. Isomorphism may be thought of as a scheme of renaming all the states, inputs, and outputs of a model, resulting in another model. Subsequent generations of models, as explained in Figure 3.1, may be obtained through homomorphism from a previously derived model. Methodological issues concerning the forms of abstractions that are needed and what must be preserved in a morphic relation are discussed in (Wymore, 1988).

3.2 Modeling the Components of the FHL

Having discussed the modeling methodology and its implications, we will proceed by representing the camera, the rack, the ITP device, and the robot by utilizing the DEVS-Scheme environment. In the following four subsections, each of the components of the FHL will be described in accordance with the elements of the seven-tuple, M , that was described in Section 2.5.

3.2.1 Modeling the Camera

The camera is modeled rather simply at a high level of abstraction. It operates without any controller (cf. Section 3.1). Hence, we will omit the explicit representation of the camera's model in this thesis, and instead describe the elements of its seven-tuple, M . The role of the camera in setting-up the electrophoresis experiment is to examine the presence of air bubbles in the capillary or any of the chambers. The camera's response can be a *zero* (signifying the absence of air bubbles), a *one* (signifying the presence of air bubbles), and a *nil* (signifying its inability to answer the question, either because it didn't understand the question correctly, or because it cannot decide whether or not there are air bubbles present). The set S contains the state variables *phase*, *sigma*, *I/O-port*, *I/O-value*, *leitp*, *teitp*, and *saitp*. The first four state variables are shared by all the models. The state variables *phase* and *sigma* were explained in Section 2.5. Both the *I/O-port*, input/output port, and the *I/O-value*, input/output value, are elements of both the input set X and the output set Y . The state variables ending with *-itp* indicate that the ITP device is under the examination of the camera. Their first two letters, described below, specify the segment of the ITP device that is under observation:

LE: leading electrolyte solution

TE: terminating electrolyte solution

SA: sample solution

For instance, the *leitp* indicates that the chamber of the ITP device containing the leading electrolyte is examined for the presence of air bubbles. Both the input set, X , and the output set, Y , of this model as well as all the other models are considered to be pairs of the form (*I/O-port*, *I/O-value*). Each pair, therefore, is a message which is either received (i.e., input) or sent (i.e., output) by any of the models. The set X containing the accepted inputs for this model is a subset of the output set Y of outputs generated by the robot (cf. Appendix A). Consequently, the camera ignores all other received inputs.

The first element of the camera's output pair, Y , informs the robot of the identity of the examined segment of the ITP device. Permissible values are *leitp*, *teitp*, and *saitp*, i.e., a state variable of the enumerated type *I/O-port*. The second element of any pair in the output set Y , *I/O-value*, contains either a *zero*, a *one*, or a *nil*. Thus the output set Y consists of pairs such as (*leitp*, 0), (*leitp*, 1), (*leitp*, nil), and so on.

The external transition function, δ_{ext} , which is responsible for the external inputs, has to select the correct object based on the content of the second element, *I/O-value*, of the received message from the robot. Once the correct segment of the ITP device has been identified (i.e., one of the chambers or the capillary), upon receiving a valid external input, the camera assigns either a *zero* or a *one* to the *I/O-value* at the termination of the *sigma*. The value of the *I/O-port* specifies which segment of the ITP device was examined. Moreover, the *I/O-port* should be consistent with the *I/O-value* which was received from the robot. That is if the robot requests the examination of the leading electrolyte chamber, the camera should

examine what was requested, and return a message containing the requested response. The state-variable, *phase*, is updated at the beginning and at the end of the external transition function, to keep track of the camera's present status; this can provide important information to an event-based controller. Once the δ_{ext} is completed, a message which contains one of the elements of the output set Y , (*I/O-port*, *I/O-value*), is sent out by the Output Function, *out-camera*, to the robot.

The Internal Transition Function, δ_{int} , which is executed immediately after the Output Function, enters a state (i.e., *phase* = passive and *sigma* = ∞ ; refer to Section 2.5) where the model stays idle until its future activation upon receiving another valid input from the robot.

3.2.2 Modeling the Rack

The rack can be considered as a model which is *static*, *dynamic*, or *static-dynamic*. The *static* model of the rack provides information pertaining to the materials and the various types of equipment (cf. Figure 2.8) required to conduct the electrophoresis experiment. For instance, if there does not exist a sufficiently large amount of a desired sample solution, there will be no need for the robot to start filling the ITP's chambers. The *dynamic* model of the rack can provide the behavioral activities which take place in the workspace of the rack. These activities are the filling/emptying of the pressurized bladder bottles and the transporters, displacement of the various pieces of equipment, picking-up

the transporters, and so on. The consideration of such processes is important for an actual implementation of an intelligent controller. Nonetheless, the modeling of such activities in our prototype environment would not reveal any additional features from which we could learn something useful, but instead, it would only complicate the matters at hand. For this reason, the rack has been viewed as a *static* model. In a real implementation, of course, the model of the rack should exhibit both the *static* aspects and the *dynamical* behavior of the activities which take place in its workspace.

The actual DEVS model of the rack is also described. The number of state variables are related to the number of entities desirable for the FHL (Table 3.1). The list of state variables shown in Table 3.1 only considers a small number of entities which must be present in the FHL.

The input set X and the output set Y are identical to those given for the camera except for the actual contents of the *I/O-port* and the *I/O-value*. This model is also similar to the previous model with regard to how its δ_{ext} , δ_{int} , and λ are structured. The rack's δ_{ext} updates the state variables related to the activities of the leading electrolyte chamber, the terminating electrolyte chamber, and the capillary depending on the received inputs. The time, *sigma*, which is required to complete the δ_{ext} is arbitrarily assumed to be 1 unit of time in accordance with the earlier assumption. Obviously in a real implementation, *sigma* would have to be chosen correctly in order to reflect the durations of the actual activities which must be carried out. The δ_{int} of the rack resets to zero the state variables *volle*,

Table 3.1: Selected state variables chosen to represent the rack.

| State Variables | Description of some of the Selected State Variables for the Rack | Default Values |
|------------------------|---|-----------------------|
| TSYR: | Total number of syringes available. | 100 |
| TMSYR: | Total number of micro syringes available. | 20 |
| SYRLE: | Number of syringes used for the LE chamber. | 0 |
| SYRTE: | Number of syringes used for the TE chamber. | 0 |
| SYRSA: | Number of syringes used for the SA chamber. | 0 |
| VOLLE: | Amount of LE used during an experiment. | 0 |
| VOLTE: | Amount of TE used during an experiment. | 0 |
| VOLSA: | Amount of SA used during an experiment. | 0 |
| VOLDW: | Amount of distilled water used during an experiment. | 0 |
| VOLWA: | Amount of waste accumulated during an experiment. | 0 |
| TOTLE: | Total amount of LE available at the start of an experiment. | 400 |
| TOTTE: | Total amount of TE available at the start of an experiment. | 200 |
| TOTSA: | Total amount of SA available at the start of an experiment. | 10 |
| TOTDW: | Total amount of distilled water available at the start of an expr. | 800 |
| TOTWA: | Total amount of waste deposited at the start of an experiment. | 0 |
| SIGMA: | Defined in section 2.5 | ∞ |
| PHASE: | Allowable phases. | PASSIVE |
| OPRT: | Allowable input ports. | NIL |
| OVAL: | Allowable input values. | NIL |

Where: LE: leading electrolyte
 TE: terminating electrolyte
 SA: sample

volte, *volsa*, *voldw*, and *volwa* upon completion of the set-up procedure. The meaning of these state variables is described in Table 3.1. This is necessary since it may be desirable to know how much of each commodity was used for a particular experiment. Furthermore before starting an experiment, the robot must inquire about the currently available commodities, which are required to complete the experiment. The rack's λ can be utilized to notify the robot of its current status which will prevent it from continuing with the set-up procedure as soon as it receives an information indicating insufficient commodities.

3.2.3 Modeling the ITP Device

In Section 2.2, it was stated that each chamber may be filled, emptied, and cleaned several times during the set-up procedure where the filling and the emptying operations were considered to be *primitive operations*. The cleaning process is a *compound operation* that consists of the sequence of *primitive operations*: emptying the chamber, filling it with distilled water, and finally re-emptying it. Recall that the segment of the capillary which contains the sample solution can only be filled and not emptied (cf. Section 2.3).

In order to monitor the volume of the liquid inside the chambers and the capillary with respect to the primitive operations, each of them is outfitted with two types of sensors: *air pressure sensors* and *contact sensors* as depicted in Figure 2.9. The underlying operations of these sensors were

explained in Section 2.3. An air pressure sensor becomes activated when the volume inside a chamber, for example, has reached some prespecified limit (see Figure 3.3).

A randomly selected time duration, representing the activation of a sensor, simulates the behavior of an actual air pressure sensor. The time required to complete a particular operation, recognized as the activation of a sensor, is not always exactly predictable since its behavior depends on one or more parameters. If we consider a primitive operation to be a function of a constant flow-rate only, the time which is needed for its completion can be calculated analytically if we know the desired volume of the liquid and the rate at which it enters/leaves the chamber. However, other parameters which may influence the characteristics of these *primitive* operations are e.g., the diameter of the chamber, the angle of the syringe inside the chamber, the viscosity of the liquid, the current ambient temperature, and the selected gravity force in the GPL.

Figure 2.9 shows that each chamber or capillary is outfitted with two kinds of sensors: air pressure sensors and contact sensors. There are two identical air-pressure sensors and two identical contact sensors. We shall refer to P_1 and C_1 as the *primary* sensors, and to P_2 and C_2 as the *backup* sensors. The primary sensors are used for control purposes, while the backup sensors are used for diagnostic purposes. The reason for including two sensors of each kind is to utilize the backup sensor for confirmation of the primary sensor in the event of an error.

The tasks *contact* and *release* (cf. Figure 3.4) include the insertion of the syringe into the chamber. That is, the filling operation begins with the

assumption that the syringe is already inside the chamber or the capillary. The model of the ITP device, therefore, only requires the air pressure sensors (e.g., P_1) and not the contact sensors (e.g., C_1). The exclusion of the contact sensors in the M-ITP-OPER is due to the assumption that the tasks *contact* and *release* will be conducted without encountering any errors.

The name M-ITP-OPER is selected to identify the operations filling, emptying, and cleaning which should be carried out in the set-up procedure of the electrophoresis experiment. The operational model of the ITP device, M-ITP-OPER, is presented in Figure 3.2. M-ITP-OPER contains three air pressure sensors (i.e., *le-press-sensor*, *te-press-sensor*, and *sa-press-sensor*). The *process-time* is calculated by dividing a desired volume, such as *vol-fill-le-itp*, the output of a random number generator (Figure 3.2), by a specified *flow-rate*. The *flow-rate* for simplicity is assumed to be constant.

The time duration for each of the primitive operations was obtained by experimenting with the ITP device² at the Center for Separation Science at the University of Arizona. The probabilities associated with each *process-time*, however, are selected according to the simulations' objectives. The longer cleaning operation is represented by estimating the total volume of the displaced distilled water in any of the chambers since the *flow-rate* is assumed to be constant.

² Each *process-time* and its associated probability, indicated by a random number generator's output, should be obtained from any of the applicable methods listed in section 2.5.

The contents of the *I/O-port* and *I/O-value*, are deducible from Figure 3.2 which are also some subsets of the robot's *I/O-value* and *I/O-port*, respectively (see Appendix A).

The purpose of the *ext-m-itp-oper*, δ_{ext} , is to represent the operations filling, emptying, and cleaning. When the *operational* model of the ITP device, M-ITP-OPER, receives a message requesting one of the above operations, it executes its δ_{ext} which would represent the reading in of a value from an air pressure sensor (see Figure 3.2). In Figure 3.6, it is shown that the air pressure sensor inside the leading electrolyte chamber (shown as 'Sensor' in Figure 3.6) is empty upon receiving the input (*LE, FILL*) from the robot (see ' X_{ITP}/Y_{robot} ' in Figure 3.6). The air pressure sensor *stays off* for 22.9 units of time (obtained by experimentation) before it *goes on*. The time which is required for the sensor to change its status from *off* to *on* depends on several parameters as mentioned earlier. The state variable *le-press-sensor* shown in Figure 3.6 as ' S_{ITP} ' represents such a change in the status of the air pressure sensor. Other state variables, *phase*, *oprt*, and *oval*, are also changed at the end of the filling operation. The M-ITP-OPER schedules *sigma* (equated to the *process-time*) in order to represent the duration of the filling operation or equivalently the behavior of the air pressure sensor. Upon completion of the δ_{ext} , a message with appropriate *I/O-port* and *I/O-value* is sent to the robot by using the *out-m-itp-oper*, λ . The ' Y_{ITP} ', as shown in Figure 3.6, demonstrates that the output (*LEITP, FILL*) is sent to the robot in order to notify it about its new status (i.e chamber is full). The *int-m-itp-oper*, δ_{int} , has no significance as

mentioned earlier except for setting the model into its idle mode upon completion of each operation.

⌘ *Atomic-model for the ITP Device*

```
(make-pair atomic-models 'm-itp-oper)
(send m-itp-oper def-state '(
  le-press-sensor ; air pressure sensor for the LE chamber: on/off
  te-press-sensor ; air pressure sensor for the TE chamber: on/off
  sa-press-sensor ; air pressure sensor for the SA capillary: on/off
  flow-rate       ; influx flow-rate
  process-time    ; time to complete filling, emptying, and cleaning
  phase          ; passive, le-fill, te-fill, sa-fill, le-empty, . . .
  sigma          ; 0 ↔ ∞
  oprt           ; output port: leitp, teitp, saitp
  oval))         ; output value: fill, empty, clean
(send m-itp-oper set-s (make-state
  flow-rate       1
  le-press-sensor 'off
  te-press-sensor 'off
  sa-press-sensor 'off
  process-time    0
  sigma          'inf
  phase          'passive
  oval           '()
  oprt           '()))
;;; Random Variables
(define (vol-fill-le-itp num)
  (let ((r random num))
    (cond
      ((<= r 35) 19.4)
      ((<= r 65) 22.9)
      ((<= r 98) 24.5)
      (else      29))))
;;; This random variables produces
;;; the necessary volume of the
;;; leading electrolyte that activates
;;; the air pressure sensor.
;;; Other random variables have identical structure as shown above.
(define (vol-empty-le-itp num) ...)
.....
;;; External Transition Function (i.e.,  $\delta_{ext}$ ):
(define (ext-m-itp-oper s e x)
  (case (content-port x)
    ('le (case (content-value x)
              ;; RECEIVED FROM ROBOT
```

```

('fill      ;; The time for the filling operation is defined as the
            ;; ratio of the vol-fill-le-ity to the flow-rate.
      (set! (state-process-time s)
            (/ (vol-fill-le-ity 100) (state-flow-rate s)))
      (set! (state-oprt s) 'leity)
      (set! (state-oval s) 'fill)
      (set! (state-le-press-sensor s) 'on)
      (hold-in 'le-fill (state-process-time s)))
('empty ... )
('clean ... )
('te ... )           ;; RECEIVED FROM ROBOT
('sa ... )           ;; RECEIVED FROM ROBOT
      (else (continue))) ;; NOT RECEIVED FROM ROBOT
;;; Internal Transition Function (i.e.,  $\delta_{int}$ ):
(define (int-m-ity-oper s)
  (if (not (equal? (state-phase s) 'passive))
      (passivate) ;; Whenever any of the operations filling,
      (continue) ;; emptying, and cleaning is completed, then
  ))           ;; passivate - it becomes idle.
;;; Output Function (i.e.,  $\lambda$ ):
(define (out-m-ity-oper s) ;; Send messages to robot when the
                          ;; filling operation is completed.
  (if (not (equal? (state-phase s) 'passive))
      (make-content 'port (state-oprt s) 'value (state-oval s))))

```

Figure 3.2: Atomic model of the ITP.

If the robot sends a request (i.e., carry out the filling operation) to a lower level controller, then at the same time it must start simulating a version of the filling operation in order to fulfil its intelligent capabilities. What is important to realize is that the M-ITP-OPER and the robot start the filling operation simultaneously (Figure 3.3). In reality, the robot has a model, M-ITP-CONT of the filling operation, and the M-ITP-OPER is replaced with a real filling operation.

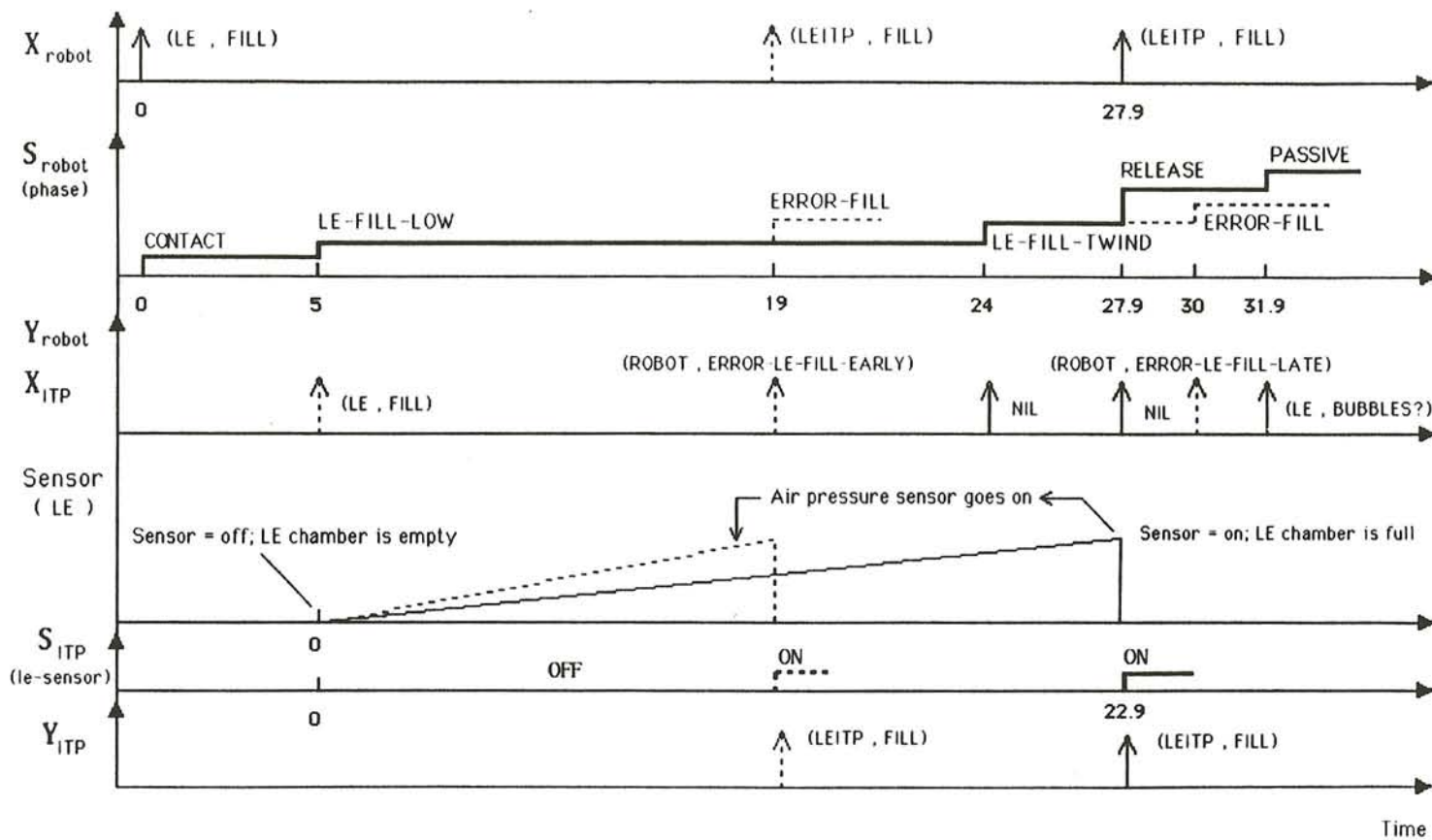


Figure 3.3: External, internal, and output activities for the ITP instrument and robot.

3.2.4 Modeling the Robot

Traditionally, the translation of an assigned task into the robot actuator commands involves the following three steps:

1. *Task Level Planning*: Given a particular task, find the manipulated object's required motion.
2. *Robot Motion Planning*: Given the manipulated object's required motion, find the desired robot trajectory.
3. *Robot Controller*: Given the desired robot trajectory, find the robot actuator commands.

