Hierarchical Controllers and Diagnostic Units for Semi-Autonomous Teleoperation of a Fluid Handling Laboratory (*)

Hessam S. Sarjoughian François E. Cellier Bernard P. Zeigler

Department of Electrical and Computer Engineering
University of Arizona

Abstract

This paper discusses the utilization of hierarchical diagnosers for hierarchical event-based control systems for robot-controlled experiments aboard the forthcoming Space Station Freedom. The analysis was performed using discrete-event simulation. The hierarchical discrete-event model was implemented in DEVS/Scheme. The concepts are exemplified at hand of a general-purpose fluid handling facility which has been proposed to be included as one component of the Life Science Module. The proposed methodology will provide the laboratory robot with sufficient intelligence that it can perform most of the needed tasks in a semi-autonomous mode. Experiments can be remotely guided by the scientific investigator from the ground. Thereby, the need for crew interaction is minimized.

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 (currently more than $30,000 per hour) 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. It is proposed that a hierarchical control scheme accompanied by hierarchical diagnostic units provides the most appropriate control structure for this enterprise.

(*) The research described in this paper was supported by NASA-Ames Cooperative Agreement # NCC 2-525.

The Laboratory Setup

The SSF will host a Life Science Module (LSM) which can accommodate various types of laboratory experiments related to Space medicine, gravitational biology, genetics, and biochemistry. It has been proposed that the LSM 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 [1]. This multi-purpose instrument can support a large set of different experiments in related disciplines. In this paper, we shall analyze one of its component, namely the Fluid Handling Laboratory (FHL) facility. In the context of the FHL, we shall analyze (simulate) the behavior of one particular instrument, the Isotachophoresis (ITP) instrument which decomposes fluids into their charged components [2]. Since there must not exist any air/liquid interfaces under microgravity conditions (except for those that are controlled by surface tension, i.e., within a capillary), all fluid containers must always be full. This is achieved by storing fluids in doubly-contained containers. An inflatable bag is contained within an aluminum bottle. The space between the bag and the bottle wall is pressurized. Thereby, the bag is constantly squeezed and reduced to minimum volume. An air-tight septum seals the bag. Liquid is injected into (extracted from) the bag through the septum by means of a syringe needle. We shall call this device a Pressurized Bladder Bottle (PBB).

We shall assume an event-based controller [3] for the operations filling, emptying, and cleaning necessary in the setup procedure of the ITP. Furthermore, we assume that there exists a lower level conventional controller that is supervised by the high-level event-based controller.

This control scheme is accompanied by an equivalent hierarchical diagnoser that enables the discovery of and recovery from anomalous conditions that may occur during the semi-autonomous execution of a scientific experiment.

Control System Configuration

Figure 1 shows the overall configuration of our experimental control system.

The plant (the robot controlled isotachopheresis apparatus - ITP) is controlled by means of a conventional controller which is responsible for smooth and accurate motion of the robot. The controller uses a model-reference adaptive control strategy. It consults with a model of the real plant in order to update the controller parameters. It is supported by a model-based diagnoser which operates on another model of the real plant in order to determine discrepancies between the controller's perception of how the system components work and the true performance of the actual plant. Both the controller and the diagnoser are supported by higher-level control/diagnostic units. While the low-level units are responsible for control/diagnosis of individual components, the high-level units are responsible for control/diagnosis of the overall system. Each of these units also consults with a (coarser) model of the plant.

Our experimental setup looks somewhat similar to a subset of the NASREM telerobot architecture [4], except that the models used by the controllers and diagnosers were separated and drawn on the outside rather than as a center "world model" column.

Each of the functional blocks is implemented as an independent intelligent agent that communicates with the neighboring intelligent agents through a message passing mechanism.

