Report Number: WUCS-90-16

1990-04-01

MOBAD Model-Based Diagnosis

Amy Brodbeck, Paul Calabrese, Joanna Liu, and Mark Maxwell

Troubleshooting measurement equipment can be complex and time consuming task. We developed a system that incorporates model-based diagnosis to troubleshoot measurement instrumentation systems. It consists of a knowledge representation scheme for modeling the structure and behavior of measurement systems and the ability to reason from first principles about these models. It accepts observed values from the user and returns a list of components that are suspected of failing. The current system was developed using the object oriented approach and was designed to be reusable for a variety of measurement systems. We discuss our expansion of Randy Davis' approach to... **Read complete abstract on page 2.**

MOBAD Model-Based Diagnosis

Amy Brodbeck, Paul Calabrese, Joanna Liu, and Mark Maxwell

Complete Abstract:

Troubleshooting measurement equipment can be complex and time consuming task. We developed a system that incorporates model-based diagnosis to troubleshoot measurement instrumentation systems. It consists of a knowledge representation scheme for modeling the structure and behavior of measurement systems and the ability to reason from first principles about these models. It accepts observed values from the user and returns a list of components that are suspected of failing. The current system was developed using the object oriented approach and was designed to be reusable for a variety of measurement systems. We discuss our expansion of Randy Davis' approach to model-based diagnosis to include instrumentation systems.

MOBAD
Model-Based Diagnosis

Amy Brodbeck, Paul Calabrese,
Joanna Liu and Mark Maxwell

WUCS-90-16

April 1990

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

CS 513 Final Report

TABLE OF CONTENTS

ABSTRACT

Troubleshooting measurement equipment can be a complex and time consuming task. We developed a system that incorporates model-based diagnosis to troubleshoot measurement instrumentation systems. It consists of a knowledge representation scheme for modeling the structure and behavior of measurement systems and the ability to reason from first principles about these models. It accepts observed values from the user and returns a list of components that are suspected of failing. The current system was developed using the object oriented approach and was designed to be reusable for a variety of measurement systems. We discuss our expansion of Randy Davis' approach to model-based diagnosis to include instrumentation systems.

1. INTRODUCTION

Measurement systems in industry, whether electrical or physical, are increasing in complexity as greater accuracy and new measurement parameters are required. Computer controlled testing is common, although manually operated systems are still dominant in many laboratories. One trait shared by all measurement systems is that the individual components of these systems are subject to failure. A characteristic of complex systems is that component failures are often difficult to diagnose.

Currently, troubleshooting of measurement systems is done by engineers and technicians. The experience of the individual plays a crucial role in diagnosing a failed system; thorough knowledge of the measurement system and component functions is required. Capturing the experts special knowledge in an expert system means that the measurement laboratory will have the capability to perform equipment diagnosis even if a knowledgeable individual is not available. In addition, engineers and technicians will have a tool to aid them in performing their tasks.

We have developed a system that utilizes model-based diagnosis to troubleshoot measurement instrumentation systems. Our diagnosis system consists of a knowledge representation scheme to model the structure and behavior of measurement systems and is able to reason about this knowledge from first principles. Our system has the following features:

- Models the structure and behavior of both physical and electrical measurement systems
- Accepts observed behavior from the user
- Applies the process of hypothesis generation and testing to determine which measurement system component(s) might be misbehaving
- Provides a graphical interface for the user to enter information and to view the structure of a measurement system

This project has concentrated on troubleshooting systems composed of devices such as instruments (e.g. voltmeters, frequency counters, digital pressure gages), transducers (e.g. load cells, accelerometers), and the connecting wiring, plumbing, valves, etc. The detailed internal structure of each system component is not normally modeled, although this is not strictly the case; some instruments can be described best in terms of their internal structures. The task of the diagnosis system is to determine what part of the measurement system deviates from correct operation given the observed symptoms.

Similar work has been done in the domain of digital electronics (Davis, 1984) and for more general domains (Genesereth, 1984). This project is distinctive because we have modeled and diagnosed systems composed of not only digital devices, but also analog devices.