These three steps are sufficient for completion of most desired tasks in the absence of errors. Since these steps are usually assigned to the lower level controllers, while our objective is to design an event-based controller (i.e., a controller applicable at higher levels of control), we will not be concerned with steps 1, 2, and 3 as defined above. Our objective, as stated in Chapter 1, is to initiate a particular task, and use an event-based controller to monitor the correctness of the lower level actuator commands. Therefore, a given task that is assigned to an event-based controller can be translated into the following steps:

1. *Task Level Planning* : Given a particular task, decompose it into smaller tasks or a set of primitive operations.
2. *Robot Controller* : Given the primitive operations, assign them to appropriate lower level controllers.

Hierarchy of Controllers

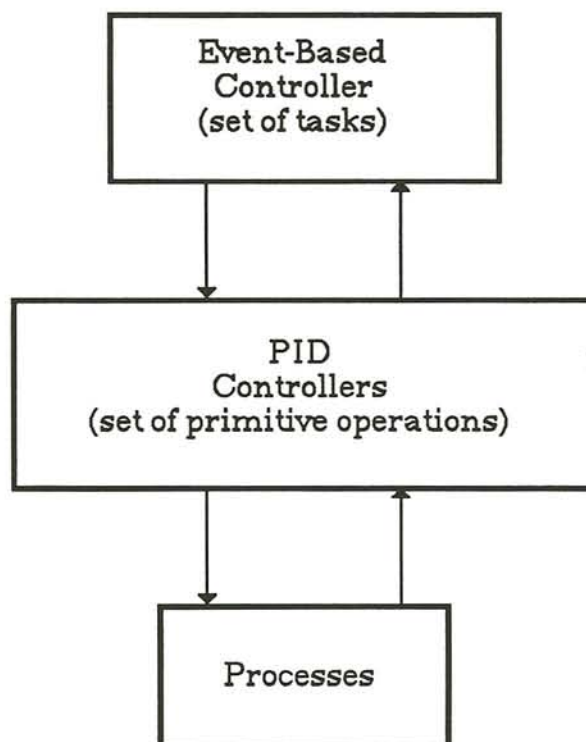


Figure 3.4: Hierarchy of higher and lower level controllers.

Notice that an event-based controller does not have any knowledge of the lower level controller's actuator commands which are responsible for the execution of primitive operations (cf. Section 2.4). Figure 3.4 depicts a block diagram representation of a set of PID controllers and their associated event-based controller.

In studying the set-up procedure of the electrophoresis experiment, the robot is viewed with respect to the following steps:

A. *Task Level Planning:* The necessary steps for the set-up procedure of the electrophoresis experiment, assuming the operations filling, emptying, and cleaning, will be conducted as follows:

1. Fill the LE chamber.
2. Examine the presence of air bubbles in the LE chamber.
3. Fill the TE chamber.
4. Examine the presence of air bubbles in the TE chamber.
5. Fill the capillary.
6. Examine the presence of air bubbles in the capillary.
7. Turn the switch on (i.e., set-up procedure is complete).

B. *Robot Controller:* Assuming the chambers and the capillary are clean, the event-based controller commands for the above tasks are:

1. Fill the LE chamber.
2. Proceed to step 3 if the filling operation is completed as scheduled, otherwise stop the operation and notify the responsible agent.
3. Examine the LE chamber for the presence of air bubbles.
4. Fill the TE chamber (if no air bubbles are present in the LE chamber), or empty/clean the LE chamber (if air bubbles are present in the LE chamber) and return to step 1.
5. Proceed to step 6 if the filling operation is completed as scheduled, otherwise stop the operation and notify the responsible agent.
6. Examine the TE chamber for the presence of air bubbles.

7. Fill the capillary (if no air bubbles are present in the TE chamber), empty/clean the TE chamber (if air bubbles are present in the TE chamber) and return to step 3.
8. Proceed to step 10 if the filling operation is completed as scheduled, otherwise stop the operation and notify the responsible agent.
9. Examine for the presence of air bubbles in the capillary.
10. Start the experiment (i.e., no air bubbles detected in the capillary) or empty/clean the chambers and the capillary (i.e., air bubbles are detected in the capillary) and repeat the set-up procedure.

The correctness of steps 1, 3, and 5 from **A** are considered to be the responsibility of an event-based controller utilizing the *operational (external)* and the *control (internal)* models of the operations necessary to complete the set-up procedure. Therefore, **B** contains steps 2, 5, and 8 in order to enforce the correctness of steps 1, 3, and 5 from **A**. When a *primitive* operation is completed within the scheduled time t_{event} (cf. Section 2.3), it is called a 'successful' operation. On the other hand, if the *primitive* operation was not completed according to the schedule, either the following commands from **B** (if an operation was completed earlier than expected), or the present operation from **B** (if the operation has not been completed within the allotted time) must be stopped. An *intelligent controller*, therefore, is able to monitor the correctness of the *dynamical* operations it is assigned to. The coordination between the *operational* and the *controlled* models, represents the *dynamical* role of a higher level controller which ensures such successful operations.

The steps 4, 7, and 10 from **B** select either the next prescribed activity from **A**, or a series of other activities determined by the event-based controller. The activities listed in steps 4, 7, and 10 from **B** include some additional operations (i.e., emptying and cleaning) which were not previously scheduled, assuming 'perfect' completion (i.e., no air bubbles) of steps 1, 3, and 5 from **A**.

A distinction between a 'successful' completion and a 'perfect' completion of an operation should be made. The decision which is based upon the perfect completion of an operation is *static* in nature; a *successfully completed* operation is one which was completed within the allotted time window. However, at this point, we don't know yet whether there are air bubbles present. The operation must be evaluated with respect to the absence of air bubbles before it can be called a 'perfect' operation. As mentioned earlier, an event-based controller is responsible for both the *static* and the *dynamical* behaviors involved in carrying out a task.

The filling operation of a chamber (and similar operations) in the GPL environment requires some other activities such as insertion of a syringe into the pressurized bladder bottle. We have defined the tasks *contact* and *release* to represent all the other activities which are required in real situations. Figure 3.5 illustrates how some new primitive operations, such as move, pick-up, and dispose, are concatenated in order to represent the tasks *contact* and *release*. The emptying process requires other compositions of the *contact* and the *release* tasks, as compared to those shown in Figure 3.5. The robot model, however, assumes an equal execution time for both the *contact* or the *release* tasks irrespective of their

internal compositions. We refer to a filling or emptying process, which embodies these two tasks, as a filling or emptying task, respectively. Note that in Figure 3.5, the operation of 'emptying a syringe' is equivalent to the operation of 'filling a chamber'.

Operations Required to Fill a Chamber

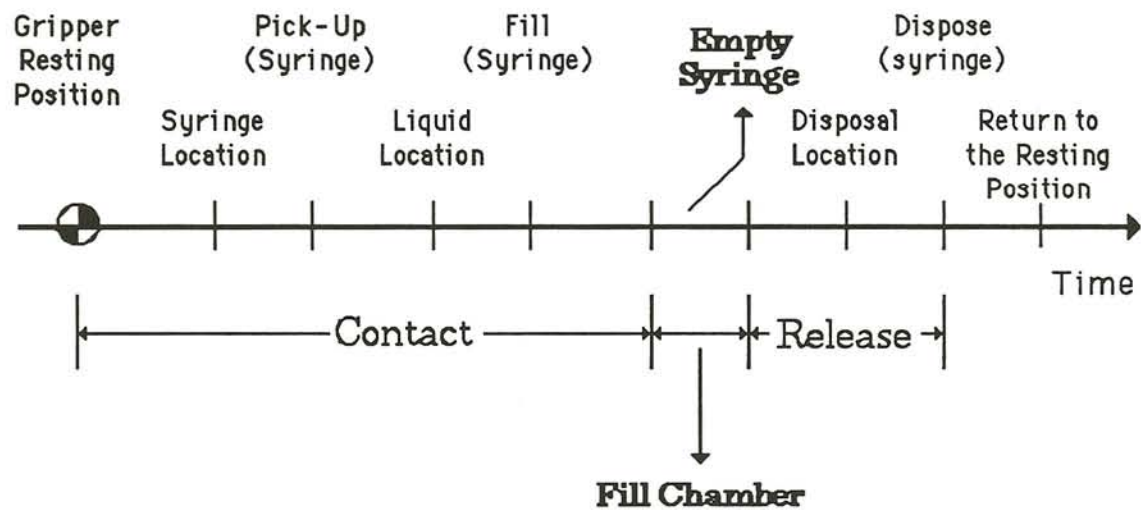


Figure 3.5: Decomposition of *release* and *contact* tasks into more specialized tasks with respect to filling operation.

Table 3.2: Selected state variables chosen to represent the Event-Based controller.

| State Variables | Description of the Robot's State Variables | Default Values |
|------------------------|---|-----------------------|
| REF-LE: | Number of refills for LE | 0 |
| REF-TE: | Number of refills for TE | 0 |
| REF-SA: | Number of reruns | 0 |
| STA-LE: | Status of the LE chamber: Dirty, Empty, Clean, Full, Error-Empty, Error-Clean, Error-Fill, LE-Full (no air bubbles) | Dirty |
| STA-TE: | Status of the TE chamber: Dirty, Empty, Clean, Full, Error-Empty, Error-Clean, Error-Fill, TE-Full (no air bubbles) | Dirty |
| STA-SA: | Status of the segment of the capillary which is allocated for for the sample solution: Full, Empty, Error-Fill | Empty |
| F-TIME-LE: | Minimum and allowable additional time to fill LE chamber | (19 , 6) |
| C-TIME-LE: | Minimum and allowable additional time to clean LE chamber | (28 , 7) |
| E-TIME-LE: | Minimum and allowable additional time to empty LE chamber | (22 , 8) |
| F-TIME-TE: | Minimum and allowable additional time to fill TE chamber | (09 , 3) |
| C-TIME-TE: | Minimum and allowable additional time to clean TE chamber | (18 , 5) |
| E-TIME-TE: | Minimum and allowable additional time to empty TE chamber | (12 , 4) |
| F-TIME-SA: | Minimum and allowable additional time to fill SA chamber | (03 , 2) |
| SIGMA: | Defined in section 2.5. | ∞ |
| PHASE: | Allowable phases. | PASSIVE |
| OPRT: | Allowable input ports. | NIL |
| OVAL: | Allowable input values. | NIL |

Where: LE: leading electrolyte
TE: terminating electrolyte
SA: sample

As discussed in Section 3.1, the event-based controller must operate on the ITP device or equivalently the (*external*) model, M-ITP-OPER, which exhibits the proper dynamical behaviors. Recalling our previous discussion of the event-based controller in Section 2.4, we must construct the ITP's model such that it is able to represent t_{min} , t_{max} , and t_{wind} (cf. Section 2.4). The required characteristics can be represented by a *control* model of the ITP device.

| Robot's State Variables | |
|-------------------------|-----------|
| Static | Dynamic |
| OPRT | OPRT |
| OVAL | OVAL |
| PHASE | PHASE |
| SIGMA | SIGMA |
| REF-LE | F-TIME-LE |
| REF-TE | C-TIME-LE |
| REF-SA | E-TIME-LE |
| STA-LE | F-TIME-TE |
| STA-TE | C-TIME-TE |
| STA-SA | E-TIME-TE |
| | F-TIME-SA |

Figure 3.6: Categorization of the robot's state variables into static and dynamic.

The atomic-model of the robot as depicted in Appendix A, represents the decision making capabilities which require both the *internal* model, M-ITP-CONT, and the *external* model, M-ITP-OPER, of the ITP device (see Appendix A). The robot's state variable descriptions and their default values are given in Table 3.2. The state-variables, as categorized in Figure 3.6, represent the *static* and *dynamical* decision making capabilities. The state variables which are devoted to the dynamical behavior of the ITP device represent the *internal* M-ITP-CONT model.

A scenario of the filling operation of the LE chamber, assuming it is clean, illustrates how the robot performs as an *eventistic* controller (Meystel and Luh, 1987). Figure 3.3 illustrates how the robot model (M-ITP-CONT) and the ITP model (M-ITP-OPER) simulate the filling process concurrently. Let us presume that the robot has already completed its *contact* task. The robot, then, schedules t_{min} (i.e., *phase* = le-fill-low) for the filling operation of LE by using the M-ITP-CONT³ and, at the same time, sends a message, (*LE, FILL*), to the lower level controller to carry out the filling operation. Note that we only represent the model of the operation by using the M-ITP-OPER and not its lower level controller. Thus, the filling operation of the LE chamber is scheduled (i.e., *phase* = le-fill) to take 22.9 units of time as described in Section 3.2.3. It is possible that the robot receives a message (i.e., notifying the end of the operation) before the completion of $t_{min} = 19$, forcing it to suspend (i.e., *phase* = error-fill) the continuation of the future activities (i.e., transmitting a halt message to the ITP's lower level controller if one exists), and to issue another message,

³ This model is integrated into the model of the robot.

(robot , error-le-fill-low) \equiv too-early, to a diagnostic unit. Again, since we have excluded the lower level controller, the 'halt' message which must go to the lower level controller is absent.

If the robot does not receive the completion message regarding the filling operation too early, it schedules (*phase* = le-fill-twind) the M-ITP-CONT to continue for $t_{wind} = 6$. If a message, (*LEITP, FILL*), is received within the time period t_{wind} confirming the completion of the filling operation for the LE chamber, then the robot proceeds by issuing its next scheduled command (i.e., a *release* operation followed by the examination of the LE chamber for air bubbles). The remaining possibility is that the t_{wind} elapses before a confirmation is received from the M-ITP-OPER (i.e., *phase* = error-fill). Again, the robot must interrupt the current operation of the ITP device (i.e., filling the LE chamber) by transmitting a stop message to the lower level controller in addition to routing a message, (robot , error-le-fill-late) \equiv too-late, to a diagnostic unit. The robot utilizes the above *eventistic* control algorithm for steps 1, 3, and 5 of list **A** as well as for the emptying operations and cleaning tasks of both the LE and TE chambers which belong to steps 4, 7, and 10 of list **B**.

Let us now entertain the second type of control scheme which must be embodied in the robot's model as well. This type of control is essentially *static* in nature since it acts on a class of information that does not depend on any dynamics. For instance, in Appendix A, it is shown that if the filling operation of either chamber is repeated more than 20 times, the event-based controller should discontinue to issue any filling command to its assigned lower level controller. This is a valid type of control strategy

that should be built into the real system in order to *recognize* and *terminate* endless processes.

Another type of control which also belongs to this category disallows a future activity to take place if its correct execution depends on a previously executed operation of which we already know that it was not completed in an acceptable (i.e., 'perfect') manner. Therefore, a known sequence of tasks must be altered such that processes that can invalidate the outcome of a future operation are tested for 'perfection'; recall the notion of 'successful' and 'perfect' operations discussed earlier.

As an example, the camera which is responsible to detect the presence of air bubbles in the chambers and the capillary, provides the desired information that is used by the robot to either continue with its previously scheduled sequence of activities (i.e., list **A**), or change the sequence (i.e., list **B**) to warrant the perfection of a desired operations.

3.3 An Overall View of the FHL

Up to the present, we have not mentioned the interaction between the robot and the rack. As stated earlier, the rack is a static model that is included to furnish knowledge for the *static* control scheme described above. The robot may inquire, at the beginning of an experiment set-up, information to ascertain that there exists an adequate amount of supplies. We have not included this type of control which depends on inquiries from the rack; instead, the FHL's environment is constituted such that the robot

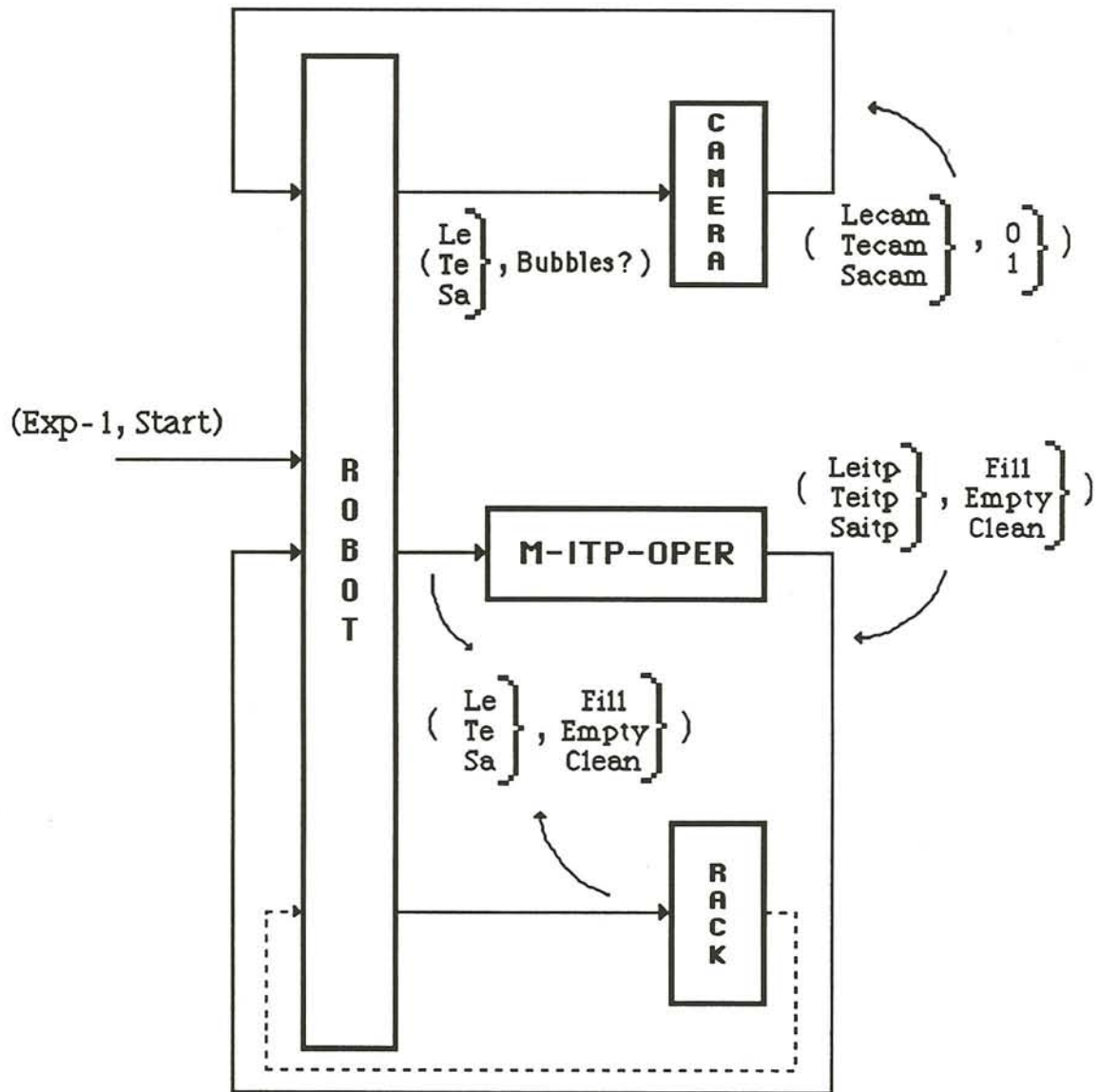


Figure 3.7: Block diagram representation of the FHL.

sends appropriate messages to the rack in order to update its list of possessions. The interactions between the robot and the ITP device provided the event-based control approach; whereas the camera and the rack were necessary for static control of the experiment. Figure 3.7 illustrates the interconnections among the entities. Figure 3.7 exemplifies the FHL's environment. Note that we have represented the model of certain processes only, and we have eliminated their lower level controllers (cf. Figures 3.4 and 3.7). Examination of Figures 3.4 and 3.7 manifests that the control commands originating from the event-based controller are routed to the processes (i.e., the ITP device) instead of a lower level controller, assuming that the robot is acting as an intelligent controller. Indeed, it is quite practical to implement both the lower level control algorithms (i.e., the actuator commands) and the higher level control algorithms (i.e., the procedural commands) inside the robot model.

Although, in delineating the FHL environment, we bypassed a PID type (low level) controller, as apparent from Figure 3.7, we have not jeopardized our goal which is to illustrate the applicability of the DEVS formalism to represent an event-based controller.