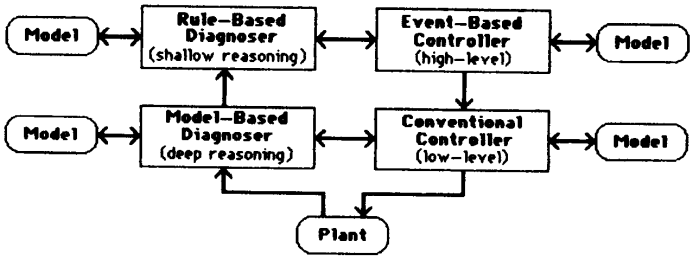

Figure 1: Control System Configuration

Modeling Methodology

In our experimental setup, both the real plant and the control/diagnostic agents are simulated using discrete-event simulation [5]. For this purpose, the lowest level (most refined) continuously changing state variables including their local controllers were aggregated, and are therefore no longer represented in our models. In particular, smooth motion control was not one of our immediate concerns, and therefore, is not represented in our simulated laboratory setup.

In representing a process through a model, the underlying objectives of that model play an important role in the model construction [5,6]. Therefore, there may exist various types of models of a single system, each devoted to a specific purpose.

In our testbed, we used three different models of the ITP device:

(1) We used an *operational model* which captures the behavior of the real plant. The operational model (M-ITP-OPER) is sometimes referred to as the *external* model, since it is not part of the intelligent controller.
(2) We also used a *controller model* (M-ITP-CONT) which is consulted by the event-based controller (the conventional controller does not currently exist, but has been lumped together with the plant).
(3) We finally made use of a *diagnostic model* (M-ITP-DIAG) which is consulted both by the high-level and low-level diagnosers. Both M-ITP-CONT and M-ITP-DIAG are referred to as *internal* models since they are parts of the intelligent controller.

Notice that we must ensure the integrity of our system by enforcing the consistency among the various models used in our control scheme. This is achieved by making use of a *master model* (MB-ITP). All model modifications are performed on that master model. The various models used in our control scheme are then automatically generated from the master model by means of *model pruning*. Figure 2 shows the hierarchical pruning of the three models from the master model using a three-level abstraction.

In practice, it is not useful to store the master model in the data base in a compact form. A more amenable approach is to store the master model as a *System Entity Structure* [6] which describes a hierarchical decomposition of the overall system into its components together with a library of discrete-event models that describe the leaf components of the hierarchical decomposition tree. However, this methodology will not be described in this paper. It is the topic of a separate publication [7].

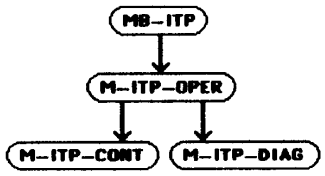

Figure 2: Hierarchical pruning of models

The modeling/simulation environment used in our research is DEVS/Scheme [8], a modeling/simulation environment designed specifically for management of hierarchies of discrete-event models. The overall modeling/simulation task of which the here described research is a part has been recently reported [9].

Diagnostic Process

The diagnostic process is defined as the recognition of diseases or faults and their causes. The overall diagnostic process can be decomposed into three separate milestones:

(1) Anomaly detection,
(2) Hypothesis generation, and
(3) Hypothesis testing and discrimination.

A fault diagnoser remains a *passive observer* of its system until it detects an anomaly. For this purpose, the diagnoser assumes that its model of the system is correct, and looks for discrepancies between the behavior of its model and the behavior of the real system. If such a discrepancy has been detected, it assumes that the system has undergone some change which is considered a fault or disease. This portion of the fault diagnoser is sometimes referred to as a *watchdog monitor*.

Once an anomaly has been detected, the diagnoser switches from a passive to an active mode. Its next task must be to come up with a set of hypotheses as to what might have gone wrong. It can do this by

shallow reasoning, i.e., using a rule-based approach. In this scenario, the diagnoser will traverse a cause-effect matrix in reverse direction. The anomaly is treated as an effect (symptom), and the diagnoser checks which causes could possibly be made responsible for the observed symptom. Alternatively, it can do this by *deep reasoning,* i.e., using a model-based approach. In this scenario, the diagnoser will make use of a system identification procedure to modify the parameters of its model until the model matches the unexpected observed behavior of the system. Each set of parameters that produces a good match with the observed behavior can then be used to generate one or several hypotheses. Shallow reasoning has the advantages of being generally faster and of aiming directly at potential causes of the observed anomaly, while deep reasoning uses an indirect approach to determine potential causes. Deep reasoning has the advantage of being theoretically able to even recognize unforeseen causes, whereas shallow reasoning cannot hypothesize beyond the scope of the previously determined set of potential problems which are hard-coded into the cause-effect matrix.