2. BACKGROUND AND SIGNIFICANCE

Work with diagnosis has been done in a variety of domains for many years. MYCIN is a rule-based system developed in the mid-70s to perform medical diagnosis (Shortliffe, 1976). Other types of diagnosis systems include decision trees, diagnostics, and fault dictionaries. These four approaches all have drawbacks which make them less useful than model-based diagnosis, as discussed by Davis and Hamscher (Davis and Hamscher, 1988). Briefly, rule-based systems are very device specific and depend on having an expert; model-based systems can troubleshoot any system for which a model can be built. Decision trees simply guide the diagnosis of a system through a predefined tree of conditions leading to a conclusion. Decision trees suffer from the same basic drawbacks as rule-based systems. Diagnostics systems actually do verification rather than troubleshooting; they make sure a device performs according to specifications rather than reasoning from observed misbehavior to find the bad component. Fault dictionary systems access a predetermined set of faults to see if the observed symptoms can be explained by a listed fault; model-based systems reason from first principles and may discover a fault not previously encountered.

This project is most closely related to the work done by Davis (Davis, 1984 and Davis and Hamscher, 1988) and Genesereth (Genesereth, 1984). The following paragraphs summarize the topics in their work that we found most relevant to our own.

Davis did much to develop the theory of reasoning about knowledge of structure and behavior (Davis, 1983). In his work he reported the development of languages to describe structure and behavior separately. Davis and Hamscher discussed the common themes in representing structure and behavior, as follows (Davis and Hamscher, 1988):

- Structure representation should be hierarchical.
- Structure representation should be object centered and isomorphic to the organization of the device.

- Behavior can be represented by a set of expressions that capture the interrelationships among the values on the terminals of the device.

Davis developed the process of hypothesis generation, testing, and discrimination (Davis, 1984 and Davis and Hamscher, 1988). He described these three steps as follows:

- Hypothesis generation: Given one discrepancy, which of the components in the device might have produced it?

- Hypothesis testing: Given a collection of components implicated during hypothesis generation, which of them could have failed so as to account for all available observations of behavior?

- Hypothesis discrimination: When, as is almost inevitable, more than one hypothesis survives the testing phase, what additional information should be gathered to discriminate among them?

In both Davis'(Davis, 1984) and Genesereth's (Genesereth, 1984) works, hypothesis generation and testing are combined. Davis came up with a clever way of testing hypotheses called constraint suspension. Constraint suspension is a way of testing suspects to determine whether a failure in the suspect is consistent with the observed behavior of the system (i.e. the inputs and outputs).

Hypothesis discrimination involves gathering additional information, as mentioned above. Additional information can be in the form of probing places in the structure not already tested, or by testing, which means to change the inputs and observe the behavior of the system again. Various methods of determining where to probe and what tests to perform are discussed in (Davis and Hamscher, 1988).

Davis develops the concept of a layered set of categories of failure. This concept is significant in that it facilitates progressive broadening of the categories of failure that can be explored by the system, thus limiting the search space at each level. The most limited category of failure is localized failure of function (i.e. misbehaving system component). For this category, a solder bridge between two pins of a chip would not be correctly diagnosed since a bridge changes the real structure from that of the model. Davis produced the following list of categories:

1) Localized failure of function
2) Bridges
3) Unexpected direction (for component ports with only one direction of communication)
4) Multiple point of failure
5) Intermittent error
6) Assembly error
7) Design error

Genesereth developed a diagnostic program called DART that models structure and behavior with predicate logic axioms (Genesereth,1984). Davis and Hamscher summarize the system as follows: "DART views diagnosis as a form of constrained theory formation. Starting with a set of observations of device misbehavior, the goal is to produce a description of its (faulty) structure. . . [To arrive at deductions] the system uses a technique called resolution residue, a variation on resolution that works as a direct proof procedure (rather than a refutation method), guided by a number of strategies like unit preference for reducing the number of useless deductions." (Davis and Hamscher, 1988)

Finally, one other system of interest is the GDE system developed by de Kleer and Williams (de Kleer and Williams, 1987). Their system can generate both single and multiple fault hypothesis using an assumption-based truth maintenance system (ATMS). This has shown to be an efficient method of hypothesis generation.

3. APPROACH

Using constraint based reasoning for hypothesis generation and testing, we designed and built an object oriented diagnosis system. The decision to use a hierarchical object oriented representation allowed us to represent devices realistically and to implement simulation efficiently. Class inheritance enabled sharing of attributes and procedures among our components and ports. Object oriented programming also allowed us to represent device behavior through expressions, which capture the interrelationships among the values on the terminals of the device. The components and ports in our model communicate with each other through their methods.

Hypothesis testing was implemented using constraint suspension. The values associated with the ports of a device are related to one another via the constraints of the device. For example, a properly working wire with a known voltage at one end will constrain the value of the other end to the same voltage. When the constraints of a component are suspended, the values of its ports are not propagated by that component during simulation. When a component is suspended and all the system's predicted values are the same as (consistent with) the observed values, then the suspended component becomes a suspect.