CHAPTER 4

DIAGNOSTIC AGENTS

4.0 Outline of the Chapter

The focus of this chapter is the consideration of failure analysis. In Section 4.1, the utilization of *conventional* schemes and *artificial intelligence* schemes for the development of a diagnostic agent is discussed. Diagnostic units will only be considered for the analysis of hardware failures. The consideration of *hierarchical* diagnostic units, expected to be utilized in parallel with a *hierarchy* of controllers, is discussed. Furthermore, the applicability of *expert systems* for higher levels of hardware failure analysis is considered. Section 4.2 describes an interactive expert system shell. The usefulness of this expert system shell for the development of a higher level diagnostic unit in the PC-Scheme environment is discussed. The selection of the *diagnostic* model of the ITP device, M-ITP-DIAG, for a diagnostic agent is treated in Section 4.3. Next, in Section 4.4, a customized diagnostic agent, constructed from CESM, is presented. It obtains directly the necessary information from the desired modeled sensory sources. This chapter concludes with Section 4.5 where the effects of certain functions, such as the cost and time requirements associated with the data acquisition, are discussed as they influence the development of diagnostic units.

4.1 The Diagnostic Process

Diagnosis: “*the act or process of deciding the nature of a diseased condition by examination of the symptoms*” (Webster’s Dictionary, 1984).

From the above definition, diagnosis is defined as the recognition of cause(s) of a disease or a fault when it is observed in the context of human made systems.

Generally, faults can be categorized into two types: *hard*, related to actual hardware failures; *soft*, related to those faults which are not categorized as *hard*. For instance, a fault related to a sensor might be due to hardware failures such as mechanical stress, over-voltage, and over-heating. On the other hand, the same sensor may fail due to the lack of some required information or electrical signals.

Here, we shall only consider errors which are categorized as *hard* in the pursuit of the selection and implementation of a prototype diagnostic agent for the FHL. By discarding errors of type *soft* in our study, the main theme of this thesis will be served without any loss of generality.

There exist three general schemes for diagnosing a fault, depending on the technology used. These schemes are referred to as *manual*, *conventional*, and *artificial intelligence* schemes. Obviously, the manual scheme is not suitable for our purpose. In the following, we shall examine the remaining schemes in order to select one which is able to serve our requirements —a diagnostic unit must be readily implementable in

relation to the previously developed event-based controller, and be capable of diagnosing higher level causes of failures within its defined domain.

4.1.1 Conventional Scheme

The conventional scheme is generally based on continuous and/or discrete-time mathematical models which are usually represented by (ordinary, partial, linear, non-linear, deterministic, stochastic) differential or difference equations. One high order equation can be used or it can be reduced to a set of first order equations, a so-called state-space model. A number of approaches which are based upon such mathematical models are: (1) "Parameter Estimation in Real Time" (Dalle-Molle and Himmelblau, 1987), (2) "Parity Space Eigenstructure Assignment" (Patton and Willcox, 1987), (3) "Hierarchical and Fast Recursive State Estimation" (Mansour and Nour-Eldin, 1987), (4) "Direct Detection" (Hassan *et al.*, 1987), (5) "Local Second-Order Observers" (Hengy and Frank, 1987), (6) "Overlapping Decomposition" (Tzafestas and Skolarikos, 1987).

Generally, all conventional methods are based on either *hardware redundancy* or *analytical redundancy*. The hardware redundancy approach uses multiple identical measuring devices (sensors, etc.) such that appropriate majority logic is able to detect the faults. Obviously, the presence of back-up measuring devices poses major difficulties in actual implementation of certain systems due to concerns related to increased cost, weight, volume, and/or complexity.

The analytical redundancy approach, on the other hand, is free of the above problems since a mathematical model representation is chosen in order to reflect inherent relationships among the measured variables of the systems. This approach has been the focus of recent research due to new developments in control theory, estimation theory, and the increased computing power of microcomputers and microprocessors. It eliminates the need for redundant hardware in the system.

This approach, however, is clearly inappropriate for a large number of systems such as nuclear power plants, aircrafts, satellites, and remotely operated systems which *must* contain hardware redundancies for safety reasons. Usually, failures of such systems have so severe consequences that cost and weight factors are considered unimportant in comparison.

A diagnostic agent which does not depend on hardware redundancies is called MBDS (Model Based Diagnostic System) (Ribbens, 1988). The design of a MBDS is based on the conventional approach, i.e., a dynamical model of a system with known relationships among its variables should be developed. Such a diagnostic unit has been designed and implemented for electronically equipped engines (Ribbens, 1988). It detects failures in actuators (complete failures), and in the calibration shift of sensors (partial failures). A linearized state-space model of the engine is used to represent its behavior in the neighborhood of a given operating condition.

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{u} \\ \mathbf{y} &= \mathbf{C} \cdot \mathbf{x} + \mathbf{D} \cdot \mathbf{u} \end{aligned}$$

Although the model of the engine is non-linear, a linearized model is able to represent the dynamics of the system as long as the matrices A and B are updated in order to cover the full dynamic range of the engine. Figure 4.1 depicts the block diagram of the MBDS:

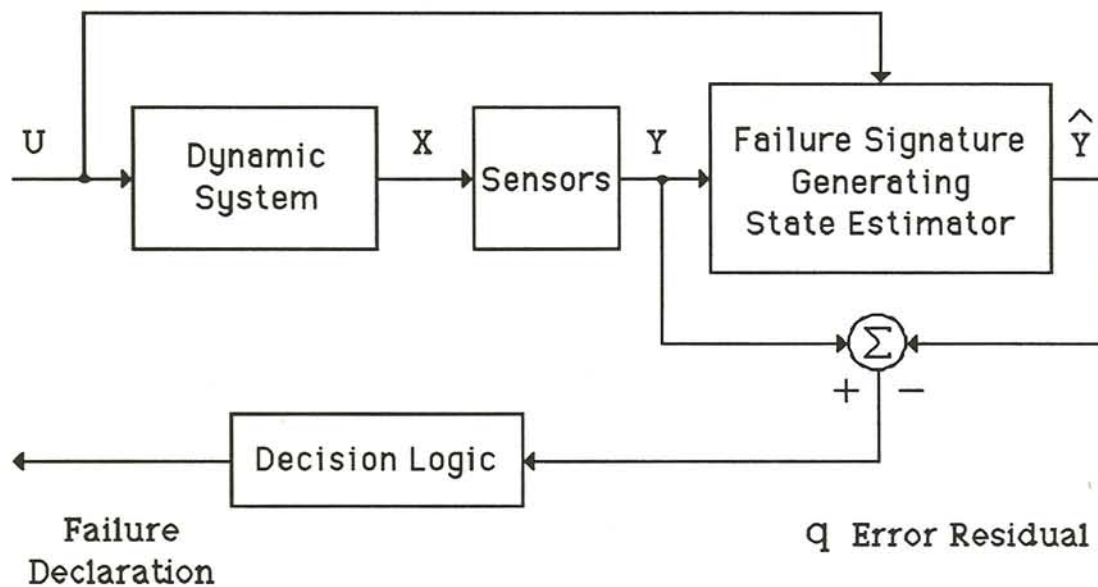


Figure 4.1: Block diagram for MBDS.

The "failure signature" generation system is a form of state estimator. The diagnostic agent compares the measured dynamic system performance with the estimated performance. If there is no failure in any component, the measured and the predicted performance of the modeled system will agree. Otherwise, the generated error residual q is analyzed for the determination of the failure(s).

4.1.2 Artificial Intelligence Scheme

Diagnostic systems which are based on A.I. are often referred to as either *Knowledge-Based Systems* or *Expert Systems*. An expert system (ES) devoted to the diagnosis process consists of a *Knowledge Base* (KB) and an *Inference Engine* (IE). The KB contains the factual knowledge and the preferential knowledge, while the IE contains the general problem-solving techniques applied to the KB. Several supporting components such as a user interface and knowledge acquisition modules are often specified to facilitate their usage. Each of these basic components of an expert system (i.e., the knowledge base and the inference engine) is further divided into several types (Rich, 1983).

For the representation of the knowledge-base, a critical issue in developing a knowledge-based system, there exist several different methodologies such as frames, production rules, lists of facts, semantic networks, and logic predicates. It should be emphasized that usually not any one knowledge representation is sufficient, and a combination of the above schemes is therefore common. The second component, the inference engine, also offers various options such as forward-chaining, backward-chaining, best-first search, and constraint satisfaction. As with the knowledge representation, a combination of various inference mechanisms is also common.

Three approaches in developing a diagnostic system using the techniques of artificial intelligence have been reported (David & Krivine, 1987):

1. *Declarative Programming:*

When there exists an algorithm that uses diagnostic knowledge represented in a declarative manner, the underlying technique is classified as *declarative programming*. This approach is appropriate when there is an explicit way to formulate the diagnostic algorithm, and if conventional programming techniques are inadequate (i.e., the diagnostic knowledge is continuously evolving, and/or the problem solving process involves symbolic manipulations, etc.).

The knowledge of the expert is assumed to be exact but not necessarily complete. The knowledge is not directly programmed out, but instead, it is expressed in a declarative manner: i.e., associations between failures and their manifestations are expressed using a production rule formalism. *Fault-trees* and *fault-dictionaries* are two forms of declarative knowledge representations. Such a data driven knowledge representation scheme thereby allows easy modification (i.e., adding/deleting) of the rules in comparison with knowledge encoded directly, i.e., using conventional program structures.

2. *Expert Systems:*

Expert systems are used when no exact explicit diagnostic algorithm is available, but there exists a *heuristics* on the basis of which occurring problems can be diagnosed. Specifically, the knowledge employed by expert systems is heuristic (i.e., uncertain, inexact, and incomplete). Expert systems are organized in a hierarchy of 'prototypes' where

each prototype is a description of a typical problem (Aikins, 1983). Designers of expert systems make use of the *hypothesis/establish/refine* strategy, thus they try to establish ever more specialized prototypes. The control of the information acquisition, the establishment/ rejection of failures, and the control of the search for the next prototypes are all decisions which are heuristic in nature.

The designer of an ES usually faces several difficulties relating to the formalization of the problem, the selection of useful knowledge, knowledge representation, knowledge utilization, knowledge acquisition, and finally, validation of the expert system. Expert systems exhibit several inherent drawbacks: they do not have a real understanding of what they are reasoning about, they are domain specific, their explanation capabilities are solely based on a trace of fired rules, and finally, they are unable to justify their knowledge.

The main difference between the declarative programming and the expert system approach lies in the nature of the knowledge, and how it is encoded and stored. That is, expert systems always instantiate declarative programming since its knowledge is continuously evolving. Expert systems can be categorized into the following A.I. formalisms:

- i) *Production Systems*: These have a single KB and a single IE.
- ii) *Structured Production Systems*: These have a single inference engine (i.e., a single set of meta-rules) operating on several distinct KBs.
- iii) *Distributed Systems*: These use a hierarchically structured network of co-operating specialists.

3. *Model-Based Reasoning:*

If some of the data upon which the decision making process relies (either using declarative programming or the expert system approach) is extracted from one or several models rather than from the real system, the system is referred to as a *model-based system*. It has been demonstrated that model-based systems can help overcome difficulties expert systems (e.g., the unavailability of explicit knowledge, the incompleteness of knowledge necessary to cover a specific domain, or the large amount of knowledge required to represent a complex system) encountered in the utilization of (Davis, 1982, 1984), (Chandrasekaran and Mittal, 1983), (Forbus, 1985). In the sequel, A.I. techniques may use knowledge inherent in previously developed models of a system *in lieu* of, and in addition, to real system knowledge. For example simulation experiments may be performed on models to generate approximate information about the expected behavior of the real system. The power of this approach lies in its ability to reason with some understanding of the system it is reasoning about. Also, the knowledge inherent in these models is stored in a much more compact form than would be the case if all possible modes of system behavior were explicitly enumerated.

These models are commonly referred to as *deep* models or *causal* models. Among those model types are the *functional models* (deKleer & Brown, 1984), the *Causal Network* (Riger & Grinberg, 1977), the *Qualitative Physics* models (Forbus, 1985), (Kuipers, 1986), and the *Black-Box Network* (Davis, 1984), (Genesereth, 1984). These methods *reason* on a model of the behavior or the structure, and relations among domain objects. For diagnostic purposes, the reasoning on these models consists of extracting associations between actions, faults, and

symptoms. The associations between symptoms and fault hypotheses result in a so-called *candidate generation*. *Test generation* is the outcome of associations among actions, fault hypotheses, and observations.

The main difficulty with this approach is the inability to find an appropriate model which is powerful enough to cover all important faults. It should be emphasized that the model must remain tractable. For this reason, there often exist several different models that reflect the system behavior under different fault assumptions (Davis, 1982, 1984).

It should be evident that the latter approach is used for classes of problems that are not successfully solvable by either of the previously discussed approaches.

4.2 Hierarchical Diagnostic Agents

As stated in Chapter 1, one of our objectives is to demonstrate that it is advantageous to construct a diagnostic agent with an appropriate level of complexity and power. That is, a diagnostic unit must be developed such that it is consistent with its associated intelligent controller. Both the conventional and the artificial intelligence approaches provide us with numerous choices to construct such a diagnostic unit.

To correspond to the hierarchy of control, we envision a hierarchy of diagnostic units. One form of a hierarchical diagnostic unit, which is based on a paradigm of 'cooperating diagnostic specialists', has been suggested and implemented by (Bylander *et al.*, 1985). In this thesis, however, the hierarchy of diagnostic units is intended to include not only

expert systems, which may themselves be constructed on a hierarchical basis, but also model-based programming techniques. Expert systems which combine heuristic reasoning based on rules with deep reasoning based on a model of a problem domain are referred to as *Second Generation Expert Systems* (Steels, 1986).

In this work, the hierarchy of diagnostic units is limited to two levels only. The justification for such a selection is as follows: earlier we decided to distinguish only between the high-level (event-based) and the low-level (conventional) types of control. Thus, in order to comply with our objective —the consistency between the various levels of controllers and their corresponding diagnostic agents—, we shall consider corresponding *low-level* and *high-level* diagnosers. Indeed, if the low-level diagnoser is built on the basis of the conventional scheme discussed earlier, and the high-level diagnoser is constructed on the basis of the artificial intelligence scheme, then there exists a one-to-one mapping between the high/low-level controller and its diagnostic unit. This means that for every *intelligent controller* there exists an *intelligent diagnoser*, and for every *conventional controller* there exists a *conventional diagnoser*. We have used the words *intelligent* and *conventional* in order to convey the same idea which was utilized in the design of controllers. Additional discussion will be presented in due course.

It is not always necessary to utilize the conventional scheme to develop a low-level diagnostic unit. The model-based approach can be used for the development of a low-level diagnostic unit as well. Other approaches within the A.I. framework are also suitable for the construction of a low-

level diagnostic unit if the underlying characteristics of the system permit. However, the converse —the applicability of a conventional approach for a high-level diagnostic unit—, is not as easily accomplished due to the necessary higher level features (e.g., nonnumeric commands).

Considering our objective, we shall eliminate the consideration of a conventional scheme for the implementation of a high-level diagnoser because of the following two reasons: first, the type of diagnostic capabilities which are desirable to us are targeted toward a higher level which is compatible with its intelligent controller; and secondly, the DEVS formalism and its corresponding environment, DEVS-Scheme, do not currently support the manipulation of continuous or discrete-time systems unless they are mapped into discrete-event models.

Although we have eliminated the consideration of the conventional scheme for the development of an intelligent diagnostic unit which is required to operate at the same level as the event-based controller, we still have many methods available to us within the A.I. framework. In order to design an intelligent (high-level) diagnoser and a conventional (low-level) diagnoser, the analyzed system should be partitioned in accordance with the approach taken for the event-based controller. For instance, the diagnostic agent that is assigned to reason on failures of a computer may be considered as either high-level or low-level depending on whether the computer itself is considered to be a system composed of several subsystems or an atomic component within a multi-processor architecture, respectively.

If the diagnostic agent assigned to this computer is considered to be high-level, then it has to locate high-level failures such as failures of some electronic boards, the power supply, and so on. For example, this diagnostic unit will not be responsible to locate the low-level failures that are inside an electronic board. Such low-level failures could be due to faulty modules, wrong wiring, wrong modules, disconnected wires, and many others —requiring a low-level diagnostic unit. On the other hand, if the computer is considered to be a component of a larger system, then the high-level diagnoser is responsible to locate the faulty component which could be the computer or any other part of the system which is also considered to be a component (e.g., peripherals), whereas the low-level diagnoser would be responsible to diagnose causes of faults within the computer.

The number of levels which are assigned to each diagnostic unit may vary depending on the partitioning of failures and the number of diagnostic units. That is, if the failures are classified into five levels and a low-level diagnoser and a high-level diagnoser are considered, one diagnoser may be responsible for two levels of failures, while the other is responsible for the remaining three levels.

The *high-level* and *low-level* knowledge representations, which are utilized by their corresponding diagnostic units, are examined to guide us in selecting an appropriate algorithm within the three A.I. approaches for a high-level diagnostic unit. Knowledge bases are used to store various kinds of information about the system and its components. A KB may be referred to as a model of the system as well. Generally, the type of knowledge enables us to conduct either *shallow-reasoning* when the

knowledge is classified as high-level or *deep-reasoning* when the underlying knowledge is classified as low-level (Chandrasekaran & Mittal, 1983), (Scherer & White, 1987).

The knowledge bases which contain the high-level associations of various components of a system are constructed from general behavioral and the structural knowledge of the system. This form of knowledge representation is adequate when we do not have the capability to represent the system more faithfully (e.g., a system such as the human body is not completely understood), or when the system is considered to be very complex.

Furthermore, we may also choose to represent a system, the low-level behavior and structure of which are known, by the shallow-reasoning paradigm because of its relative simplicity, the type of analysis, and the insight that it is capable of delivering. This form of reasoning is primarily based on *rules-of-thumb* that are often acquired from experience. Nonetheless, it may become difficult and too complex to define all interactions at the system level and determine what part of this knowledge is useful and how it should be used.

The other approach, deep reasoning, reasons on structural and behavioral information (Davis, 1982, 1984)¹. The system is described as a hierarchical network of black boxes. The structural information describes the interconnection among the modules which can be organized in two ways: functional and physical. The functional view considers the system's

¹ There exist various definitions of *deep-reasoning* with minor differences among them. We chose the definition given here due to its wide-spread usage and acceptance.

organization according to how its modules interact, while the physical view specifies how it is physically constructed. The behavior determines how the information leaving a component is related to the information entering it. This approach considers issues such as multiple descriptions of a structure, differences between structure and behavior, enumeration and layering of failures, methodical enumeration and relaxation of underlying assumptions, and systematic generation of categories of failures by examining the underlying assumptions.

The concept of layering of failures can be advantageously used in deciding what approach (i.e., conventional or A.I.) is pertinent for a high-level diagnostic unit, assuming a hierarchy of diagnosers. Figure 4.2 shows that either a single diagnoser (i.e., Figure 4.2a) or two diagnosers (i.e., Figure 4.2b) may be constructed. That is, we have only considered two classes of failures: high-level (i.e., system level) and low-level (i.e., component level) without indicating how many internal levels each diagnoser may be responsible for.

It may be necessary to consider more than two levels of failures, just as it may be necessary to have several levels of controllers. Once a high-level diagnoser determines a faulty device, it notifies a selector which in turn decides what kind of diagnoser (i.e., a more specialized diagnoser compared to the previous one) should be utilized for further diagnosis. The hierarchy of failures is similar to our earlier categorization of controller commands into high-level commands and low-level commands (cf. Section 3.2.4). This interpretation serves our purpose well as we do not wish to perform diagnoses at primitive levels (conventional or model-based

diagnoser) but rather at comparatively higher levels (intelligent diagnoser). It should be emphasized that the distinction between these two categorizations of failures is application dependent.