The next step in the diagnostic process is hypothesis testing. Each proposed hypothesis is usually associated with a series of symptoms. We shall thus proceed by taking additional measurements to check whether other symptoms than the one that was initially observed are also present in the system. If an essential associated symptom is missing, the tested hypothesis can be rejected. In shallow reasoning, we traverse the same cause-effect matrix in forward direction to determine other effects of the postulated cause, and test the system for presence/absence of these additional effects. In deep reasoning, we use a (usually precoded) model of the postulated defective system, and compare the trajectory behavior of this fault model with the behavior of the real system.

It may be that this approach reduces the number of hypotheses to one. If this is not the case, the diagnoser can resort to a mode of actively probing the real system by interspersing its own test signals to discriminate among the remaining hypotheses.

Rule-based (shallow) reasoners find single faults usually faster than model-based (deep) reasoners. However, the fault tree (a refined hierarchical version of the cause-effect matrix) grows exponentially with the number of simultaneous faults being considered. This is not the case with model-based (deep) reasoners [10]. Also, deep reasoners are much less vulnerable to missing data than shallow reasoners. They are therefore more robust. Consequently, both types of reasoners have their place, and a combination of the two may often be the best solution.

Generally, faults can be categorized into two types: *persistent* (component) faults, and *intermittent* (communication) faults. Once a persistent fault has occurred, it remains in the system until actively removed. Intermittent (sporadic) faults come and go without external intrusion. This research was concerned with the diagnosis of persistent faults exclusively.

Hierarchical Diagnostic Units

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 intended type of controller.

We prefer to envision a hierarchy of diagnostic units. In the reported research, the hierarchy consists of two levels only, *high* and *low,* accompanying the high-level (event-based) and the low-level (conventional) controllers. Thus, we have considered both a high-level and a low-level diagnoser. This means that for every *intelligent controller* there exists an *intelligent diagnoser,* and for every *conventional controller* there exists a *conventional diagnoser.*

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 if both a low-level 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.

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 should be utilized for further diagnosis. It should be

emphasized that the categorization of failures into levels is highly application dependent.

The high-level/low-level knowledge representations which are utilized by their corresponding diagnostic units determine the class of a diagnostic unit (*high* or *low*). Knowledge bases are used to store various types of information about the system and its components. 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 [11].

Classification Expert System Maker

An appropriate expert system is able to provide the medium for shallow-reasoning. Thus, an expert system is considered to be the most suitable candidate for the development of a high-level (intelligent) diagnostic unit.

The Classification Expert System Maker (CESM) is an expert system shell which is designed for developing *classification-based* expert systems [12]. This expert system shell has been coded in PC-Scheme [13] which supports the *object oriented programming paradigm* (SCOOPS). Since DEVS/Scheme [8] has also been implemented in PC-Scheme, 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 played a significant role in selecting CESM as our expert system shell.

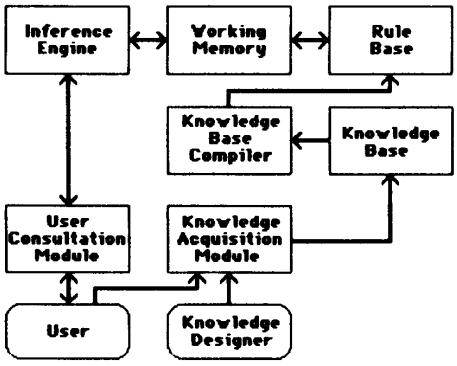

Figure 3: Architecture of CESM

The structure of CESM is shown in Figure 3.