Gathering suspects is called hypothesis generation and hypothesis testing. During the diagnosis process, each of the suspected components are collected and each one is considered a hypothesis. These hypotheses must be evaluated to determine which is the faulty component.

The system was implemented incrementally. Initially, a simple one-level model was developed to test the simulation methods. This was followed by a multiple-level model for the prototype. Then the diagnosis process was implemented and finally, the user interface was constructed. In general, we developed iteratively throughout the process, constantly revising and rewriting. Using this methodology, the model was refined and generalized resulting in a more efficient program.

Modeling multiple layers of components was difficult. For example, one component may be comprised of several subcomponents. This means that a component and a subcomponent usually share one or more ports. We had to solve the complex problem of simulating and diagnosing multiple layer systems. Our decision was to simulate at all levels. During simulation the ports and components communicate to each other all the way down to the lowest level. However, during diagnosis components are suspended at the top level. When a suspect is encountered, diagnosis continues at the next level down within that component, and so forth.

Due to the short duration of this project, we decided to reduce the scope of our efforts. Our main goal was diagnosis through hypothesis testing. We determined that hypothesis discrimination would be too time consuming, therefore it was eliminated. We also decided to assume a single localized failure of function. We believe that most failures fall into this category.

4. TOOL SELECTION

After researching other model-based reasoning applications, it was evident that an object oriented tool would be ideal for simulation and diagnosis. Reasoning about a model can be achieved at a high level of abstraction. Our main candidates included C++, Smalltalk/V, and PCL (Portable Common LOOPS). C++ was eliminated because it would require a lot of time to learn and use. After investigating Smalltalk/V and PCL, we determined that both tools appeared to be adequate. We chose Smalltalk/V because of its rapid prototyping capabilities and our curiosity about the language.

5. IMPLEMENTATION

The following sections describe the details of MoBaD's implementation as they pertain to the structure, behavior, diagnosis, and user interface, respectively.

5.1 STRUCTURE

The structure of a system being modeled in MoBaD is represented in the way that the various objects are interconnected. These objects represent the individual devices or components of the system and the connections between these components. Each connection between two components is represented by a port object, which contains all of the values necessary for that connection. For example, a port that corresponds to an electrical connection might contain values for the voltage, frequency and phase of the signal at that point.

Each component object contains either a method that describes its behavior or a list of its subcomponents. The method that determines a device's behavior uses the known values of its ports to compute the values associated with its other ports, if possible. When a list of subcomponents is included in the component, they must also contain either subcomponents or descriptions of their behavior. Therefore,

the behavior of each component must be described either directly or in terms of its subcomponents.

Each component also contains a list of its ports and each port contains a list of the components that share it. In this way, the interconnection of the objects corresponds to the actual structure of the system being tested (Figure 5.1-1). The structure of this model also corresponds closely to the mental models used by human engineers in such applications. They tend to have a hierarchical model of devices that are described in terms of their behavior and interconnections that have values or sets of values associated with them.