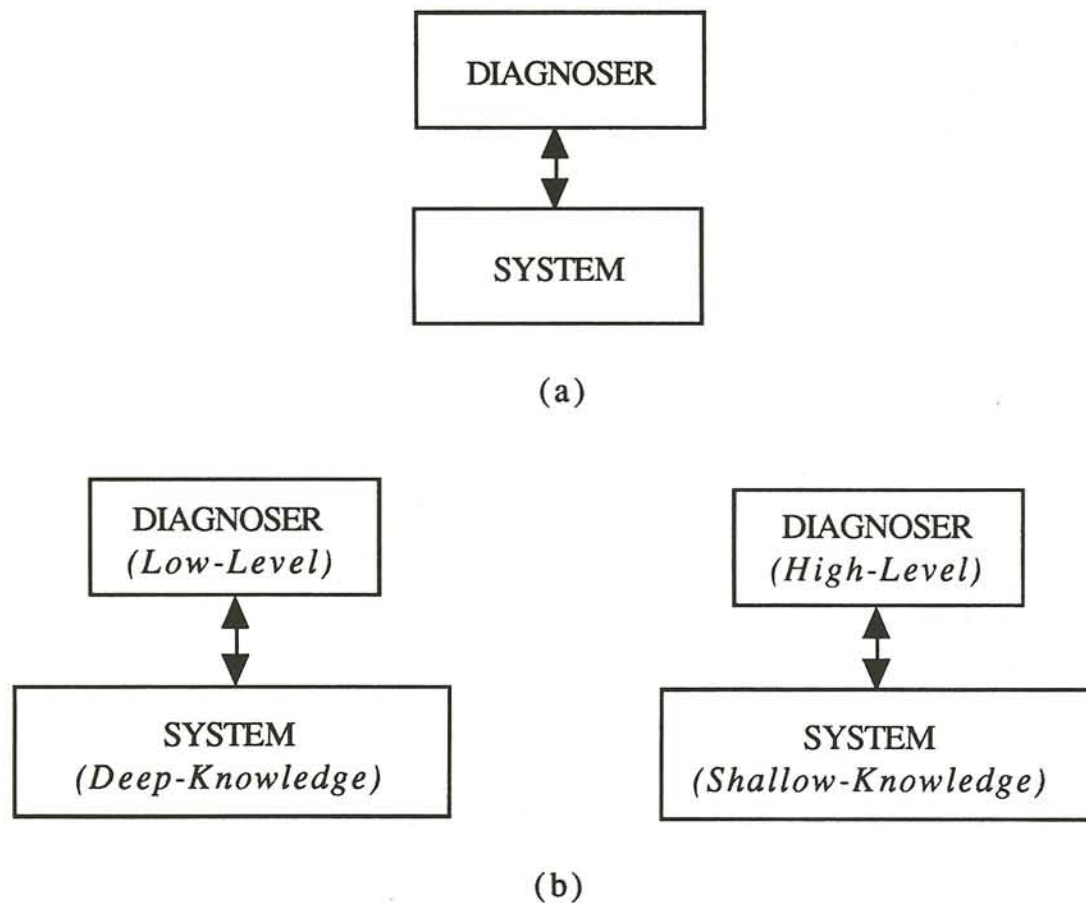


Figure 4.2: Hierarchy of diagnostic units. (a) Single diagnoser; (b) high-level and low-level diagnosers.

Model-based reasoning which is well suited for locating low-level faults is based on deep-reasoning. Note that the conventional approaches are capable of generating approximately the same types of failure analysis as the model-based approach. The approaches which are based on deep-reasoning, therefore, are not considered to be appropriate for higher levels since the diagnostic capabilities afforded by deep-reasoning is extensive and provides more detail than what is useful at the higher levels. It is also more difficult to develop these types of diagnostic units due to difficulties related to the knowledge representation and the control strategies. An intelligent diagnostic unit which should be operating at the same level of complexity as its intelligent controller does not require such detailed information. An appropriate expert system is able to provide the medium for shallow-reasoning since the underlying knowledge can be made available at the higher level. Thus, an expert system is considered to be a good candidate for the development of an intelligent diagnostic unit.

We must also consider the remaining approach, declarative programming, for possible implementation of an intelligent diagnostic unit. This approach is not desirable since it is usually difficult to formulate an explicit diagnostic algorithm. This is specially important since the operating environment is exposed to both *system upgrading* and *system degradation*. The instruments used in the GPL will be frequently upgraded, while some components of the SSF may fail, and cannot immediately be repaired or replaced. The diagnoser must be able to operate correctly and reliably under such varying and not completely foreseeable conditions. Furthermore, the exact nature of the operations required to

conduct experiments in the microgravity environment is not completely known, and the possibility of new or modified experiments and thereby the prevalence of modified or new operating conditions must be considered.

From the discussion of the available approaches within the A.I. framework, expert systems are the most suitable for the development of a high-level diagnostic unit.

Nonetheless, we must still select one formalism among the possible three (i.e., Production Systems, Structured Production Systems, and Distributed Systems). Obviously, the choice of one of these formalisms depends on its availability and appropriateness with respect to the problem at hand. Consequently, we postpone the selection process until the presentation of an expert system which is available to us.

Given the foregoing explanation of failure levels and the diagnostic systems representing them, it is appropriate to formalize the general diagnosis definition, that was given at the beginning of this chapter, by concisely defining the set of its steps as follows:

1. Error Recognition:

The diagnosis process begins with the observation of misbehavior which is recognized as a discrepancy (i.e., commonly referred to as a symptom) between the expected and the observed behavior. This is generally assigned to a controller such as an event-based controller or a conventional controller. The event-based controller is able to generate error messages carrying non-numeric information such as "too-late" or "too-early" (cf. Chapter 3).

2. Candidate Generation:

Once an error has been recognized, the problem solver attempts to generate one or several hypotheses about the possible causes of the malfunction. The possible discrepancies are often expressed in several different forms using appropriate knowledge representations. One class of knowledge representation can express the known associations between symptoms and causes, that are usually referred to as the diagnostic knowledge. Other forms of knowledge representations are fault-dictionary, fault-tree, and directed graphs (Rich, 1983). The same associations can also be represented as a set of rules which may be based on an expert's knowledge. The expert system approach usually utilizes diagnostic knowledge which is heuristic in nature. Another form of knowledge representation which usually pertains to model-based reasoning can generate the necessary knowledge from a model or a schema as explained earlier (Davis, 1984).

3. Candidate Reduction:

The diagnostic process continues with the confirmation/rejection of cause(s) for a determined fault. Indeed, it is rare that a diagnoser would indicate only one possible cause for an observed fault unless there exists exact knowledge to determine the cause unambiguously. In the domain of expert systems, the knowledge is often inexact, uncertain, and incomplete (i.e., heuristic). The non-uniqueness of faults also holds true for knowledge acquired from the model-based reasoning approach; it is difficult to acquire/maintain all the necessary information for positive identification of the causes of an error. Hence, a diagnoser often produces a number of possible candidates for a given fault. Then, it is expected to

select only one of the possible causes based on some criteria of conflict resolution. Once the possible source of a failure has been determined, an attempt to correct it by appropriate means is conducted.

The complexity of each of the above steps is dependent on the specific application. In particular, Step 2 deals with various knowledge representations such as lists of facts, semantic networks, production rules, logic predicates, and frames. The last step which is responsible for the inferencing includes methods such as forward-chaining, backward-chaining, and best-first (Rich, 1983).

In the next section, we will describe an expert system shell which is being used to implement a diagnostic unit for the FHL and in particular for the electrophoresis experiment. Moreover, we shall examine the knowledge representation scheme and the control strategies which are offered by this expert system.

4.3 Classification Expert System Maker

The Classification Expert System Maker (CESM) is an expert system shell which is designed for developing *classification-based* expert systems (Zeigler, 1987-b). This expert system shell is written in PC-Scheme which supports the *object oriented programming paradigm*. CESM is of particular interest to us since DEVS-Scheme has also been developed in PC-Scheme (Texas Instruments, 1985). Consequently, interfacing between

CESM and DEVS-Scheme is straightforward. Having the ability to utilize an expert system (i.e., a diagnostic unit) in an environment which also supports the modeling and simulation of discrete-event systems plays a significant role in selecting CESM as an expert system shell.

This expert system shell provides an expert system building environment similar to most other expert system shells. The structure of CESM as shown in Figure 4.2 supports several important features:

- system classification models built on the taxonomy entity structure,
- automatic compilation of a knowledge base into rules,
- a straightforward interfacing with the underlying environment (i.e., PC-Scheme),
- a convenient way to add/delete rules (i.e., knowledge) to/from the knowledge base².
- expressions and function descriptions in addition to synonyms and anti-synonyms, and
- uncertainty handling using qualitative words (i.e., *usually*, and *rarely*), and evidence accumulation utilizing the Dempster-Shafer approach.

² It should be noted that this feature requires knowledge that is well defined with respect to its domain. It is not always possible to add rules to a rule base without influencing the existing rules, thus introducing inconsistencies.

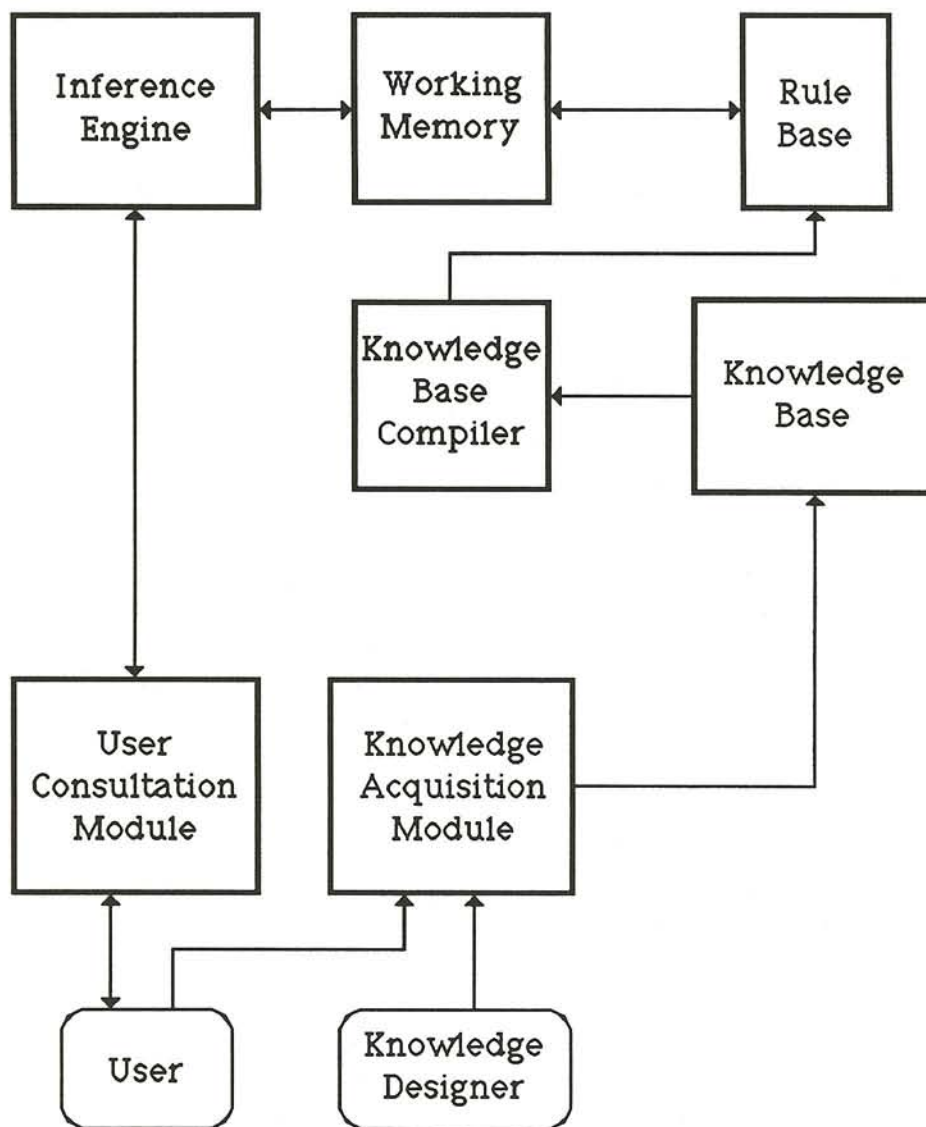


Figure 4.3: Architecture of CESM —components and their interactions.

The data types and data values for CESM are as follows: *class* (i.e., entity) and *predicate* are two primary types of data where the class plays the role of an unknown class in the inference engine, and a predicate is used to describe a property of a class.

The value of a class is a measure of the evidence relevant to it. CESM represents the values of each entity by using a quadruple as defined below:

$$(EF, EA, N, X)$$

where:

$$0.0 \leq EF, EA, N, X \leq 1.0$$

$$EF + EA + N + X = 1.0.$$

Some of the evidential status values which are displayed to the users are as follows:

$(1.0, 0.0, 0.0, 0.0) \Rightarrow$ CEF : Conclusive Evidence For

$(0.9 \leq EF < 1.0, EA, N, X \leq 0.1) \Rightarrow$ EF : Strong Evidence For

$(0.5 \leq EF < 0.9, EA, N, X \leq 0.1) \Rightarrow$ EF : Unchallenged Evidence For

$(0.0 < EF < 0.5, EA, N, X \leq 0.1) \Rightarrow$ EF : Weak Evidence For

$(1.0, 0.0, 0.0, 0.0) \Rightarrow$ CEA : Conclusive Evidence Against

$(EF, 0.9 \leq EA < 1.0, N, X \leq 0.1) \Rightarrow$ EA : Strong Evidence Against

$(EF, 0.5 \leq EA < 0.9, N, X \leq 0.1) \Rightarrow$ EA : Unchallenged Evidence Against

$(EF, 0.0 < EA < 0.5, N, X \leq 0.1) \Rightarrow$ EA : Weak Evidence Against

$(1.0, 0.0, 1.0, 0.0) \Rightarrow$ X : Neutral

$(0.0, 0.0, 0.0, 1.0) \Rightarrow$ X : Inconsistent

If three of the four elements of a quadruple are known, the remaining element can be calculated from the equation $EF + EA + N + X = 1.0$. Thus, it is sufficient to store the first three elements of the quadruple (EF EA N X) only.

Permissible values for a predicate are 'yes', 'maybe-yes', 'no', 'maybe-no', and 'unknown'. Since the predicate and the class have different data values (e.g., *maybe-yes* for the predicate and quadruple for the class), the inference engine makes them compatible by equating them as follows:

yes \equiv (1.0 0.0 0.0)

maybe-yes \equiv (0.5 0.0 0.5)

no \equiv (0.0 1.0 0.0)

maybe-no \equiv (0.0 0.5 0.5)

unknown \equiv (0.0 0.0 1.0)

In the remaining of this section, we will describe briefly the individual components of CESM followed by a discussion of the knowledge representation and the inferencing mechanisms of CESM.

Knowledge Acquisition Module: This is a knowledge base development program which provides a user-friendly interface to define knowledge for a particular domain. The knowledge which is required from a user must follow the structure of an entity structure. The acquired knowledge, which is systematically stored in a file, is saved in the *knowledge base*.

The Knowledge Base Description Language (KBDL) in CESM is simple and easy as it allows any English word, abbreviation, or phrase to be used for a class or a predicate. As mentioned earlier, qualitative words *usually* and *rarely* are available to qualitatively indicate the degree of certainty of any predicate or equivalent rules which are constructed from them.

Knowledge Base Compiler: The knowledge stored in the knowledge base is translated into a set of rules by this component. Then, a file which contains the compiled rules is saved in the *rule base*. The rule base is made accessible to the *inference engine* through the *working memory*.

Working Memory: The working memory contains either the entity structure(s) and other associated information (i.e., synonyms, anti-synonyms, expressions, and functions) or the objects which are mainly rules, predicate objects, and class objects. The restriction to allow either a consultation session or a knowledge development session to take place at any point in time is due to the current memory limitations of Scheme.

Inference Engine: The inferencing mechanisms provided by CESM are *forward/backward* chainings. In either case, the ultimate goal of the inference engine is to confirm or reject an atomic class (i.e., a leaf entity).

User Consultation Module: The interactive process between the user and the inferencing system is the responsibility of this module. The functionality of this module varies depending on the selected inferencing mechanism.

4.3.1 Knowledge Representation

The knowledge representation in CESM is hierarchical. The entity structure is built from a set of entities (each entity is a class or a subclass) which represent real world objects or concepts. Each entity has properties, attributes, and other characteristics which are described by a set of predicates (cf. Figure 4.4). There are semantic relations among predicates which could be synonyms, anti-synonyms, expressions, or functions.

There exists an entity called the *root entity* which has children (entities), and is not itself a child. Children entities which do not have children themselves are called *leaf entities*. An entity structure can be developed such that the most general information is assigned to the root entity, and the most specific information is assigned to the leaf entity. The development of an entity structure using this approach can represent faults from the least specific entity (i.e., the root entity) to the most specific entities (i.e., the leaf entities)—a scheme which is often exercised by experts in the fields of medicine, electronics, and automotive among others.

The predicates are chosen such that they either provide evidence in favor of or against a failure. An entity structure can be constructed to represent the knowledge which is required by *shallow reasoning* without specifying the structural and behavioral information required by its counterpart, *deep reasoning*. Therefore, it is only necessary to construct an entity structure (i.e., a *fault-tree* or more precisely a *precompiled diagnosis knowledge*) for a problem, which is then automatically translated into a set of rules by the knowledge base compiler.

4.3.2 Control Strategies

The inferencing mechanisms for expert systems are primarily based on heuristic searches (i.e., traverses of a tree in which each node represents a problem state and each arc represents a relationship between the states represented by the nodes it connects) with a technique which is able to guide them toward a final goal. Since these search methods (e.g., forward/backward-chaining) do not depend on any particular task or problem domain, they are very powerful.

However, the efficacy of these heuristic search methods is often highly dependent on the way they exploit domain-specific knowledge. Furthermore, they are also unable to overcome the problem of *combinatorial explosion* to which they are vulnerable (Rich, 1983). For this reason, heuristic search methods are often referred to as *weak* search methods (Rich, 1983). Two of these methods which provide the basis for nearly all other search methods are *forward-chaining* and *backward-chaining*.

Depending on the topology of the problem space, it may be significantly more efficient to search either from general to specific (i.e., by forward-chaining) or vice versa (i.e., by backward-chaining). Several of the domain-specific problems, where it is shown that one search algorithm is significantly better than the other, are presented in (Rich, 1983). Consequently, we should determine which of the two control strategies of CESM is more promising, if any, for diagnosing the high-level failures of the experiments hosted by the GPL. To answer this, three considerations

are often evaluated in order to compare the forward-chaining and backward-chaining algorithms (Rich, 1983):

1. Are there more possible goal states or start states? We should explore from the smaller set of states to the larger set of states.
2. Is it important to proceed in the direction that corresponds more closely to a way a technician thinks?
3. In which direction is the branching factor (i.e., the average number of nodes that can be reached directly from a single node) larger?

Usually in a system such as the FHL, there are less *start* states (i.e., high-level cause(s) of a failure) in comparison to *goal* states (i.e., the low-level cause(s) of a failure). That is, forward-chaining is more appropriate than backward-chaining. An error message, (*ROBOT, ERROR-LE-FILL-LATE*), which is received from the event-based controller, can be caused by either a syringe, or a pressurized bladder bottle (high-level failures), assuming the error is caused by a hardware failure. A failure which is related to a syringe can be specialized into a broken syringe, a bent needle, or a clogged needle (low-level failures). Similarly, a failure which is related to a pressurized bladder bottle, can be specialized into a broken bottle, or a ruptured inflatable bag (low-level failures).

Generally, a technician begins the diagnostic process of a failure by conducting *general tests* (which are usually easy and efficient to conduct) in an attempt to reach a conclusion. Since general examination of the symptoms usually is not capable of finding the exact cause(s) of a failure, in the sequel, he either performs *more specialized tests* in order to narrow down the causes of the failure (i.e., he uses the forward chaining technique), or he makes one or several *hypotheses* based on the information accumulated thus far which he then tries to verify (using the backward chaining approach)³. It appears that in the majority of cases related to experiments performed in our FHL laboratory, forward-chaining is more appropriate and natural for diagnostic purposes.

The last consideration, the branching factor, indicates that it might be more appropriate to reason backward since the branching factor is lower going from the cause(s) of low-level failures to high-level failures.

Taking into consideration all three factors, it is nevertheless concluded that, in our case, it may be more appropriate to reason *forward*. It should be stated that the selection of the forward-chaining algorithm does not impair our study, since our software configuration makes it relatively easy to switch back and forth between forward-chaining and backward-chaining.