CESM supports several important features:

- system classification models built on the taxonomy entity structure,
- automatic compilation of rules into a knowledge base,
- a straightforward interfacing with the underlying environment (PC-Scheme),
- expressions and function descriptions in addition to synonyms and anti-synonyms, and
- uncertainty handling using qualitative words (e.g., *usually,* and *rarely*), and evidence accumulation utilizing the Dempster-Shafer approach [14].

The data types and data values for CESM are as follows: *predicate* and *class* (i.e., entity) are two primary types of data where a predicate is used to describe a property of a class, and the class plays the role of an unknown class in the inference engine. Permissible values for a predicate are '*yes*', '*maybe yes*', '*no*', '*maybe no*', and '*unknown*.' 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:

$$0.0 \leq (EF, EA, N, X) \leq 1.0$$
$$EF + EA + N + X = 1.0.$$

The evidential status values are as follows:

CEF:	Conclusive Evidence For	→ (1, 0, 0, 0)
EF:	Evidence For	→ (EF, 0, (1-EF), 0)
CEA:	Conclusive Evidence Against	→ (0, 1, 0, 0)
EA:	Evidence Against	→ (0, EA, (1-EA), 0)
N:	Neutral Evidence	→ (0, 0, 1, 0)
X:	Inconsistent	→ (0, 0, 0, 1)

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. Since the predicate and

the class have different data values (e.g., *maybe yes* for the predicate, and *EF* for the class), the inference engine makes them compatible by equating them as follows:

yes	≡	(1, 0, 0)
maybe yes	≡	(0.5, 0, 0.5)
no	≡	(0, 1, 0)
maybe no	≡	(0, 0.5, 0.5)
unknown	≡	(0, 0, 1)

Knowledge Representation in CESM

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 as shown e.g. in Figure 4. There exist semantic relations among predicates which can 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 entities. 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).

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*) for a problem, which is then automatically translated into a set of rules by the knowledge base compiler.

The ITP Experiment

In order to monitor the volume of the liquid inside the pressurized bladder bottles, each of them has been outfitted with *air pressure sensors*. We assume that there are two identical threshold type air pressure sensors: a *primary* sensor, and a *backup* sensor. 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 primary sensors are represented in M-ITP-OPER, and the backup sensors are represented in M-ITP-DIAG.

Although the diagnoser unit must represent the backup air pressure sensors, it may not be necessary to include the same level of complexity in the M-ITP-DIAG as in 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.

Figure 4 depicts the faults (classes or entities) and related symptoms (predicates) that should be examined for determining the cause(s) of a failure in setting up the ITP.

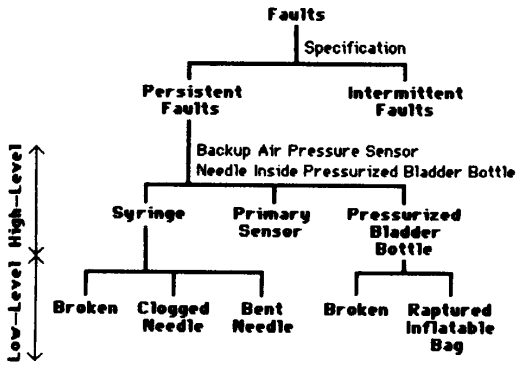

Figure 4: A fault-tree for the ITP experiment.

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*'). Although, all the sensors in this study are assumed to have threshold type characteristics (i.e., *binary logic*), we would presume that their responses are of the predicate data type defined earlier.

Figure 4 shows how, in our FHL example, faults can be partitioned into general (i.e., high-level), and specific

(i.e., low-level). Faults of the *pressurized bladder bottle,* the *syringe,* and the *primary air pressure sensor* without further qualification are classified as high-level causes. Once a high-level diagnoser has determined the possible high-level cause(s) of a fault, 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 fault. Characterized as low-level causes were: *broken syringe, clogged needle, bent needle, ruptured inflatable bag,* and *broken bottle.*

Figure 5 illustrates how a fault-tree may utilize binary logic in order to identify the cause(s) of a failure.