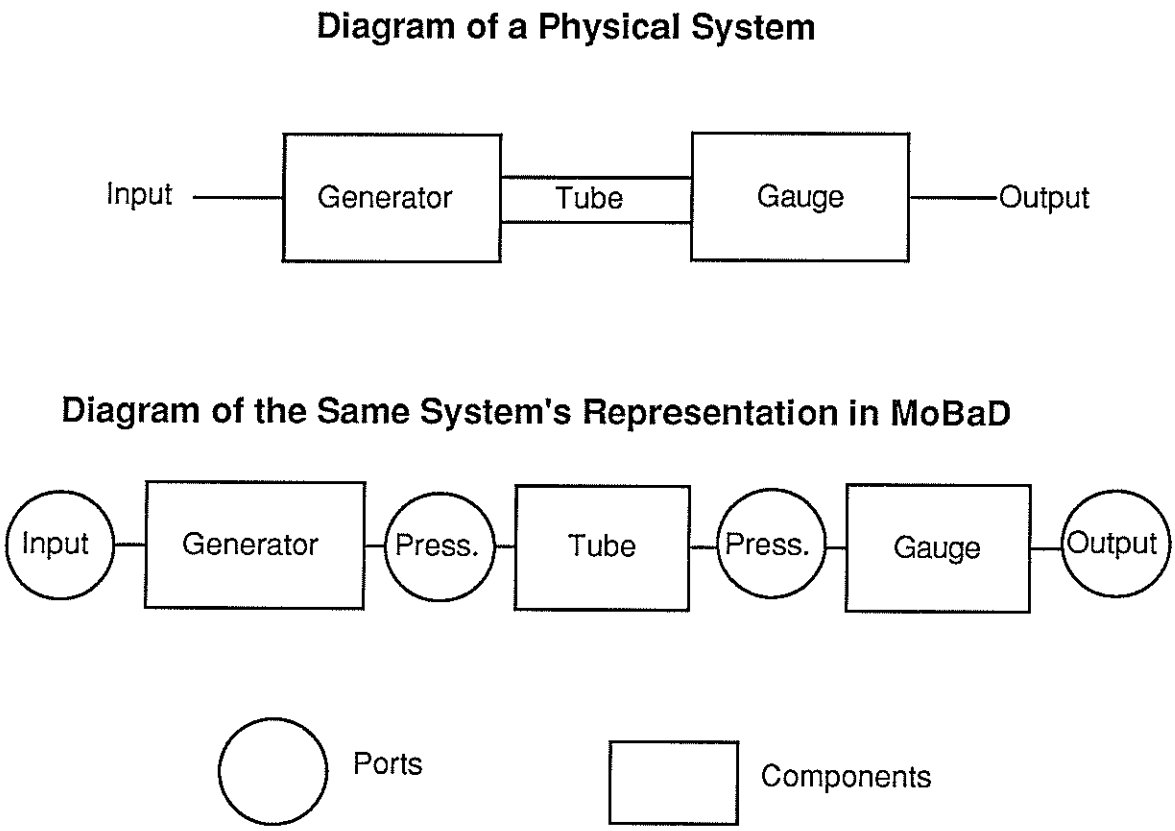

Figure 5.1-1 Diagram of System Representation

5.2 BEHAVIOR

The behavior of the simulation in MoBaD closely follows the behavior of the actual devices. When no constraints are suspended (normal simulation) the values of the ports are propagated from the inputs of the system towards the outputs. A value moves from an input port into a component, where its effect is calculated for the other ports of that component. As these other ports are updated, their values are propagated to other components which in turn update their other ports, and so on. This movement of values from inputs towards outputs and from

component to component closely matches the behavior of the devices being modeled. When a component's constraints are suspended, the simulation process may work backwards from known output values towards the inputs. This is similar to the way a person infers that 5 volts exists at the inputs of a voltmeter, because its display reads 5 volts.

5.3 DIAGNOSIS

As previously described, the diagnosis consists of three steps: hypothesis generation, hypothesis testing, and hypothesis discrimination. MoBaD uses an exhaustive method of hypothesis generation, which means that all components (at a particular level) are considered as possible failures. Hypothesis testing is accomplished by using constraint suspension. We have not automated hypothesis discrimination.

Constraint suspension takes a list of candidates, which are possibly failed components, and tests each to see whether its failure would be consistent with the observed results. This is accomplished by suspending the candidate's constraints (setting its suspended slot to "true") and then simulating until either an inconsistency is encountered or the simulation is complete. An inconsistency exists when any port's predicted value significantly varies from its observed value. If an inconsistency is encountered, then the candidate is no longer suspected of failing since its failure does not explain the inconsistency. If the simulation is consistent, then that candidate becomes a suspect. Suspects with subcomponents can be tested individually to further refine the suspect list.

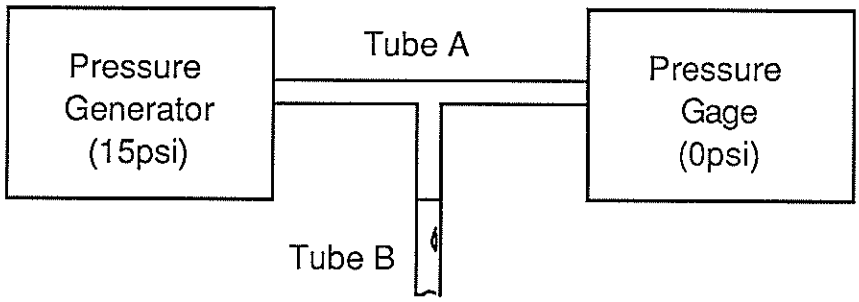

Figure 5.3-1 Example of a Failed Conductor

During the development of MoBaD, it became apparent that some types of components need to treat their constraint suspension differently. These components, which include pressure tubes and electrical wires, were placed into a subclass of components called conductors. When a conductor has its constraints suspended all conductors directly connected to it are also suspended. This corresponds to the situation in Figure 5.3-1, where if Tube A has a leak in it, it causes Tube B to cease functioning as well. If only Tube A's constraints were suspended, Tube B would cause a predicted value of 15 psi at the pressure gage. This inconsistency would cause the program to not list Tube A as a suspect. If

Tubes A and B are suspended together then the system correctly diagnoses both tubes as possible failures.