Within the artificial intelligence framework, CESM provides the necessary environment for the development of an expert system. However, since we have only considered the electrophoresis experiment (i.e., one

³ In medical diagnosis, forward-chaining followed by backward-chaining is customary, that is, a combination of both strategies are used. Knowing when to switch from one to another is often difficult to assess.

entity structure), one high-level diagnoser is sufficient. Therefore, neither the *structured production systems* nor the *distributed systems* need to be considered⁴. Hence, the applicability of CESM with respect to the availability and the appropriateness criteria, set forth earlier, has been demonstrated.

4.4 A Diagnostic Unit and M-ITP-DIAG

The failures which may be encountered in the FHL were classified as either software or hardware failures. As our intention has been to determine the cause(s) of hardware failures only, the diagnostic model of the ITP device should provide the type of information needed by the diagnostic unit.

Since a faulty primary sensor may interrupt a primitive operation, backup sensors were considered for their confirmation, i.e., the *diagnostic* model of the ITP device must represent the backup sensors for both chambers and the capillary. Therefore, the M-ITP-DIAG model of the ITP device is similar to M-ITP-OPER since it is representing the backup sensors.

As stated in Chapter 3, M-ITP-DIAG may contain a smaller amount of information in comparison to MB-ITP (cf. Figures 3.1a, 3.1b, and 3.1c). Although the diagnoser unit must represent the backup air pressure sensors (i.e., LE, TE, and SA), it may not be necessary to include the same

⁴ Note that CESM supports multiple knowledge bases and thereby rule bases.

level of complexity in the M-ITP-DIAG as the M-ITP-OPER. A less detailed diagnostic model of the ITP device is sufficient since the actual operations of the backup sensors need not be monitored by the event-based controller. The diagnostic model of the ITP device is considered to contain the same level of complexity as the controlled model (cf. Figure 3.1b). M-ITP-DIAG may also contain a smaller amount of information in comparison to M-ITP-CONT (cf. Figure 3.1c).

The diagnostic model of the ITP device, an atomic-model, is not included in this thesis since its exclusion is not relevant to our study. It is also expected that the M-ITP-DIAG will be used much less frequently in comparison to M-ITP-OPER and M-ITP-CONT.

The responses of the backup sensors are not sufficient for identification of high-level cause(s) of a failure (cf. Figure 4.5). Consequently, we need additional sources of information (e.g., the response of a vision sensor) in order to determine the high-level cause(s) of a failure in addition to the low-level cause(s). Figure 4.4 depicts the failures (classes or entities) and related symptoms (predicates) that should be examined for determining the cause(s) of a failure. The diagnoser, therefore, requires information that is available from both the backup sensors and the camera (i.e., a vision sensor).

The diagnostic model of the ITP device may respond with any of the values which were defined for the data type predicate (i.e., 'yes', 'maybe yes', 'no', 'maybe no', and 'unknown'). Of course, it may be desirable/necessary to include other forms of logics; e.g., *binary logic* (i.e., 'yes', 'no'), or *fuzzy logic* (i.e., [0,1], where 0.0 \equiv 'no' and 1.0 \equiv 'yes'). All the

sensors in this study are assumed to have threshold type characteristics (cf. Section 2.3) with their responses being either *on* or *off*; i.e., they are of the *binary logic* type.

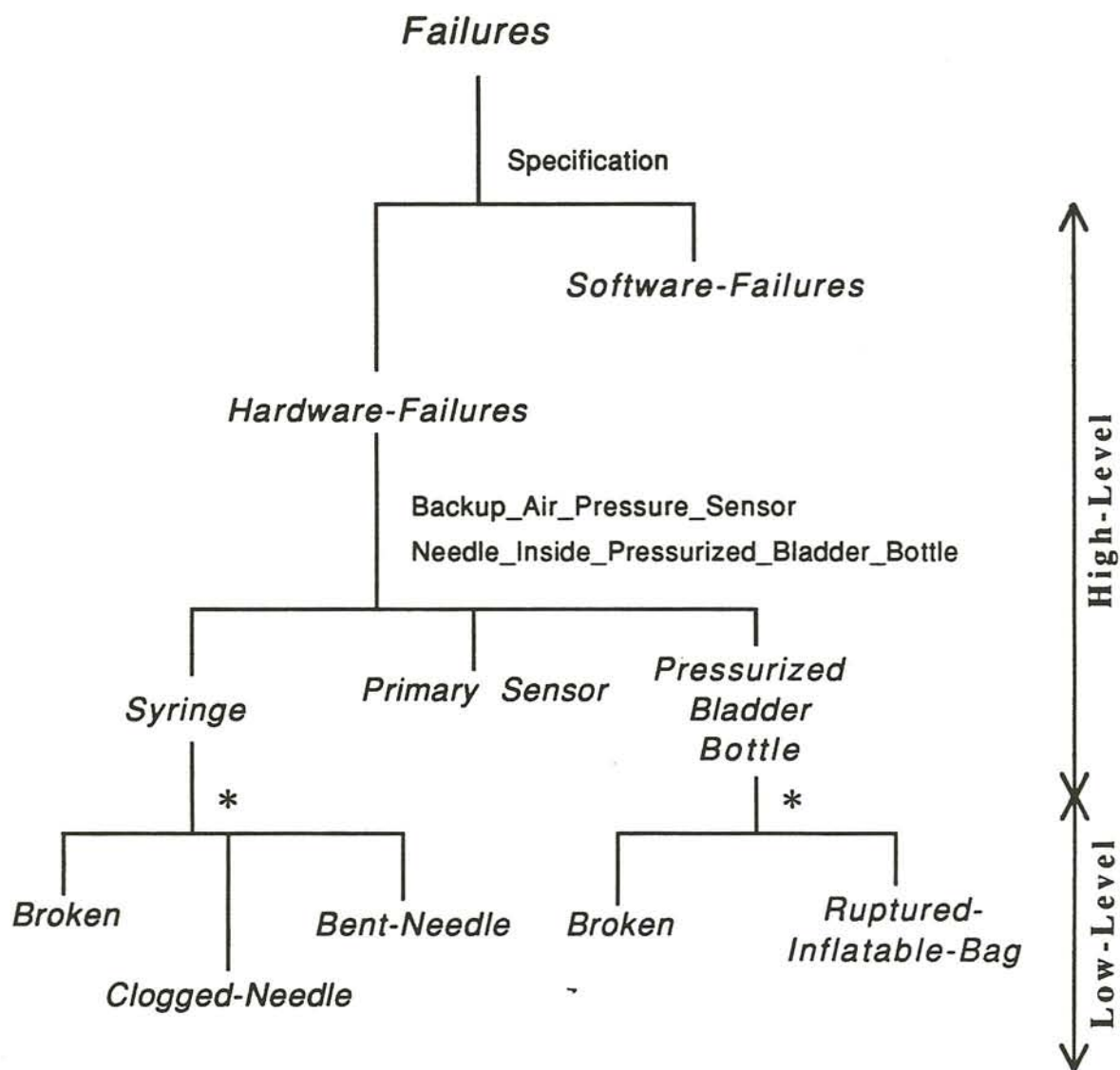
Figure 4.4 shows how, in our FHL example, failures can be partitioned into general (i.e., high-level) and specific (i.e., low-level). Failures of the *pressurized bladder bottle*, the *syringe*, and the *faulty primary air pressure sensor* are classified as high-level failures. Once a high-level diagnoser has determined the possible high-level cause of a high-level failure, a more specialized diagnostic unit (i.e., a low-level diagnoser) may be used to identify the low-level cause(s) of the previously determined high-level failure. Characterized as low-level failures are: *broken syringe*, *clogged needle*, *bent needle*, *ruptured inflatable bag*, and *broken bottle*. High-level failures can be efficiently detected by an event-based controller, and their cause(s) identified by a high-level diagnoser.

Figure 4.5 illustrates how a fault-tree or precompiled diagnosis knowledge may utilize *binary logic* in order to identify the cause(s) of a failure. From this fault-tree, it cannot be determined conclusively which of the three causes is responsible for a hardware failure, it only indicates which is the most likely candidate depending on the received responses for Q2 and Q3. There is, however, an exception to that: '*false*' responses to both Q2 and Q3 in Figure 4.5 do not indicate which of the causes is the most probable. Furthermore, note that the information provided by the backup sensors alone is not sufficient for determining whether the syringe or the pressurized bladder bottle is responsible for the failure, i.e., Q3 is necessary (cf. Figure 4.5). The predicates are expressed as sentences which are

equivalent to the rules compiled by the *knowledge base compiler* component of CESM.

Although it may be desirable/practical to simply replace a syringe, a pressurized bladder bottle, or a faulty sensor rather than repairing it, it is our intention to demonstrate the possibility of using *high-level* and *low-level* diagnostic units for large-scale systems. It may also be necessary to locate the low-level cause(s) through a more thorough analysis if the problem persists. For instance, by replacing a ruptured inflatable bag with a new one, the problem may not be resolved if the cause that was responsible for the damage of the inflatable bag is not removed as well (e.g., if the liquid contained in the bladder is able to dissolve the bladder, or if there is a thorn inside the bottle that perforates the full bladder, or if the control algorithm pushes the needle so far into the bottle that the needle perforates the almost empty bladder).

It is also possible that several devices (e.g., the syringe *and* the pressurized bladder bottle) may be involved in a single failure. E.g., if the control algorithm places the syringe in an incorrect position (not centered on the septum), an injection attempt could damage both the syringe and the pressurized bladder bottle at the same time. Retrospectively, it will be difficult to identify the true cause of the problem since both the syringe and the pressurized bladder bottle are meanwhile defective. The design of diagnostic capabilities that are able to solve such problems are non-trivial. Nonetheless, a diagnostic system may be able to detect/correct one cause/failure at a time and continue progressively until all or most of the failures have been detected and corrected.



* Tests which are able to distinguish among *detailed failures* (i.e., low-level failures) compared to those which are classified as *general failures* (i.e., high-level failures).

Figure 4.4: Diagnostic structure for the electrophoresis experiment; Italics: causes, non-italics: symptoms.

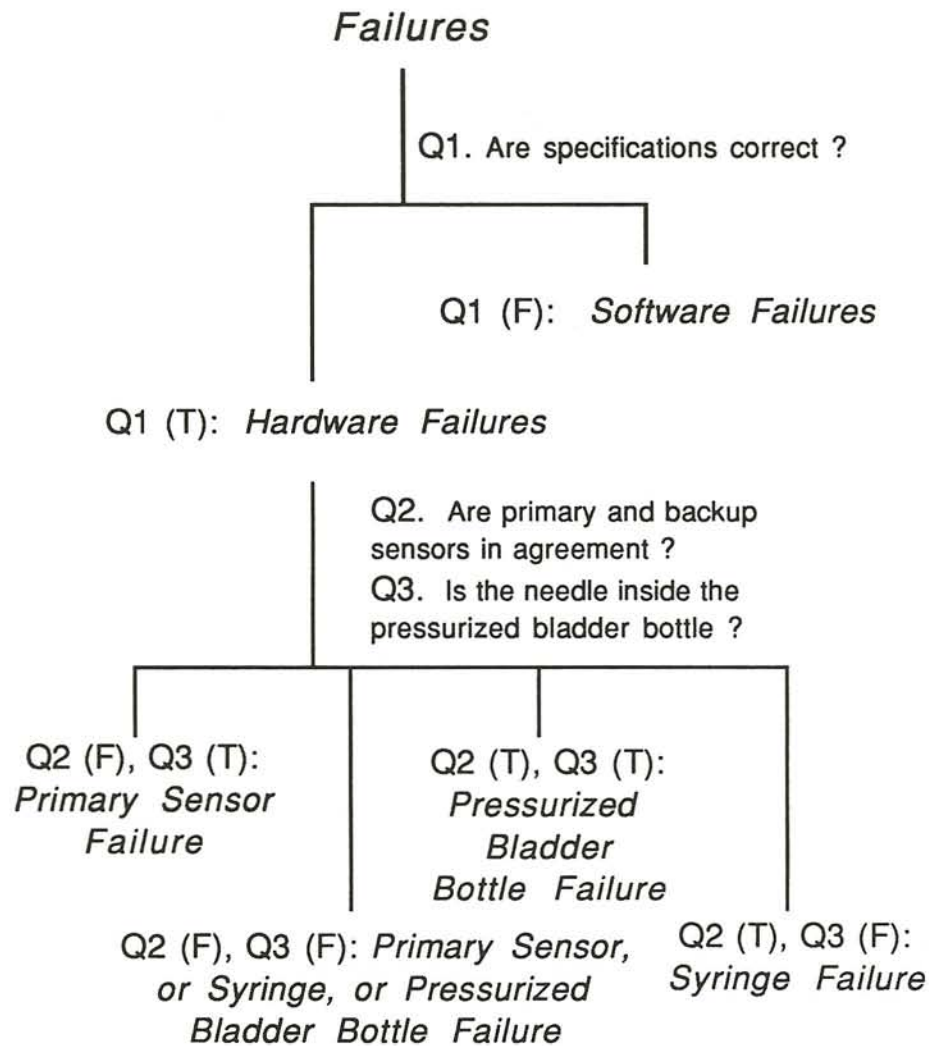


Figure 4.5: Part of the diagnostic structure given in Figure 4.4; binary logic is utilized in determining high-level failures.

It seems important to keep a *track record* of past failures and their repairs. If a failure that was recently removed reoccurs again shortly after it was removed, there is a strong indication that the true cause of the problem had not been identified, and that the repair approach was

unsuccessful. In that case, the diagnoser should either take this new information into account in an extended analysis of the problem, or recognize its own limits and call for help.

In Section 4.2, we concluded that CESM was well suited for the implementation of the diagnostic unit (expert system) assigned to the FHL. We also decided that the diagnoser needs to have access to a vision sensor in addition to the backup air pressure sensors.

The development of a diagnostic unit requires a knowledge base. This knowledge base must be constructed from the knowledge of an expert. Since the operations involved in the electrophoresis experiment are simple, the knowledge is represented in the form of a fault-tree as depicted in Figure 4.4. Since the fault-tree shown in Figure 4.5 contains high-level knowledge, it provides the medium for *shallow-reasoning*. The reasoning process starts from the root entity and traverses to the leaf entities in order to locate high-level hardware failure(s) that are caused during the execution of the primitive operations. Note that the most specific causes (e.g., *clogged-needle*, and *ruptured-inflatable bag*) are at the lowest level of the fault-tree which is shown in Figure 4.4.

There are numerous possibilities to perform a diagnostic task. Figure 4.4 is not a unique fault-tree, and it may not contain all possible failures. Note that the segment of the fault-tree which distinguishes between software and hardware cause(s) is for illustration only, and is not currently implemented.

The fault-tree (i.e., the entity structure or precompiled diagnostic knowledge) is transformed by the *knowledge base compiler* into a set of

rules, an automated process offered by CESM. However, since CESM was implemented as an interactive expert system environment, we could not use CESM for our task as it was, but had to extract portions of the CESM code to form another, a non-interactive expert system environment from it. CESM's interactive consultation capability was thereby eliminated from the code. Our code, instead of asking the *user* to supply the necessary information in an interactive session, obtains the required information directly from *models* which represent actual sensory devices. Note that this is not equivalent to the *model-based* approach discussed earlier. These models are merely used since our system is not currently connected to an operational set-up (i.e., FHL) with appropriate real sensory devices.

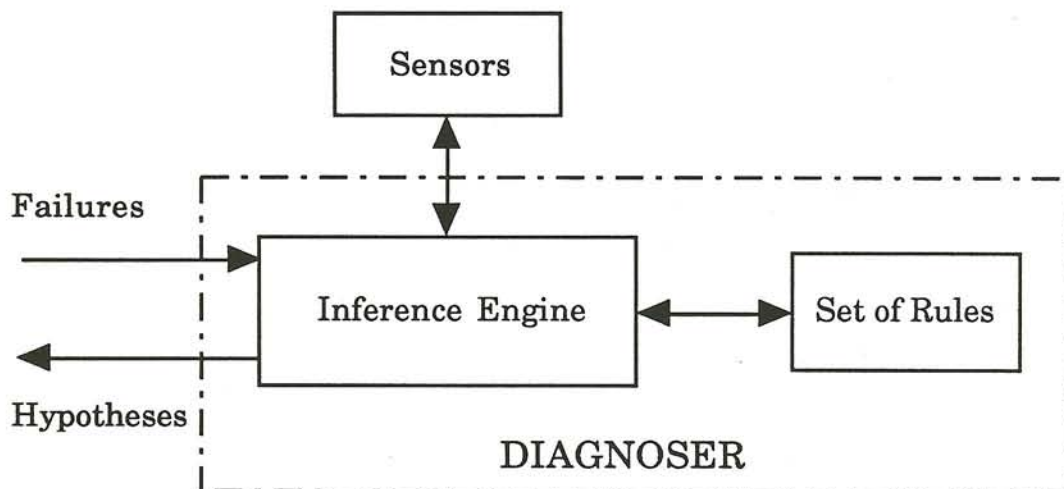


Figure 4.6: A customized diagnostic unit.

This customized diagnostic unit —an inference engine operating in forward chaining mode, a set of heuristic rules, and the above models of the measurements devices—, requires rules which are written in advance. These rules do not exhibit the same structure as those which would be generated automatically by CESM from an entity structure. This diagnostic unit is shown in Figure 4.6.

The necessity for a non-interactive expert system can be envisioned for an automated environment with *none* or *minimal* human intervention, as this is truly the case in the context of the SSF. That is, once an error is detected, the event-based controller sends a message to a high-level diagnoser indicating the encountered failure. The schematic of such an environment is depicted in Figure 4.7. Thereafter, the diagnoser begins its diagnostic process by identifying the cause(s) of the failure. At the SSF, it may be impractical to have a human operator consult with an expert system for identifying the possible cause(s) of a failure, instead, the expert system must be able to directly and unsupervised interrogate sensors to obtain the necessary information enabling its decision making process.

An initial version of the knowledge base (or equivalently a set of rules) should be constructed in advance, but it must be possible to constantly upgrade it incorporating new experiences in order to cope with a constantly changing world due to system upgrading and system degradation. However, in the currently available expert system, which is a modified version of CESM, all rules must be written in advance. There is currently no provision in the code for learning new rules on the fly.

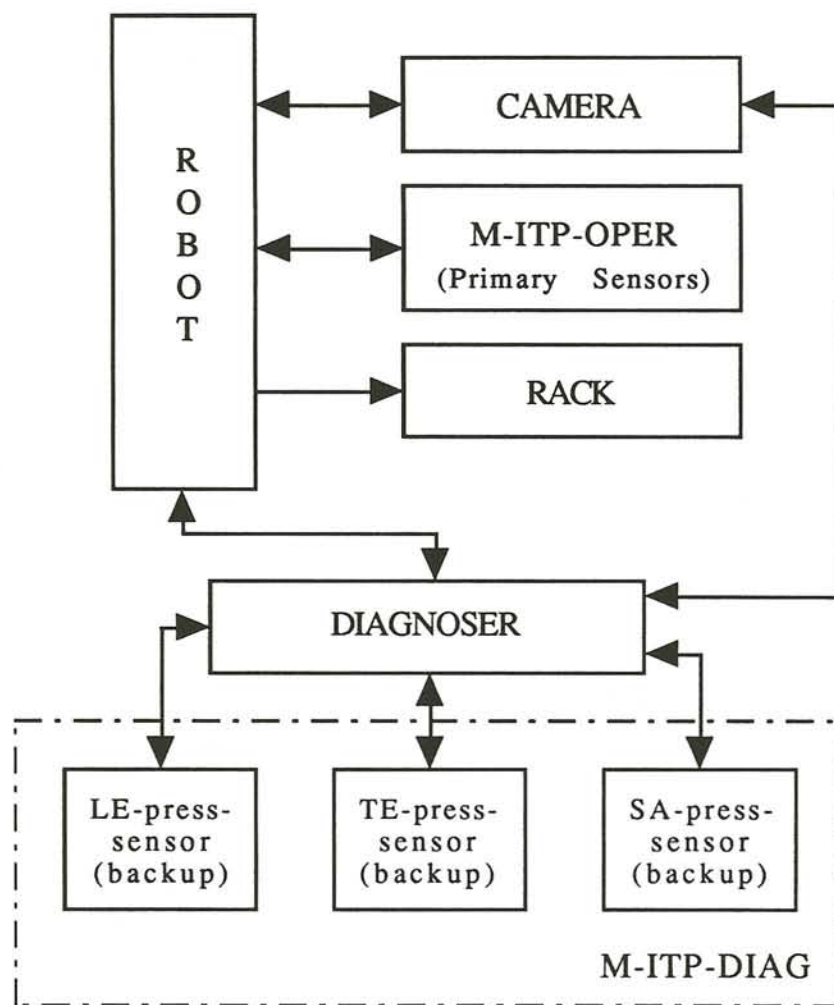


Figure 4.7: Architecture of the FHL depicted in Figure 3.7 with the addition of a diagnostic unit.