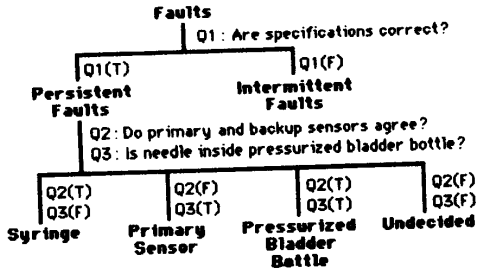

Figure 5: Part of the fault-tree given in Figure 4; binary logic is utilized in determining the high-level failures.

The responses of the backup sensors are not sufficient for a unique discrimination between all high-level cause(s) of a fault. Consequently, we shall need additional sources of information (e.g. the response of a vision sensor) in order to discriminate between the high-level cause(s) of a failure.

From this fault-tree, it cannot be determined conclusively which of the three components is responsible for a hardware failure, it only indicates which is the most likely candidate depending on the received responses for questions Q2 and Q3. Moreover, there is an exception: If the responses to both questions Q2 and Q3 are *false* (F), we cannot conclude which of the components is the most likely culprit. The predicates are expressed as sentences which are equivalent to the rules compiled by the *knowledge base compiler* component of CESM.

Once the faulty component has been determined, it may be necessary to locate the low-level cause(s) through a more thorough analysis. This can be achieved by the model-based low-level diagnoser which will analyze the behavior of the faulty component only, but will do so with a much finer granularity.

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.

For instance, if a perforated inflatable bag is replaced with a new one, and shortly thereafter, the new bag is perforated again, the problem may be that the true cause which was responsible for the damage of the inflatable bag had not been removed (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 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 certainly non-trivial.

The fault-tree can be 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, 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 as replacements for the real system since our diagnoser is not currently connected to a real process with real sensory devices.

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. 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.

Summary

Our goal has been to demonstrate that 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 and/or justifiable. 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.

References

[1] **Schooley L. C. and Collier F. E. (1989).** Telescience Testbed Pilot Program, Final Report, USRA Grant, Tech. Report TSL-021/88, Dept. of Electrical Engr., University of Arizona, Tucson, AZ 85721.

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

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

[4] **Albus, J. S., McCain, H. G., and Lumia, R. (1987).** NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM), NBS Technical Note 1235, Robot System Division, Center for Manufacturing Engineering, National Technical Information Services, Gaithersburg, MD.

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

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

[7] **Wang, Q., Collier, F. E. and Zeigler, B. P. (1990).** "A Five-Level Hierarchy for the Manipulation of Simulation Models", in preparation.

[8] **Zeigler, B. P. (1989).** " DEVS Representation of Dynamical Systems: Event-Based Intelligent Control", *Proceedings of the IEEE*, **77**(1), pp. 72–80.

[9] **Zeigler, B. P., Collier, F. E., and Rozenblit, J. W. (1988).** "Design of a Simulation Environment for Laboratory Management by Robot Organizations", *J. Intelligent and Robotic Systems*, **1**, pp. 299–309.

[10] **Searl, E. (1989).** "Sensor Failure and Missing Data: Further Inducements for Reasoning with Models", personal communication.

[11] **Chandrasekaran, B. and Mittal, S. (1983).** "Deep Versus Compiled Knowledge Approaches to Diagnostic Problem-Solving", *Int. J. Man-Machine Studies*, **19**, pp. 425–436.

[12] **Zeigler, B.P. (1987).** CESM: Classification Expert System Management, Tech. Rep., CERL Lab., Dept. of ECE, Univ. of Arizona, Tucson, AZ 85721.

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

[14] **Zeigler, B. P. (1988).** "Some Properties of Modified Dempster-Schafer Uncertainty Management Operators in Rule-Based Inference Systems", *Inter. J. General Systems*, **14**(4), pp. 345–356.