5.4 USER INTERFACE

The user interface of MoBaD is menu driven with a main window that displays the top level structure of the system being tested. The user can click the mouse on any port and set an observed value for that port. Components that contain subcomponents will display their internal structure when clicked on. The diagnostic process will pop up a text window where its results are displayed and the user can optionally monitor the values of all ports as MoBaD tests the system and its components. For further information on the details and use of the MoBaD user interface see Appendix A.

6. EXAMPLE

Figure 6-1 shows an example of a real-world measurement system one might wish to troubleshoot. The system would be used to calibrate the pressure transducer using the pressure standard as a reference. The transducer can measure 0 to 50 pounds per square inch and provides a linear 0 to 10 volt output. The standard provides a stimulus (a known pressure) and the voltmeter measures the response of the transducer. Two other components of this system go almost unnoticed: the pressure tube and the coaxial cable. These are conductors, since they are intended to cause some value to be conducted from one end to the other.

The pressure standard can be broken down into subcomponents: an input keypad and associated display, a pressure controller to generate a pressure entered at the keypad, a pressure gage to measure and display the pressure generated, and a pressure tube connecting the pressure controller, pressure gage, and pressure out port.

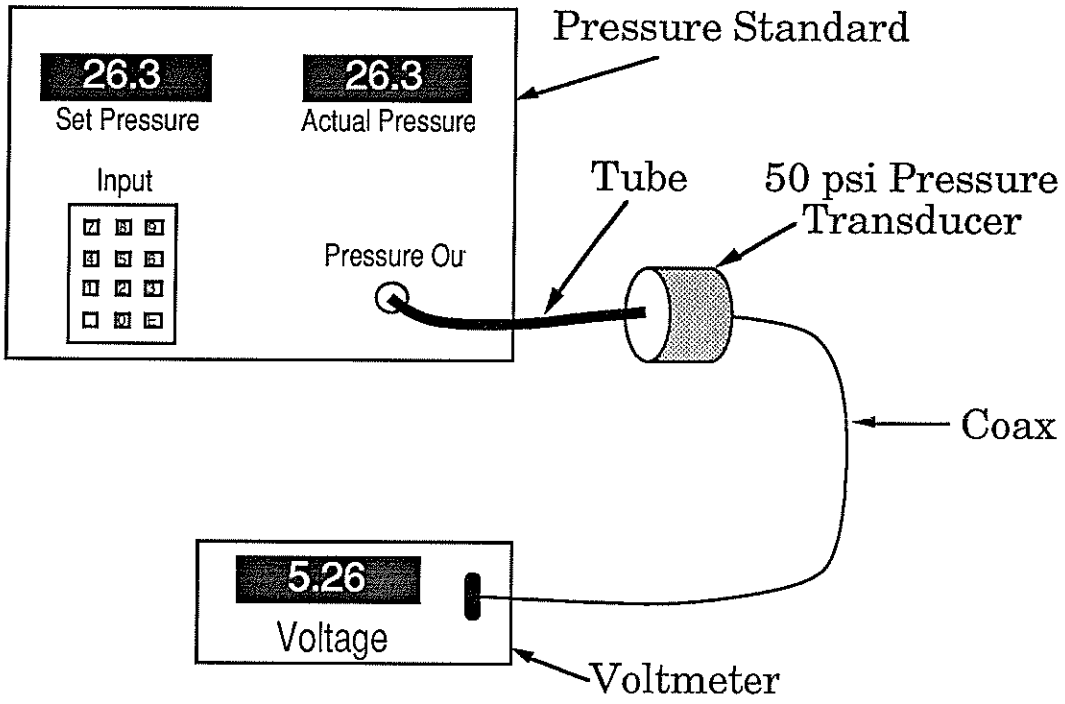

Figure 6-1. A real world measurement system.