In the event of receiving an error message from the robot, the diagnoser begins to evaluate the antecedences of its rules by using the models of the appropriate sensors (i.e., the LE, TE, and SA backup air pressure sensors, and the vision sensor). The triggered rules (i.e., rules

the antecedences of which are satisfied) are stored in a queue. Then, one of the triggered rules is chosen by a conflict resolution algorithm, and it is fired, i.e., its consequence is executed or determined.

This expert system selects the first triggered rule in the queue —a primitive form of conflict resolution—, and fires it which results in evidence accumulation for one or more of the entities or failures. The evidence accumulation could result in CEF, EF, CEA, EA, N, and X (cf. Section 4.2). A hypothetical case is considered next in order to clarify the diagnostic process of an encountered failure. When an error message (*ROBOT, ERROR-LE-FILL-LATE*) is received, the diagnoser inquires the status of the backup leading electrolyte air pressure sensor in order to determine the correctness of the primary sensor (cf. Figures 4.5 and 4.7). Thereafter, if the primary sensor and the backup sensor agree, additional information is needed in order to determine which of the remaining devices (i.e., the syringe or the pressurized bladder bottle) is responsible for the failure. On the other hand, if the two sensors disagree, the primary sensor has been identified to be faulty, and can be replaced without any additional diagnostic efforts. Of course, it would also be possible that the backup sensor is defective as well which would then indicate a simultaneous *double failure* which is more difficult to diagnose. In order to minimize the chances for undetected faulty backup sensors, it is recommended to interchange the primary and the backup sensor on a regular (scheduled) basis (which can be done in software).

If the primary sensor is not responsible for the failure, then more rules will be examined. Consequently, all the newly triggered rules, if any,

must be added to the queue. In the case of existing triggered rules in the queue, the first rule is selected and fired, a process which continues until no rules are left in the queue, and no further rules are triggerable. The results of the above simulation is depicted below.

Sensors' Responses:

Backup_Air_Pressure \equiv yes

Needle_Inside_Pressurized_Bladder_Bottle \equiv maybe no

Diagnostic unit:

Primary-Sensor: (0.00 0.98 0.02)

Syringe: (0.70 0.00 0.09)

Pressurized Bladder Bottle: (0.21 0.21 0.09)

Note that the responses of the sensors are consistent with the data type values of the predicates listed in Section 4.2. Therefore, the results of the diagnosis, as shown above, reflect the received responses.

Our goal has not been to develop an expert system which is able to determine causes of all failures, but instead, to demonstrate two issues: first, the ITP device should be modeled according to its purpose, and second, a high-level diagnostic unit which is designed to function at the same level of complexity as an event-based controller may be beneficial.

It may not be necessary to rely entirely on hardware redundancy if other means are available. More accurate and dependable methods in

comparison to the backup sensors can be utilized as well. For instance, it may be desirable to substitute backup air pressure sensors with a video camera to quantify the amount of the liquid inside a chamber.

The application of hierarchical diagnostic systems may reduce the required effort of knowledge acquisition, and simplify the inferencing mechanisms in comparison to a diagnostic system which is aimed at all levels of failures within a complex system. Moreover, hierarchical diagnostic systems can be more easily implemented in a distributed computing architecture than monolithic diagnosers, thereby reducing the amount of time needed for the decision making process which may be essential in the context of a real-time diagnoser.

4.5 Constraint Driven Diagnostic Units

Cost is an important criterion in most system designs, and consequently results in the exclusion of certain design options. Considering measurement devices, which would be necessary for the diagnostic units, the selection procedure depends on several criteria such as *cost*, *reliability*, and *response time*. We note that the cost can be either the unit's purchase cost or its operation cost. Generally, such criteria are inter-related since a more expensive unit (e.g., a video-camera) often produces more reliable responses, and requires a longer time duration at a higher operation cost in comparison to a less expensive unit (e.g., a threshold type sensor). We will not be concerned with the exact inter-

relationship of time, cost, reliability, or any other constraints. Instead, we lump together the operation cost and the response time into a single term: '*time/cost*'. This term will be treated as a performance index, i.e., we assume that a higher *time/cost* parameter results in a higher quality analysis. In cases where this is not automatically true, we simply will have to redefine the term *time/cost* to make it true.

The importance of selecting a suitable measurement device, which minimizes the interrogation time/cost, lies in the fact that it may be possible to conduct a certain task *successfully* with a lesser amount of accuracy being the result of a lower operation cost and a shorter time duration.

To illustrate the above discussion, consider an automotive technician. He/She does not need to disassemble an engine in order to find out what may have possibly caused an engine related failure. Instead, in the majority of cases, it is sufficient to conduct simple test(s) and arrive at reasonably accurate/reliable hypotheses. Of course, the reason for taking such an approach is simple —the technician is trying to minimize the repair time and cost.

Our intention is to integrate the time/cost constraint into the design of a diagnostic unit which is expected to operate in real-time. Such a diagnostic unit shall be referred to as a *Constraint Driven Diagnostic Unit* (CDU). We should caution that *certainty* (i.e., accuracy) offered by various sensory devices are often not the same. In fact, usually as the certainty/reliability of knowledge increases, the actual interrogation time/cost increases as well.

A diagnostic unit may have access to several sensory devices in order to inquire certain types of information. However, it may be significantly more economical to utilize one subset of available sensory devices instead of using the entire set or another subset. A diagnostic unit, therefore, should be able to minimize the interrogation time/cost by selecting an appropriate set of sensory devices which collectively require the least time/cost in determining one or more hypotheses. Recall that a diagnostic unit which is to operate in the FHL must obtain the required information in real-time.

The selection process of the measuring devices, therefore, must follow a scheme or a strategy which is applicable in all situations. Well known algorithms such as A* (Hart, 1968, 1972) and B* (Berliner, 1979) minimize the necessary efforts for a search of a graph or a tree.

Additionally, a search algorithm should determine whether to discontinue (i.e., a *reasonable dependable hypothesis* is concluded) or continue (i.e., an acceptable hypothesis is not concluded) its search, assuming there exists a goal state. That is, appropriate provisions must be available to such a search algorithm, either for its termination or its continuation whichever necessary.

The logic from which a diagnostic unit can determine whether '*enough*' evidence has been accumulated for a hypothesis, requires *prior knowledge*. This knowledge can be one or more lower-level threshold value(s) which are assignable to the hypotheses or entities. Appropriately selected threshold values can determine the *goodness* of the accumulated evidence for each of the hypotheses at any instance during a failure analysis.

One way of obtaining such knowledge is by storing an adequate number of the previous case histories for which the same failure had been analyzed. Given all the encountered failures and their subsequent recoveries, the required knowledge (i.e., threshold values) can be determined. The selection of an inappropriate threshold value can have adverse effects on the performance of a diagnostic unit. That is, if the threshold value is chosen to be relatively large, then it may be necessary to interrogate all the measurement devices, while on the other hand selecting a relatively small threshold value may result in a hypothesis which is not worth any investigation (i.e., step 3 of the diagnostic process outlined in Section 4.1).

In the set-up of the electrophoresis experiment, we assumed the utilization of threshold type sensors (i.e., primary and backup) and a camera in detecting high-level failures (cf. Figure 4.4). We may also choose to use one or more vision sensors in place of the primary and the backup sensors if they are available. That is, assuming the availability of several devices for a particular measurement, the time/cost constraint can be used to select one among them.

Although, we have not yet mentioned anything about the performance (i.e., accuracy, reliability, time, and cost) of a vision sensor or a threshold type sensor, we assume a higher certainty factor for the former in comparison to the latter. The degree of certainty offered by measuring devices can influence the diagnostic process with respect to the time duration and the desired certainty of a hypothesis.

It is illustrated in Figure 4.8 that two devices can be used to measure the volume of a liquid inside a container. The hypothetical time and cost measurements for a camera (Volume_Camera) and a backup air pressure sensor (Backup_Air_Pressure) are represented graphically (i.e., various segments of the concentric circles). Note that the certainty of each response (i.e., length of the arc) is assumed to be a function of the associated time/cost. Thus, the camera's responses, as depicted in Figure 4.8, provide a higher certainty in comparison to the threshold type sensor.

Time/Cost Assignments For Interrogation Of Sensors

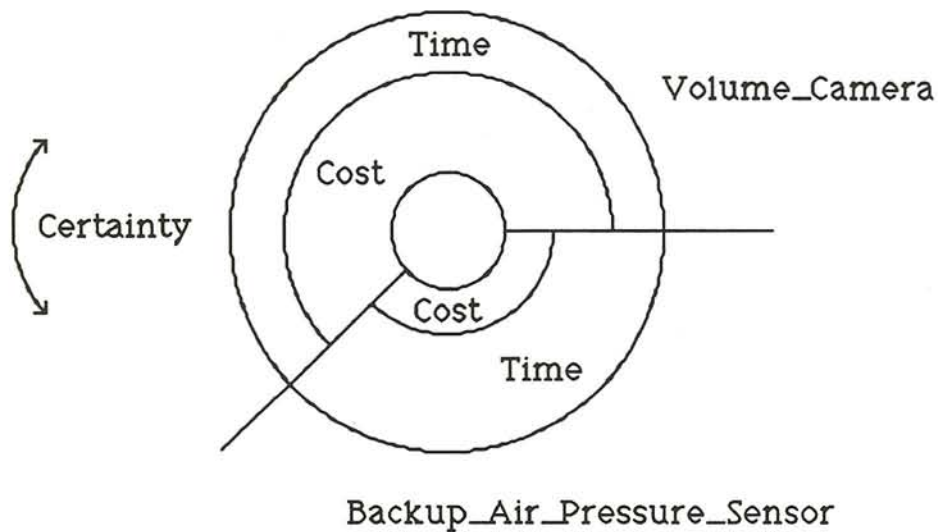


Figure 4.8: Graphical representation of time/cost.

Due to our earlier assumption, each sensor will be assigned a single time/cost measurement. Moreover, each sensor will also be assigned a qualitative certainty factor as well. As mentioned earlier, uncertainty or certainty handling using qualitative words (i.e., usually and rarely) is supported by CESM. Therefore, the assignments of qualitative certainties to each sensory device or predicate is straightforward. The certainty of each predicate can be either *always* \Rightarrow (1.0 0.0 0.0), *usually* \Rightarrow (0.7 0.0 0.3), *rarely* \Rightarrow (0.0 0.7 0.3), and *never* \Rightarrow (0.0 1.0 0.0). Although, it may be desirable to quantitatively evaluate the certainty of each sensor, the usefulness of this approach will be demonstrated equally well with the assigned qualitative certainties.

To illustrate the foregoing discussion, we shall present an example for explanation purposes instead of Figure 4.4. Since CESM is our expert system shell, a fault-tree (i.e., a knowledge structure) which consists of a set of entities and a set of predicates is constructed. Figure 4.9 illustrates this precompiled diagnostic knowledge where each node (e.g., B2), except the root-entity (i.e., Failures), contains several predicates (e.g., B2_P1, B2_P2, and B2_P3) with various interrogation time/cost assignments. Recall that the root-entity always has an evidential status of (1.0 0.0 0.0), thereby there is no need for any predicate(s). The assigned values to the predicates indicate the necessary time/cost interrogation before receiving any responses from the measurement devices.

An expanded version of CESM which facilitates the time/cost assignments of predicates in an interactive environment is referred to as

CDU. Predicates with unknown time/cost (e.g., B1_P2, B2_P1, and B4_P1) are currently assigned the highest interrogation time/cost. Although it would be possible to assume the contrary (i.e., unassigned predicates require the lowest interrogation time/cost), such an option is not currently available to the users of CDU. Note that all the predicates which are depicted in Figure 4.9 offer the same degree of certainty irrespective of their time/cost interrogation. In particular, when the certainty qualifier of any predicate is absent as those which are shown in Figure 4.9, its certainty qualifier is equated to 'usually' by default.

Furthermore, it is conceivable for each predicate to be assigned to any number of entities. Since upon the interrogation of a predicate, one or more entities may become affected, the diagnostic unit can potentially fire one or more of the triggered rules.

It is also possible that several predicates require the same time/cost interrogation. If this occurs, the diagnostic unit should employ an appropriate conflict resolution algorithm to select an appropriate measurement device. For instance, if two predicates of two entities require the same time/cost interrogation, then the conflict resolution algorithm should be able to choose the predicate for which its entity has accumulated the most or the least evidence thus far. There exist other strategies which may be considered as well. Currently, the modified CESM or CDU interrogates all the predicates which require equal time/cost.

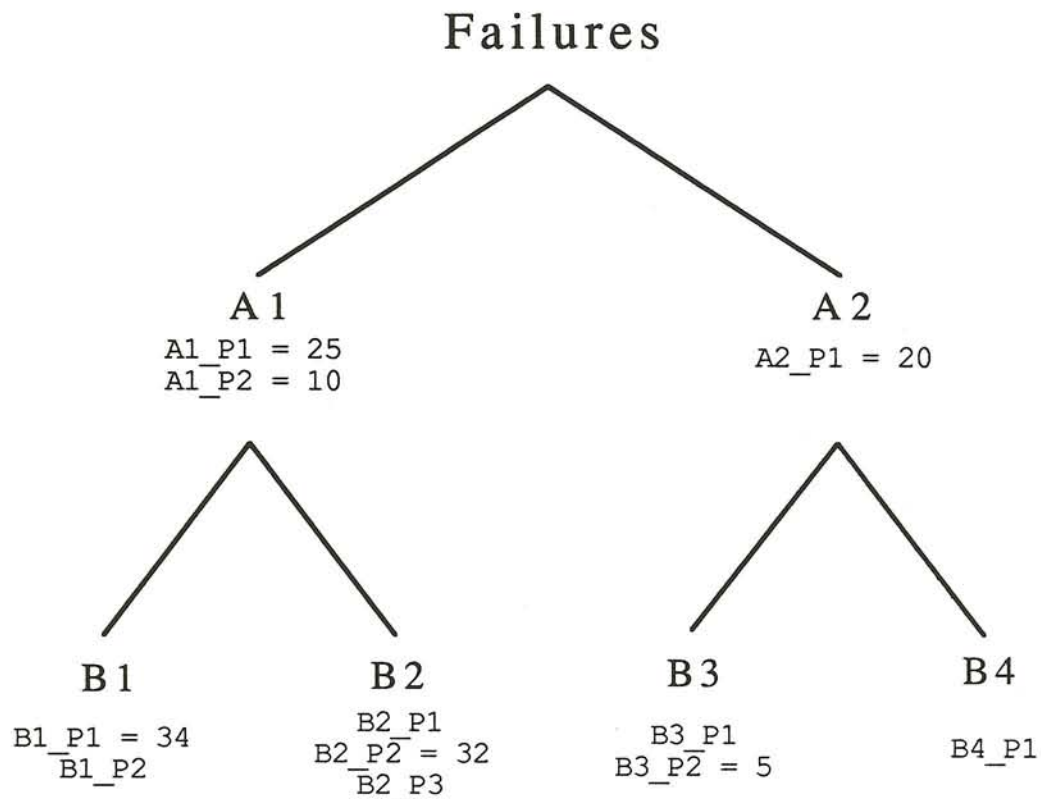


Figure 4.9: A hypothetical diagnostic structure with various measurement devices.

Given Figure 4.9, an expert system (e.g., CDU) should attempt to reach at least one goal state (i.e., B1, B2, B3, or B4) by minimizing the total interrogation time/cost. Although, the *A** algorithm ensures that the minimum time/cost path of a problem graph is found, provided one exists, the *Best-First search algorithm* has been implemented instead since it was easier to implement. A description of the details of the Best-First algorithm can be found in many A.I. textbooks, e.g., Rich (1983).

Each predicate (sensor) is assigned a variable which represents its actual interrogation time/cost.

As stated earlier, a diagnostic unit must be able to terminate its searching process based on some prespecified criteria if it is to minimize the time of a search. The evidential status of each entity, except the root-entity, is calculated by CESM and represented by a quadruple (cf. Section 4.3). A diagnostic process can be terminated either by assigning an acceptable minimum to any of the elements of the quadruple of a leaf-entity, or various forms of their combinations, or by assigning a maximum accumulated interrogation time/cost of all sensors. Currently a diagnostic process may be discontinued when either of the elements EF or EA of the quadruple of any of the goal states reaches a previously assigned threshold value (e.g., 0.3). Currently the user is allowed to specify this threshold value before the start of a consultation session. Therefore, the user's freedom to either continue or terminate a consultation session depends on the selected threshold value. As mentioned earlier, the selection of an inappropriate threshold value can either prolong the diagnostic process or result in hypotheses with inadequate certainties.

The steps of a consultation session for Figure 4.9 is depicted in Figure 4.10. The steps *zero to five* illustrate how the fault-tree (i.e., Figure 4.9) is searched in order to reduce the interrogation time/cost. It is noted that only five of the predicates have been queried; they are identifiable as bold numbers in the *inquiry order* column. Also note that each predicate's response is represented as the first element of its 3-tuple in the column *predicates..* For example, when the response to B2_P2 is 'yes', it is

sufficient to be displayed as (B2_P2 . 0.5) instead of (B2_P2 . (0.5 0.0 0.5)) (refer to Figure 4.10).

At the beginning, *step zero*, all the entities are *unknown* (0.0 0.0 1.0) except the root-entity (1.0 0.0 0.0), which remains unchanged throughout the consultation session. Steps *one* through *five* show that only one predicate is chosen in each step and queried from the user.

It must be emphasized that reaching a leaf-entity may not be sufficient for the diagnostic process to be discontinued, depending on the selected threshold value. That is, although the entity B1 in Figure 4.10 has accumulated some evidence (B1 \equiv (0.24 0.0 0.75)) at the end of *step two*, the diagnostic process had to be continued until a goal entity has accumulated '*enough*' evidence (i.e., either $EF \geq 0.3$ or $EA \geq 0.3$).

Each of the five steps shows that once a predicate is interrogated, the previous evidential status for its corresponding entity is updated accordingly. Obviously, every received response from a measuring device may not necessarily result in a change of an entity's evidential status. For instance, when A2_P1 was interrogated, the response was *unknown* (i.e., A2_P1 \equiv don't know), thereby the evidential status of A2 remained unchanged. Furthermore neither of the entities B3 and B4 could be considered for a possible goal entity, and thereby their predicates would not become eligible. Once the third element of an entity's quadruple, N, is not equal to 1 (i.e., the entity has accumulated some evidence), then the predicate(s) of any of its immediate children (i.e., entities) become eligible. This is due to the fact that the knowledge structure is constructed assuming a hierarchy of faults.

Thereafter, in *step four*, the next appropriate predicate (i.e., A1_P1) is interrogated. Due to the content of the received response, the evidential status of the A1 changes (cf. Figure 4.10). This step, however, cannot influence the evidential status of the goal entities B1 and B2 unless the previously fired rule(s) which do not require any further interrogation of predicate(s) are revised (i.e., they are 'un-fired' and fired with the modified knowledge). Roughly speaking, this form of reasoning falls within the framework of *non-monotonic reasoning* (NMR) (Genesereth and Nilsson, 1987). In the current implementation of CDU, however, the consultation process must continue.

Step *five* of this consultation session corresponds to the interrogation of the B1_P1, the next appropriate predicate. The received response resulted in $B1 \equiv (0.35 \ 0.0 \ 0.65)$, which consequently prompts the user to either terminate or continue the diagnostic process (i.e., $EF \geq 0.3$ for B1).

If the user decides to continue, the remaining eligible predicates will be interrogated accordingly. Again, when any of the leaf-entities reaches an appropriate evidential status, the user may either continue or discontinue the consultation session, a process which continues until there exist no more eligible predicates.