This measurement system is represented in MoBaD as a subclass of Component named XducerCalSystem. The subcomponents of XducerCalSystem are also subclasses of Component: a PressureStandard, a Tube, a PressureXducer, a Coax, and a Voltmeter. Two of these components are actually subclasses of Conductor, which is itself a subclass of Component: the Tube and the Coax. PressureStandard also has subcomponents: a PressureController, a PressureGage, and a ThreePortTube. These components are all connected via ports, as shown in Figures 6-2 and 6-3. These figures are actual screen snapshots of the MoBaD user interface.

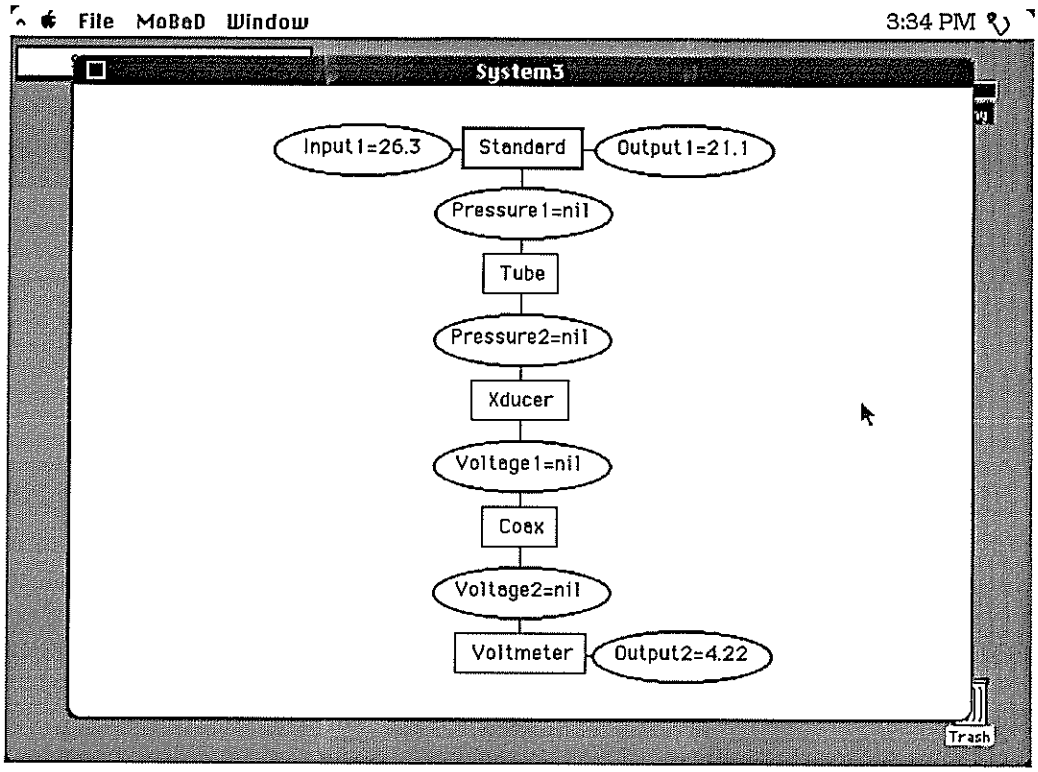

Figure 6-2. Model of structure of example system.