Step Zero:

| <u>ENTITIES</u> | <u>PREDICATES</u> | <u>TIME COST</u> | <u>INQUIRY ORDER</u> |
|--------------------|-------------------|----------------------|--------------------------|
| (A1 (0.0 0.0 1.0)) | (A1_P1 . U) | 25 | (N) |
| | (A1_P2 . U) | 10 | (N) |
| (B1 (0.0 0.0 1.0)) | (B1_P1 . U) | 34 | (N) |
| | (B1_P2 . U) | — | (N) |
| (B2 (0.0 0.0 1.0)) | (B2_P1 . U) | — | (N) |
| | (B2_P2 . U) | 32 | (N) |
| | (B2_P3 . U) | — | (N) |
| (A2 (0.0 0.0 1.0)) | (A2_P1 . U) | 20 | (N) |
| (B3 (0.0 0.0 1.0)) | (B3_P1 . U) | — | (N) |
| | (B3_P2 . U) | 5 | (N) |
| (B4 (0.0 0.0 1.0)) | (B4_P1 . U) | — | (N) |

Step one:A1_P2 \equiv maybe yes

| | | | |
|----------------------|-----------------------|----|-----|
| (A1 (0.35 0.0 0.65)) | (A1_P1 . U) | 25 | (N) |
| | (A1_P2 . 0.5) | 10 | (1) |
| (B1 (0.0 0.0 1.0)) | (B1_P1 . U) | 34 | (N) |
| | (B1_P2 . U) | — | (N) |
| (B2 (0.0 0.0 1.0)) | (B2_P1 . U) | — | (N) |
| | (B2_P2 . U) | 32 | (N) |
| | (B2_P3 . U) | — | (N) |
| (A2 (0.0 0.0 1.0)) | (A2_P1 . U) | 20 | (N) |
| (B3 (0.0 0.0 1.0)) | (B3_P1 . U) | — | (N) |
| | (B3_P2 . U) | 5 | (N) |
| (B4 (0.0 0.0 1.0)) | (B4_P1 . U) | — | (N) |

Step Two:B2_P2 \equiv yes

| | | | |
|----------------------|-----------------------|----|-----|
| (A1 (0.35 0.0 0.65)) | (A1_P1 . U) | 25 | (N) |
| | (A1_P2 . 0.5) | 10 | (1) |
| (B1 (0.0 0.0 1.0)) | (B1_P1 . U) | 34 | (N) |
| | (B1_P2 . U) | — | (N) |
| (B2 (0.24 0.0 0.75)) | (B2_P1 . U) | — | (N) |
| | (B2_P2 . #T) | 32 | (2) |
| | (B2_P3 . U) | — | (N) |
| (A2 (0.0 0.0 1.0)) | (A2_P1 . U) | 20 | (N) |
| (B3 (0.0 0.0 1.0)) | (B3_P1 . U) | — | (N) |
| | (B3_P2 . U) | 5 | (N) |
| (B4 (0.0 0.0 1.0)) | (B4_P1 . U) | — | (N) |

Step Three:A2_P2 \equiv don't know \Rightarrow NO CHANGEStep Four:A1_P1 \equiv maybe yes

| | | | |
|----------------------|-----------------------|----|-----|
| (A1 (0.58 0.0 0.42)) | (A1_P1 . 0.5) | 25 | (4) |
| | (A1_P2 . 0.5) | 10 | (1) |
| (B1 (0.0 0.0 1.0)) | (B1_P1 . U) | 34 | (N) |
| | (B1_P2 . U) | — | (N) |
| (B2 (0.24 0.0 0.75)) | (B2_P1 . U) | — | (N) |
| | (B2_P2 . #T) | 32 | (2) |
| | (B2_P3 . U) | — | (N) |
| (A2 (0.0 0.0 1.0)) | (A2_P1 . U) | 20 | (3) |
| (B3 (0.0 0.0 1.0)) | (B3_P1 . U) | — | (N) |
| | (B3_P2 . U) | 5 | (N) |
| (B4 (0.0 0.0 1.0)) | (B4_P1 . U) | — | (N) |

Step Five:B1_P1 \equiv maybe yes

| | | | |
|----------------------|-----------------------|----|-----|
| (A1 (0.58 0.0 0.42)) | (A1_P1 . 0.5) | 25 | (4) |
| | (A1_P2 . 0.5) | 10 | (1) |
| (B1 (0.35 0.0 0.65)) | (B1_P1 . 0.5) | 34 | (5) |
| | (B1_P2 . U) | — | (N) |
| (B2 (0.24 0.0 0.75)) | (B2_P1 . U) | — | (N) |
| | (B2_P2 . #T) | 32 | (2) |
| | (B2_P3 . U) | — | (N) |
| (A2 (0.0 0.0 1.0)) | (A2_P1 . U) | 20 | (3) |
| (B3 (0.0 0.0 1.0)) | (B3_P1 . U) | — | (N) |
| | (B3_P2 . U) | 5 | (N) |
| (B4 (0.0 0.0 1.0)) | (B4_P1 . U) | — | (N) |

— *Time/cost* is unknown.
 U Unknown.
 N not interrogated.

Figure 4.10: Steps of a consultation session.

Results after the interrogation of five predicates:

Unchallenged evidence for A1.
 Weak evidence for B1.
 Weak evidence for B2.

A few other experiments based on Figure 4.9 are conducted. Their results are depicted in Tables 4.1-3. The column *case* identifies any of the six consultation sessions. The column *time/cost* depicts the total interrogation time/cost values for each case. The third and fourth column

from the left signify the particular hypotheses for each case. The remaining column, *Diagnoser*, identifies the chosen method: CDU or CESM.

Table 4.1: Numerical results of consultation sessions for cases (a), (b), and (c).

| <i>Case</i> | <i>Time/Cost</i> | <i>A1</i> | <i>B2</i> | <i>Diagnoser</i> |
|-------------|------------------|-----------------|-----------------|------------------|
| (a) | 42 | (0.70 0.0 0.30) | (0.35 0.0 0.65) | CDU |
| | 241 | (0.91 0.0 0.09) | (0.58 0.0 0.42) | CESM |
| (b) | 42 | (0.70 0.0 0.30) | (0.48 0.0 0.65) | CDU |
| | 241 | (0.91 0.0 0.01) | (0.50 0.0 0.16) | CESM |
| (c) | 121 | (0.70 0.0 0.30) | (0.35 0.0 0.65) | CDU |
| | 241 | (0.70 0.0 0.30) | (0.58 0.0 0.42) | CESM |

Table 4.2: Numerical results of consultation sessions for cases (d) and (e).

| <i>Case</i> | <i>Time/Cost</i> | <i>A2</i> | <i>B3</i> | <i>Diagnoser</i> |
|-------------|------------------|-----------------|-----------------|------------------|
| (d) | 35 | (0.70 0.0 0.30) | (0.35 0.0 0.65) | CDU |
| | 140 | (0.70 0.0 0.30) | (0.58 0.0 0.42) | CESM |
| (e) | 35 | (0.70 0.0 0.30) | (0.48 0.0 0.51) | CDU |
| | 140 | (0.70 0.0 0.30) | (0.50 0.0 0.33) | CESM |

Table 4.3: Numerical results of consultation sessions for case (f).

| <i>Case</i> | <i>Time/Cost</i> | <i>A2</i> | <i>B4</i> | <i>Diagnoser</i> |
|-------------|------------------|-----------------|-----------------|------------------|
| (f) | 140 | (0.70 0.0 0.30) | (0.35 0.0 0.65) | CDU |
| | 140 | (0.70 0.0 0.30) | (0.35 0.0 0.65) | CESM |

A partial graphical representation of Tables 4.1-3 is depicted in Figure 4.11. Only the EF segment of each evidential status of each hypothesis is graphed in Figure 4.11. The filled/unfilled 'diamond' shape symbols with the lowest interrogation time/cost signify the utilization of CDU, while the remaining ones signify the utilization of CESM (cf. Figure 4.11 and Tables 4.1-3). The utilization of CESM usually results in hypotheses with higher certainties in comparison to the utilization of CDU alone since potentially the latter should interrogate fewer times whenever applicable.

Figures 4.11 (a), (b), and (c) illustrate some variations of consultation sessions which resulted in accumulating evidence for A1 and B2. It is shown that by interrogating more predicates, the certainty of each hypothesis is increased. It is noted that the difference between lower level hypotheses is always less than or equal to the difference between the higher level hypotheses. That is, as the depth of a diagnostic structure increases, the impact of interrogating measuring devices with higher time/cost becomes less significant.

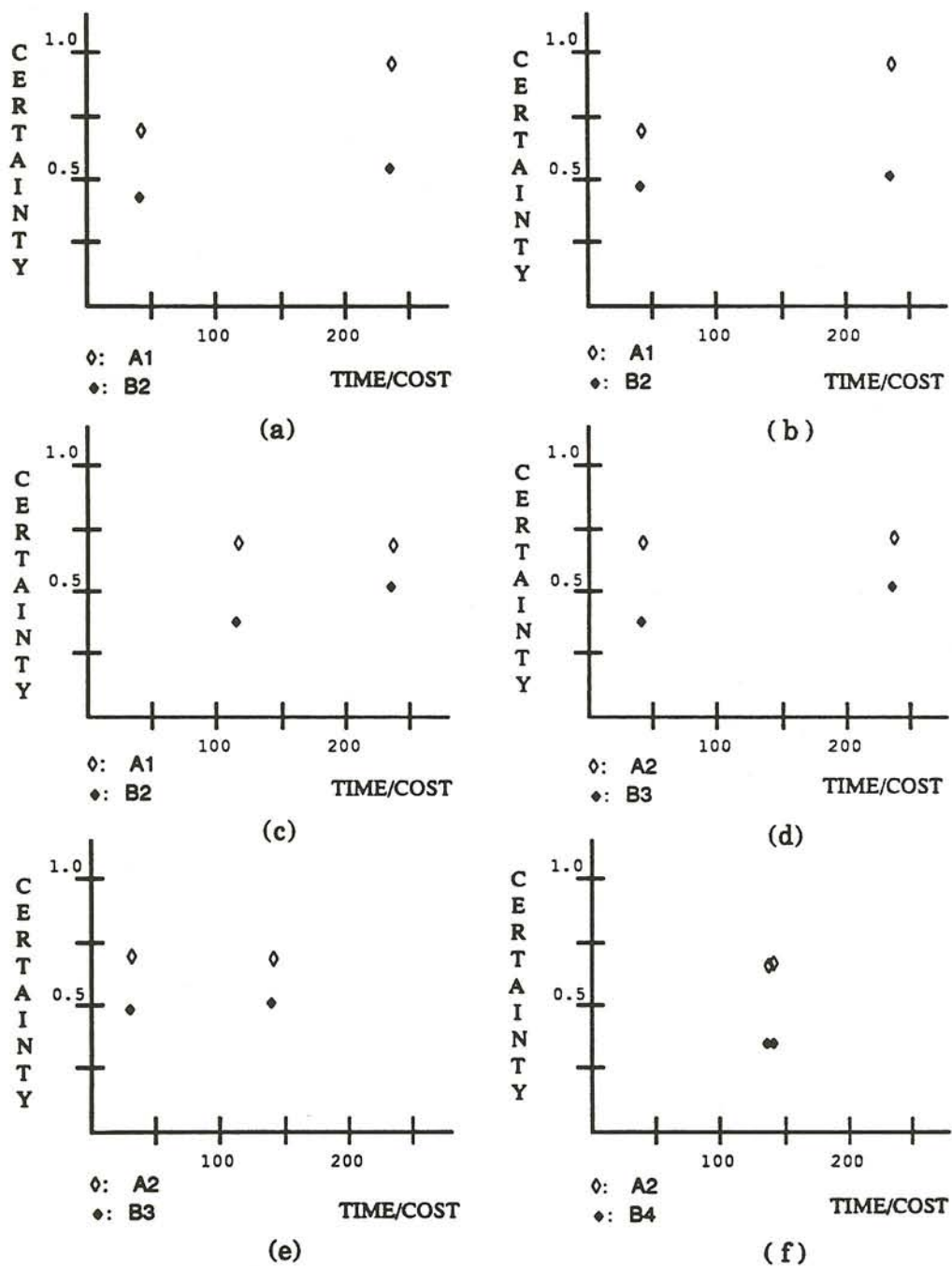


Figure 4.11: Partial graphical representation for Tables 4.1-3.

For instance, Figure 4.11 (b) shows that even though the difference between the certainty level of A1 from CESM is higher than CDU, the difference between certainty level of B2 obtained by these two methods is insignificant. Figures 4.11 (d) and (e) demonstrate various certainty levels for another set of hypotheses: A2 and B3. Note that in the latter, the difference is insignificant which is a reflection of the received responses.

Referring to Figure 4.9, there exists only one predicate for B4. Therefore, there is no difference between the utilization of CDU and CESM, i.e., at worst CDU behaves identical to CESM. Consequently, if there does not exist more than one predicate for each entity, CESM is an instantiation of CDU —it always generates the same hypotheses with the same levels of certainties as CDU.

It must be emphasized that it is not the purpose of CDU to conclude the same degree of certainty as CESM. This is in accord to what we stated at the beginning of this section —in certain instances, it is possible to detect the failures successfully without inquiring all the information which may be made available to a diagnostic unit.

The reduction in time/cost in the context of the SSF can have significant effects. The operation cost of the GPL is much higher than a comparable laboratory on Earth. Subsequently, an optimum usage of the resources of the GPL is of essential concern. Constraint driven diagnostic units can effectively contribute to economical and optimal operation of the FHL and thereby the GPL. It is also noted that the diagnostic units may also be employed in other applications afforded by artificial intelligence.

CHAPTER 5

CONCLUSIONS

In this work, we have demonstrated the applicability of the DEVS formalism to the analysis of an intelligent controller. It was shown that various model representations of the ITP instrument are important in the model construction of intelligent agents. The ITP instrument with various levels of abstraction was modeled for operation, control, and diagnostic purposes. All of these models were abstracted from the reference model.

It was suggested that hierarchical diagnosers for hierarchical event-based controllers can be beneficial. That is, hierarchical diagnostic units are to be constructed on the basis of both expert systems and model-based programming techniques, and not a strict hierarchy within either of the two. The applicability of a high-level diagnostic unit, operating at the same level of complexity as an event-based controller, was demonstrated. It was proposed that the application of hierarchical diagnostic agents may reduce the efforts required in knowledge acquisition, and may simplify the inferencing mechanisms in comparison to a single diagnostic agent which is aimed at all levels of failures within a complex system. Moreover, hierarchical diagnostic agents can be more easily implemented in a distributed computing architecture than monolithic diagnosers, thereby reducing the amount of time needed for the decision making process which may be essential in the context of a real-time diagnoser.

Requirements for real-time diagnostic units was mentioned for a remotely operated laboratory such as the GPL. A customized diagnostic

unit, which requires precompiled rules, was constructed by using CESM in order to allow it to interrogate directly the models of the appropriate measuring devices. Nonetheless, such an expert system is inadequate since it is not able to utilize rules which can be generated automatically by CESM. Thus, it would be desirable to expand CESM such that it can operate in a non-interactive mode as well.

The applicability of the DEVS formalism for building model-based diagnostic units should be investigated, and if successful, these diagnostic units should be subsequently integrated. It is necessary to develop guidelines and/or tools which can help in determining the appropriate number of diagnostic agents. In constructing hierarchical diagnostic units, the number of levels assigned to each of the diagnostic agents should be also decided appropriately.

We discussed the necessity of constraint driven diagnostic units where the consideration of constraints was considered to be important. Two such constraints, the interrogation time and its associated cost for each inquiry, were considered together as one time/cost parameter. A diagnostic unit—an expanded version of CESM—, was implemented that takes time/cost into consideration. It can interrogate a lesser number of measuring devices, and yet conclude reasonably acceptable hypotheses. The usefulness of a constraint driven diagnostic unit becomes important when only a subset of measuring devices can assure a successful failure analysis. In this work, a preliminary study suggested that such diagnostic agents can be beneficial if economical utilization of some commodities (e.g., time and cost) are important.

The constraint driven diagnostic unit as presented in this work, however, requires improvements. Its current search strategy, best-first search, does not ensure an optimal minimization of time/cost. Thus, other search algorithms (e.g., A* or B*) should replace the current algorithm in order to ensure a minimum path to a goal entity. Furthermore, we acknowledged the deficiency of the current termination strategy. Additional research is necessary to determine an appropriate termination strategy as well as how non-monotonic reasoning can lend itself to an enhancement of the constraint driven diagnostic units.

APPENDIX—A

DISCRETE-EVENT MODEL OF A ROBOT

;;; This model includes M-ITP-CONT.

(make-pair atomic-models 'robot)

;;; External Transition Function (i.e., δ_{ext}):

```
(define (ext-robot s e x)
  (case (content-port x)
    ('exp-1      ;;; RECEIVED FROM A USER OR SPACE MANAGER
     (case (content-value x) ;;; Robot assumes that there is adequate
       ('start ;;; amount of supplies and syringes. This
        (case (state-sta-le s) ;;; can be assured by sending a message
          ;;; to rack asking for such information.
          ('flush ;;; Content of the LE chamber is unknown.
           (set! (state-oprt s) 'le)
           (set! (state-oval s) 'empty)
           (hold-in 'contact 5) )
          ('empty ;;; The LE chamber is empty, but not clean!
           (set! (state-oprt s) 'le)
           (set! (state-oval s) 'clean)
           (hold-in 'contact 5) )
          ('clean ;;; The LE chamber is empty and clean.
           (set! (state-oprt s) 'le)
           (set! (state-oval s) 'fill)
           (hold-in 'contact 5) )
          ('full ;;; The LE chamber contains LE liquid.
           (case (state-phase s)
             ('release ;;; The robot has completed the release task
              ;;; successfully; camera is ready to examine
              ;;; the presence of bubbles in the LE chamber.
              (set! (state-oprt s) 'le)
              (set! (state-oval s) 'bubbles?)
              (passivate) ) )
           ('le-full ;;; The LE chamber is filled with LE liquid (no bubbles),
            ;;; thus start by examining the status of the TE chamber.
            (case (state-sta-te s)
              ('flush ;;; The content of the TE chamber is unknown.
               (set! (state-oprt s) 'te)
               (set! (state-oval s) 'empty)
               (hold-in 'contact 5) )
              ('empty ;;; The TE chamber is empty, but not clean.
```



```

    (set! (state-oprt s) 'te)
    (set! (state-oval s) 'clean)
    (hold-in 'contact 5) )
('flush      ;; The TE chamber is empty and clean.
 (set! (state-oprt s) 'te)
 (set! (state-oval s) 'fill)
 (hold-in 'contact 5) )
('full      ;; The TE chamber contains TE liquid.
 (case (state-phase s)
   ('release ;; The has completed successfully the rele-
             ;; ase task; camera is ready to examine the
             ;; presence of bubbles in the TE chamber.
    (set! (state-oprt s) 'te)
    (set! (state-oval s) 'bubbles?)
    (passivate) ) )
 ('te-full  ;; The TE and LE chambers are filled with
             ;; appropriate solutions and no bubbles,
             ;; continue by injecting the SA solution.
    (set! (state-oprt s) 'sa)
    (set! (state-oval s) 'fill)
    (hold-in 'trans-sa 1) ) ) ) ) )
('leitp      ;; RECEIVED FROM ITP
 (case (content-value x)
   ('empty
    (case (state-phase s)
      ('le-empty-low      ;; Receiving the completion
        (set! (state-sta-le s) 'error-empty) ;; time too early.
        (set! (state-oprt s) 'robot)
        (set! (state-oval s) 'le-empty-early)
        (hold-in 'error-early 0) )
      ('le-empty-twind    ;; Receiving the completion
        (set! (state-sta-le s) 'empty)      ;; time as expected.
        (set! (state-oprt s) '())
        (set! (state-oval s) '())
        (hold-in 'release 4) )
      ('error-late        ;; Not receiving the completion
        (set! (state-sta-le s) 'error-empty) ;; time as expected.
        (set! (state-oprt s) 'robot)
        (set! (state-oval s) 'le-empty-late)
        (hold-in 'error 0.5) ) ) )
   ('clean
    (case (state-phase s)
      ('le-clean-low      ;; Receiving the completion time too early.
        (set! (state-sta-le s) 'error-clean)
        (set! (state-oprt s) 'robot)
        (set! (state-oval s) 'le-clean-early)
        (hold-in 'error-early 0) )
    )
  )
)