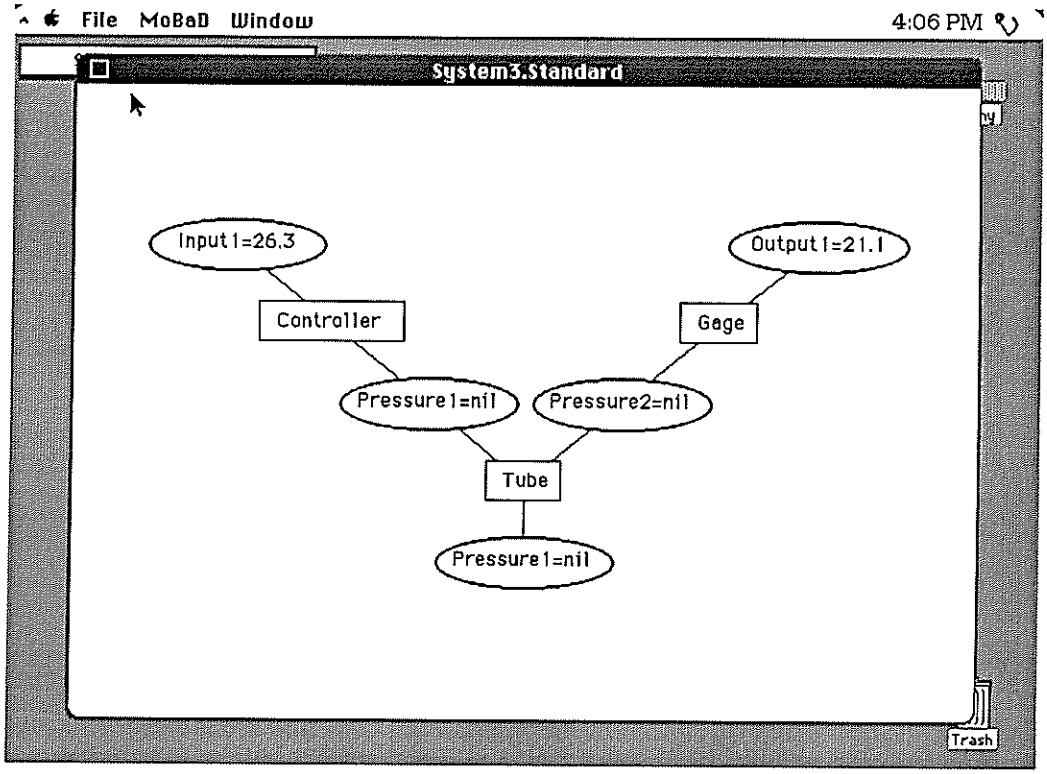

Figure 6-3. Internal structure of pressure standard.

The system is instantiated with the following Smalltalk/V code:

```
System3 := XducerCalSystem new initialize: 'System3'.
Input1 := InputPort new initialize: 'Input1'.
Output1 := OutputPort new initialize: 'Output1'.
Output2 := OutputPort new initialize: 'Output2'.
System3 connect: 'input' port: Input1.
System3 connect: 'std_output' port: Output1.
System3 connect: 'test_output' port: Output2.
DiagnoseSystem := System3.
System3 open.
```

System3, Input1, Output1, Output2 and DiagnoseSystem are Smalltalk/V global variables. The first line instantiates the system. The next three lines instantiate the ports which are external to the system. The next three lines establish a mapping between the internal and external names of the ports. The internal names are predefined in the classes.

DiagnoseSystem specifies the name of the system to troubleshoot when "Diagnose System" is selected from the MoBaD menu. The "open" message to System3 causes the MoBaD user interface to be activated.

For this example, suppose the tube that is internal to the pressure standard is leaking, which causes reduced system pressures. The observed values are as shown in Figure 6-2. MoBaD returns the diagnosis in Figure 6-4. As expected, MoBaD is unable to pinpoint the problem with the given information. Notice that both tubes are implicated as suspects since a leak in either one would cause the same symptoms. Another observation is needed for MoBaD to decide if the pressure controller or one of the pressure tubes is at fault. An observed value of 26.3 is assigned to the pressure port between the controller and the pressure standard tube. In Figure 6-5, a second diagnosis with this new information shows that the fault must lie in one of the two pressure tubes.