```

```

('le-clean-twind                                     ;;; Receiving the completion
  (set! (state-sta-le s) 'clean)                   ;;; time as expected.
  (set! (state-oprt s) '())
  (set! (state-oval s) '())
  (hold-in 'release 4) )
('error-late                                         ;;; Not receiving the completion
  (set! (state-sta-le s) 'error-clean) ;;; time as expected
  (set! (state-oprt s) 'robot)
  (set! (state-oval s) 'le-clean-late)
  (hold-in 'error 0.5) ) )
('fill
  (case (state-phase s)
    ('le-fill-low                                     ;;; Receiving the completion time too early.
      (set! (state-sta-le s) 'error-fill)
      (set! (state-oprt s) 'robot)
      (set! (state-oval s) 'le-fill-early)
      (hold-in 'error-early 0) )
    ('le-fill-twind                                   ;;; Receiving the completion time as expected.
      (set! (state-sta-le s) 'full)
      (set! (state-oprt s) '())
      (set! (state-oval s) '())
      (hold-in 'release 4) )
    ('error-late                                     ;;; Not receiving the completion
      (set! (state-sta-le s) 'error-fill)           ;;; time as expected.
      (set! (state-oprt s) 'robot)
      (set! (state-oval s) 'le-fill-late)
      (hold-in 'error 0.5) ) ) )
('teitp                                             ;;; RECEIVED FROM ITP
  (case (content-value x)                           ;;; In the event of receiving inputs
    ('empty                                          ;;; from the ITP wrt the TE chamber,
      (case (state-phase s)                          ;;; the robot follows the same pattern
        ('te-empty-low ... )                        ;;; of logic as in the case of receiving
        ('te-empty-twind ... )                      ;;; it for the LE chamber.
        ('error-late ... ) ) )
    ('clean
      (case (state-phase s)
        ('te-clean-low ... )
        ('te-clean-twind ... )
        ('error-late ... ) ) )
    ('fill
      (case (state-phase s)
        ('te-fill-low ... )
        ('te-fill-twind ... )
        ('error-late ... ) ) )

```

```

('saitp                                     ;; RECEIVED FROM ITP
  (case (content-value x)
    ('fill
      (case (state-phase s)
        ('sa-fill-low                       ;; Receiving the completion time too early .
          (set! (state-sta-sa s) 'error-fill)
          (set! (state-oprt s) 'robot)
          (set! (state-oval s) 'sa-fill-early)
          (hold-in 'error-early 0) )
        ('sa-fill-twind                       ;; Receiving the completion time as
expected.
          (set! (state-sta-sa s) 'sa)
          (set! (state-oprt s) '())
          (set! (state-oval s) '())
          (hold-in 'release 4) )
        ('error-late                         ;; Not receiving the completion
          (set! (state-sta-sa s) 'error-fill) ;; time as expected.
          (set! (state-oprt s) 'robot)
          (set! (state-oval s) 'sa-fill-late)
          (hold-in 'error 0.5) ) ) )
('lecam                                     ;; RECEIVED FROM CAMERA
  (if (< (state-ref-le s) 20)
    (begin                                   ;; If the LE chamber filling process has been repe-
                                           ;; ated less than 20 times, continue the process.
      (case (content-value x)
        (1                                   ;; Bubbles were detected in the LE chamber.
          (set! (state-oprt s) 'le)
          (set! (state-oval s) 'empty)
          (set! (state-sta-le s) 'flush)
          (set! (state-ref-le s) (1+ (state-ref-le s)))
          (hold-in 'contact 5) )
        (0                                   ;; Bubbles were absent in the LE chamber.
          (set! (state-sta-le s) 'le)
          (set! (state-oprt s) 'le)
          (set! (state-oval s) 'flush)
          (set! (state-ref-le s) (state-ref-le s))
          (hold-in 'trans-te 1) ) )
      (BKPT "excess number of refilling " (state-ref-le s) ) )
('tecam                                     ;; RECEIVED FROM CAMERA
  (if (< (state-ref-te s) 20)
    (begin                                   ;; This portion of the code is identical
                                           ;; to the inputs received from the
      (case (content-value x)               ;; camera examining the LE chamber.
        (1 ... )
        (0 ... ) ) ) )
('sacam                                     ;; RECEIVED FROM CAMERA
  (if (< (state-ref-te s) 5)

```

```

(begin
    ;;; If the capillary has been filled less than 5 times,
    ;;; continue the process.
    (case (content-value x)
      (1
        ;;; Bubbles were detected in the capillary.
        (set! (state-oprt s) 'le)
        (set! (state-oval s) 'empty)
        (set! (state-sta-le s) 'flush)
        (set! (state-sta-te s) 'flush)
        (set! (state-ref-sa s) (1+ (state-ref-sa s)))
        (hold-in 'contact 5) )
      (0
        ;;; Bubbles were absent in the capillary
        (set! (state-sta-le s) 'full)
        (set! (state-oprt s) 'turn-on)
        (set! (state-oval s) 'switch)
        (passivate) ) ) )
    (BKPT "Excess number of reruns for the set-up " (state-ref-sa s))
  )))

```

;;; The δ_{int} is responsible for the recurrent operations involved in the
 ;;; set-up procedure and scheduling of the t_{min} and t_{wind} for the
 ;;; primitive operations and tasks.

;;; Internal Transition Function (i.e., δ_{int}):

```

(define (int-robot s)
  (case (state-sta-le s)
    ;;; The LE chamber is under consideration
    ('flush
     ;;; If it's content is unknown, empty it.
     (case (state-phase s)
       ('contact
        ;;; Contact task is completed (refer to
        (set! (state-oprt s) 'le) ;;; figure 3.32), schedule the minimum
        (set! (state-oval s) '()) ;;; time for emptying process.
        (hold-in 'le-empty-low (car (state-e-time-le s) ) ) )
       ('le-empty-low
        ;;;  $t_{min}$  has elapsed, schedule for  $t_{wind}$ .
        (set! (state-oprt s) 'le)
        (set! (state-oval s) '())
        (hold-in 'le-empty-twind (cadr (state-e-time-le s) ) ) )
       ('le-empty-twind
        ;;;  $t_{wind}$  has elapsed, issue an error message.
        (set! (state-sta-le s) 'error-empty)
        (set! (state-oprt s) 'robot)
        (set! (state-oval s) 'le-empty-late)
        (hold-in 'error 0.5) ) ) )
    ('empty
     ;;; If the state of the LE chamber is empty,
     (case (state-phase s)
       ;;; empty, the same pattern is followed
       ('contact
        ;;; as above.

```

```

    (set! (state-oprt s) 'le)
    (set! (state-oval s) '())
    (hold-in 'le-clean-low (car (state-c-time-le s))) )
('le-clean-low
  (set! (state-oprt s) 'le)
  (set! (state-oval s) '())
  (hold-in 'te-clean-twind (cadr (state-e-time-le s))) )
('te-clean-twind
  (set! (state-sta-le s) 'error-clean)
  (set! (state-oprt s) 'robot)
  (set! (state-oval s) 'le-clean-late)
  (hold-in 'error 0.5) ) )
('clean
  (case (state-phase s)
    ;;; If the state of the LE chamber is
    ;;; empty, the same pattern is followed
    ('contact
      ;;; as the one above.
      (set! (state-oprt s) 'le)
      (set! (state-oval s) '())
      (hold-in 'le-fill-low (car (state-e-time-low s))) )
    ('le-fill-low
      (set! (state-oprt s) 'le)
      (set! (state-oval s) '())
      (hold-in 'le-fill-twind (cadr (state-e-time-low s))) )
    ('le-fill-twind
      (set! (state-sta-le s) 'error-fill)
      (set! (state-oprt s) 'robot)
      (set! (state-oval s) 'le-fill-late)
      (hold-in 'error 0.5) ) ) )
('full
  (case (state-phase s)
    ;;; If the state of the LE chamber is
    ;;; empty, the same pattern is followed
    ('release
      ;;; as the one above.
      (set! (state-oprt s) 'le)
      (set! (state-oval s) 'bubbles?)
      (hold-in 'passive 1) ) ) )
(if (eqv? (state-sta-le s) 'le-full)
  (begin
    (case (state-sta-te s)
      ;;; This portion provides logic as
      ;;; applied above when the state of
      ;;; LE chamber was not LE-FULL.
      ('flush
        ;;; Here the only difference is the
        (case (state-phase s)
          ;;;  $t_{min}$  and  $t_{wind}$  used for each of
          ('trans-te
            ;;; the emptying, cleaning, and
            (set! (state-oprt s) 'te)
            ;;; filling.
            (set! (state-oval s) 'empty)
            (hold-in 'contact 5) )
          ('contact
            (set! (state-oprt s) 'te)
            (set! (state-oval s) '())
            (hold-in 'te-empty-low (car (state-e-time-te s))) )
          ('te-empty-low

```

```

      (set! (state-oprt s) 'te)
      (set! (state-oval s) '())
      (hold-in 'te-empty-twind (cadr (state-e-time-te s) ) ) )
    ('te-empty-twind
     (set! (state-sta-te s) 'error-empty)
     (set! (state-oprt s) 'rob)
     (set! (state-oval s) 'te-empty-late)
     (hold-in 'error 0.5) ) ) )
  ('empty
   ;;; Same form of logic is used for
   (case (state-phase s) ;;; empty, clean, full, and trans-sa
     ('trans-te ... ) ;;; with appropriate  $t_{min}$  and  $t_{wind}$ 
     ('contact ... ) ;;; for each of them.
     ('te-clean-low ... )
     ('te-clean-twind ... ) ) )
  ('clean
   (case (state-phase s)
     ('trans-te ... )
     ('contact ... )
     ('te-fill-low ... )
     ('te-fill-twind ... ) ) )
  ('full
   (case (state-phase s)
     ('release
      (set! (state-oprt s) 'te)
      (set! (state-oval s) 'bubbles)
      (hold-in 'passive 1) ) ) )
  ('te
   (case (state-phase s)
     ('trans-te
      (set! (state-oprt s) 'te)
      (set! (state-oval s) '())
      (hold-in 'trans-sa 1) ) ) ) ) )
  (if (and (eqv? (state-sta-le s) 'le-full)
           (eqv? (state-sta-te s) 'te-full))
      (begin
        (case (state-phase s)
          ('trans-sa
           ;;; This portion tests whether both
           ;;; the LE and TE chambers are
           ;;; filled with appropriate liquids.
           (set! (state-oprt s) 'sa) ;;; if indeed the test is passed, the
           (set! (state-oval s) 'fill) ;;; filling of the capillary with the
           (hold-in 'contact 5) ) ;;; sample solution begins. Note
          ('contact
           ;;; that no emptying, or cleaning
           (set! (state-oprt s) 'sa) ;;; takes place.
           (set! (state-oval s) '())
           (hold-in 'sa-fill-low (car (state-f-time-sa) ) ) )
        )
      )
  )

```

```

('sa-fill-low
  (set! (state-oprt s) 'sa)
  (set! (state-oval s) '())
  (hold-in 'sa-fill-twind (cadr (state-f-time-sa) ) ) )
('sa-fill-twind
  (set! (state-sta-sa s) 'error-fill)
  (set! (state-oprt s) 'robot)
  (set! (state-oval s) 'sa-fill-late)
  (hold-in 'error 0.5) )
('release
  (set! (state-oprt s) 'sa)
  (set! (state-oval s) 'bubbles)
  (hold-in 'passive 0.5)
))))

```

;;; The λ is responsible for sending messages to camera and rack.

;;; **Output Function** (i.e., λ):

```

(define (out-robot )
  (case (state-phase s)
    ('() (make-content) )
    ('error
      (set! (state-sigma s) 'inf)
      (make-content 'port (state-oprt s) 'value (state-oval s) ) )
    (else (make-content 'port (state-oprt s) 'value (state-oval s) ) )
  ))

```

REFERENCES

- Aikins, J.** (1983). "Prototypical Knowledge for Expert Systems", *Artificial Intelligence*, **20**, pp. 163-210.
- Berliner, H.** (1979). "The B* Tree Search Algorithm: a Best-First Proof Procedure", *Artificial Intelligence*, **12**, pp. 23-40.
- Bylander, T. et al.** (1985). "CSRL: A Language for Expert Systems for Diagnosis", *Comp. & Maths. with Appls.*, **11**, pp. 449-456.
- Chandrasekaran, B. and Mittal, S.** (1983). "Deep Versus Compiled Knowledge Approaches to Diagnostic Problem-Solving", *Int. J. Man-Machine Studies*, **19**, pp. 425-436.
- Dale-Molle, D.T. and Himmelblau, D.M.** (1987). "Fault Detection in an evaporator via Parameter Estimation in Real Time", *Proceedings of the Second European Workshop on Fault Diagnostics, Reliability and Related Knowledge Based Approaches*, UMIST, Manchester, April 6-8, pp. 131-138.
- David J.M., and Krivine, J.P.** (1987). "Three Artificial Intelligence Issues in Fault Diagnosis: Declarative Programming, Expert systems, and Model-Based Reasoning", *Proceedings of the Second European Workshop on Fault Diagnostics, Reliability and Related Knowledge Based Approaches*, UMIST, Manchester, April 6-8, pp. 19-27.
- Davis, R. et al.,** (1982). "Diagnosis Based on Structure and Function", *Proceedings National Conference on Artificial Intelligence*, Pittsburgh, PA, August, pp. 137-142.
- Davis, R.** (1984). "Diagnostic Reasoning Based on Structure and Behavior", *Artificial Intelligence*, **24**, pp. 347-410.
- deKleer, J. and Brown, J.S.** (1984). "A qualitative Physics Based on Confluences", *Artificial Intelligence*, **24**, pp. 7-83.
- Forbus, K.** (1985). "Qualitative Physics", *SIGART Newsletter* **93**, July, pp. 7-9.
- Genesereth, M.** (1984). "The Use of Design Description in Automated Diagnosis", *Stanford Heuristic Programming Memo*, HPP-81-20, January.

Genesereth, M. and Nilsson N. (1987). *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., pp. 115-159.

Hack, B. (1988). "Man to Machine, Machine to Machine, and Computer to Instrument Interfaces for Teleoperation of a Remote Fluid Handling Laboratory", MS Thesis, Dept. of Electrical & Computer Engineering, University of Arizona, Tucson, AZ, 85721.

Hall, S.B. and Wolbers, H.L. (1984). *The Human Role in Space*, MacDonaldd Douglas Corp., Report MDC H1295, October.

Hart, P.E., et al., (1968). "A Formal Basis of the Heuristic Determination of Minimum Cost paths", *IEEE Transactions on SSC*, 4, pp. 100-107.

Hart, P.E., et al., (1972). "Correction to 'A Formal Basis of the Heuristic Determination of Minimum Cost paths'", *SIGART Newsletters*, 37, Decemeber, pp. 28-29.

Hassan, M.A.M., et al., (1987). "Direct Detection and Identification of Topological Errors in EPS Data Base", *Proceedings of the Second European Workshop on Fault Diagnostics, Reliability and Related Knowledge Based Approaches*, UMIST, Manchester, April 6-8, pp. 181-188.

Hengy, D. and Frank, P.M. (1987). "Component Failure Detection Using Local Second-Order Observers", *Proceedings of the Second European Workshop on Fault Diagnostics, Reliability and Related Knowledge Based Approaches*, UMIST, Manchester, April 6-8, pp. 147-154.

Kelly, M. (1989). "Intelligent Space Station Laboratory Organization Design Using System Entity Structure Concepts", MS Thesis, Dept. of Electrical & Computer Engineering, University of Arizona, Tucson, AZ, 85721.

Mansour, A.E., and Nour-Eldin, H.A. (1987). "Hierarchical and Fast Recursive State Estimation with Robust Bad Data Pre-Cleaning for Large-Scale Power Systems", *Proceedings of the Second European Workshop on Fault Diagnostics, Reliability and Related Knowledge Based Approaches*, UMIST, Manchester, April 6-8, pp. 173-180.

McKinley, S. E. (1989). "C³I for Drug Interdiction", *Signal, AFCEA's Int. J. of C⁴I*, 43, August, pp. 69-70.

Meystel, A. and Luh, J.Y.S. Eds., (1987). *IEEE International Symposium on Intelligent Control*, New York, NY: IEEE press.

Ören, T.I., (1987). "Taxonomy of Simulation Model Processing", *In Encyclopedia of Systems and Control*, M. Signh, Ed. New York, NY: Pergamon Press.

Patton, R.J. and Willcox, S.W. (1987). "A Robust Method for Fault Diagnosis Using Parity Space Eigenstructure Assignment", *Proceedings of the Second European Workshop on Fault Diagnostics, Reliability and Related Knowledge Based Approaches*, UMIST, Manchester, April 6-8, pp. 155-163.

Ribbens, W.B. (1988). "Failure Diagnosis for Automotive Applications", *18th Int. Symposium on Automotive Technology and Automation*, Florence, Italy, June.

Rich, E. (1983). *Artificial Intelligence*, McGraw-Hill, Inc., New York, NY.

Reiger, C.R. and Grinberg, M. (1977). "The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms", *5th IJCAI*, Cambridge Massachusetts, pp. 250-256.

Saridis, G.N. (1977). *Self-Organizing Control of Stochastic Systems*, Marcel Dekker, Inc.

Saridis, G.N. (1979). "Toward the Realization of Intelligent Controls", *Proceedings of the IEEE*, 67, August, pp. 1115-1133.

Saridis, G.N. (1983). "Intelligent Robotic Control", *IEEE Transactions on Automatic Control*, 28, May, pp. 547-557.

Scherer W.T., and White, C.C. (1987). "A Survey of Expert Systems for Equipment Maintenance and Diagnosis", *Proceedings of the Second European Workshop on Fault Diagnostics, Reliability and Related Knowledge Based Approaches*, UMIST, Manchester, April 6-8, pp. 3-18.

Schooley L. C. and Cellier F.E. (1989). "Telescience: Remote Interaction With Scientific Experiments", *Proceedings of CIPS Conference on Telepresence and Remote Interaction*, October 30—

November 1, Edmonton, Alberta, Canada: Canadian Information Processing Society.

Steels, L. (1986). "Second generation Expert Systems", *Proceedings of the Sixth Annual Technical Conference of British Computer Society Specialist Group on Expert Systems*, Brighton, 15-18 December, pp.175-183.

Texas Instruments, Inc. (1985). TI-Scheme: Language Reference Manual, Texas Instruments, Dallas, TX.

Thormann, W. (1984). "Principles of Isotachophoresis and Dynamics of the Isotachophoresis Separation of Two Components", *Separation Science and Technology*, **19**, pp. 455-467.

Tzafestas, S. and Skolarikos, M. (1987). "On the Sensor Fault Detection of Large Scale Systems Using the Overlapping Decomposition Approach", *Proceedings of the Second European Workshop on Fault Diagnostics, Reliability and Related Knowledge Based Approaches*, UMIST, Manchester, April 6-8, pp. 131-1129.

Wang, Q. (1989). "Management of Continuous System Models in DEVS-Scheme", MS Thesis, Dept. of Electrical & Computer Engineering, University of Arizona, Tucson, AZ, 85721.

Webster's Dictionary. (1984), USA.

Wymore, A.W. (1988). "A Mathematical Theory of System Design", unpublished manuscript.

Zeigler, B.P. (1976). *Theory of Modelling and Simulation*, New York, NY:Wiley, (reissued by Krieger Pub. Co., Malabar, FL, 1985).

Zeigler, B.P. (1984). *Multifaceted Modelling and Discrete Event Simulation*, London and Orlando, FL: Academic Press.

Zeigler, B.P. (1986). DEVS-Scheme: A Lisp-Based Environment for Hierarchical, Modular Discrete Event Models, Tech. Rep. AIS-2, CERL Lab., Dept. of ECE, Univ. of Arizona, Tucson, AZ 85721.

Zeigler, B.P. (1987-a). "Hierarchical, Modular Discrete Event Modelling in an Object Oriented Environment", *Simulation*, **49** (5), pp. 219-230.

Zeigler, B.P. (1987-b). CESM: Classification Expert System Management, CERL Lab., Dept. of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721.

Zeigler, B.P., et al., (1988). "Design of a Simulation Environment for Laboratory Management by Robot Organizations", *J. of Intelligent Robotic Systems*, 1, pp. 299-309.

Zeigler, B.P. (1989). "The DEVS Formalism: Event-Based Control for Intelligent Systems", *Special Issue of Proceedings of IEEE*, January, pp. 72-80.