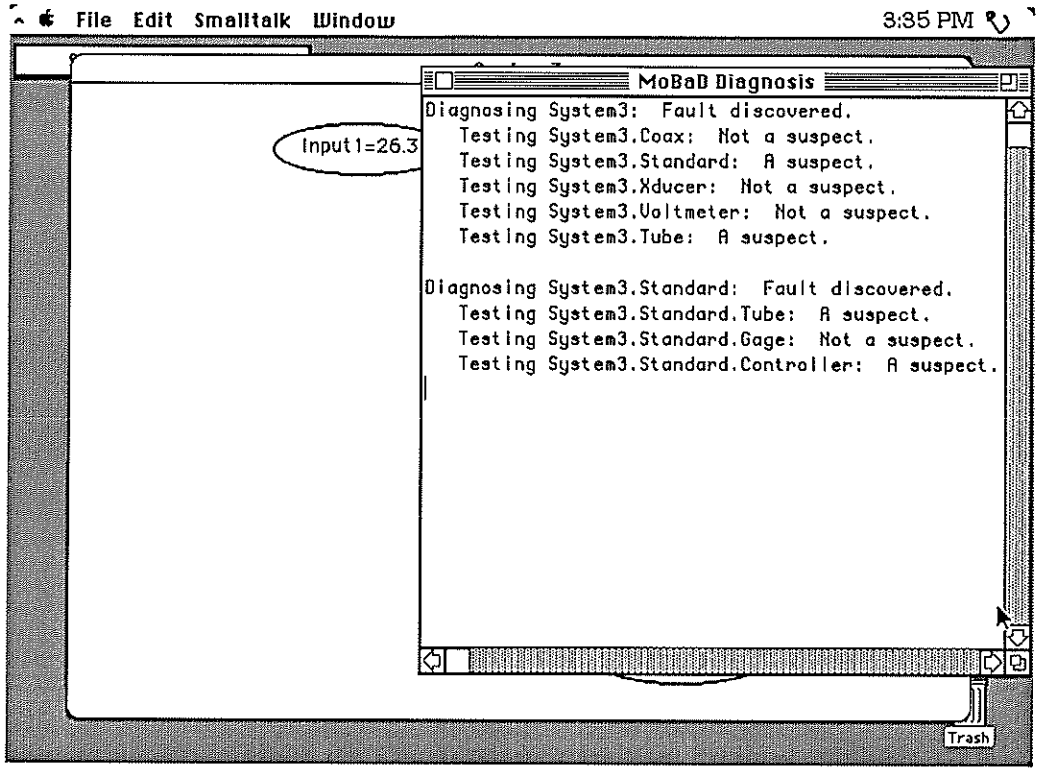

Figure 6-4. Diagnosis of system resulting in multiple suspects.

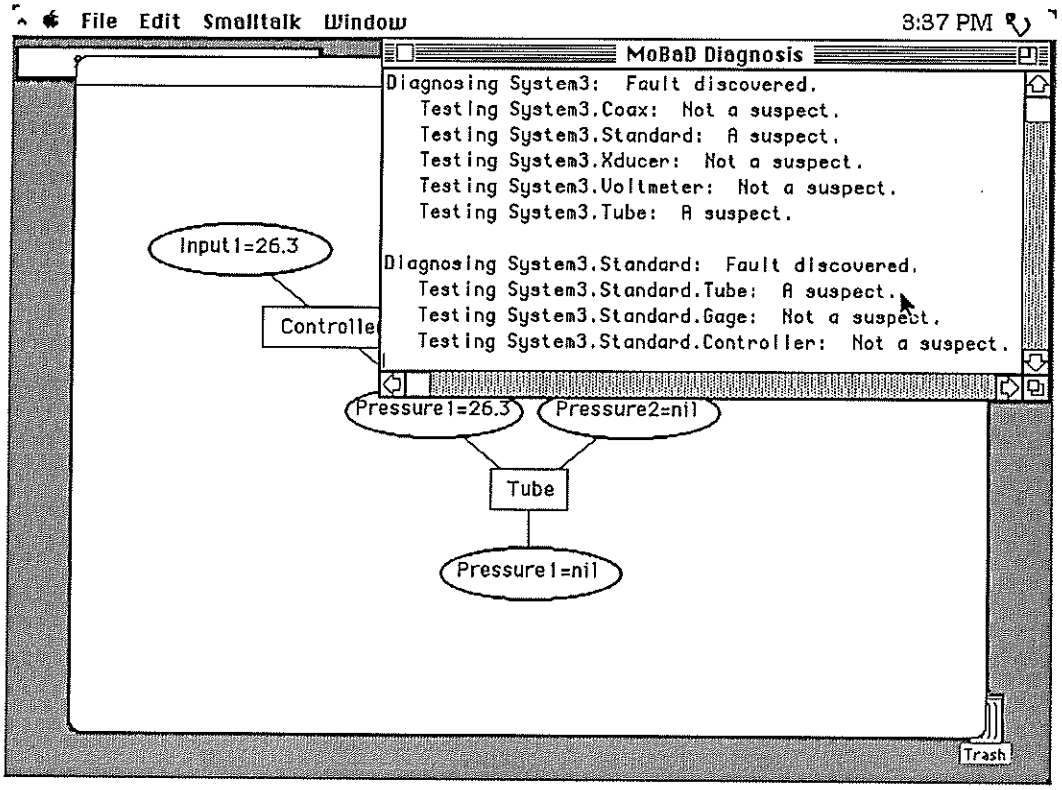

Figure 6-5. Diagnosis of system with additional information.

7. LESSONS LEARNED

Object oriented programming calls for thorough advanced planning. Many hours were spent in designing our objects and methods before the coding started. The initial task was to design the object hierarchy. Several brainstorming sessions resulted in a class hierarchy that was built using a bottom up methodology. We enumerated all the different types of objects in our model and abstracted their properties into two common classes: components and ports. For example, a tube and a voltmeter both contain ports and transfer data; therefore, both should be members of the component class. In the end, we hierarchically organized various types of ports and components in our system as seen in Figure 7-1.

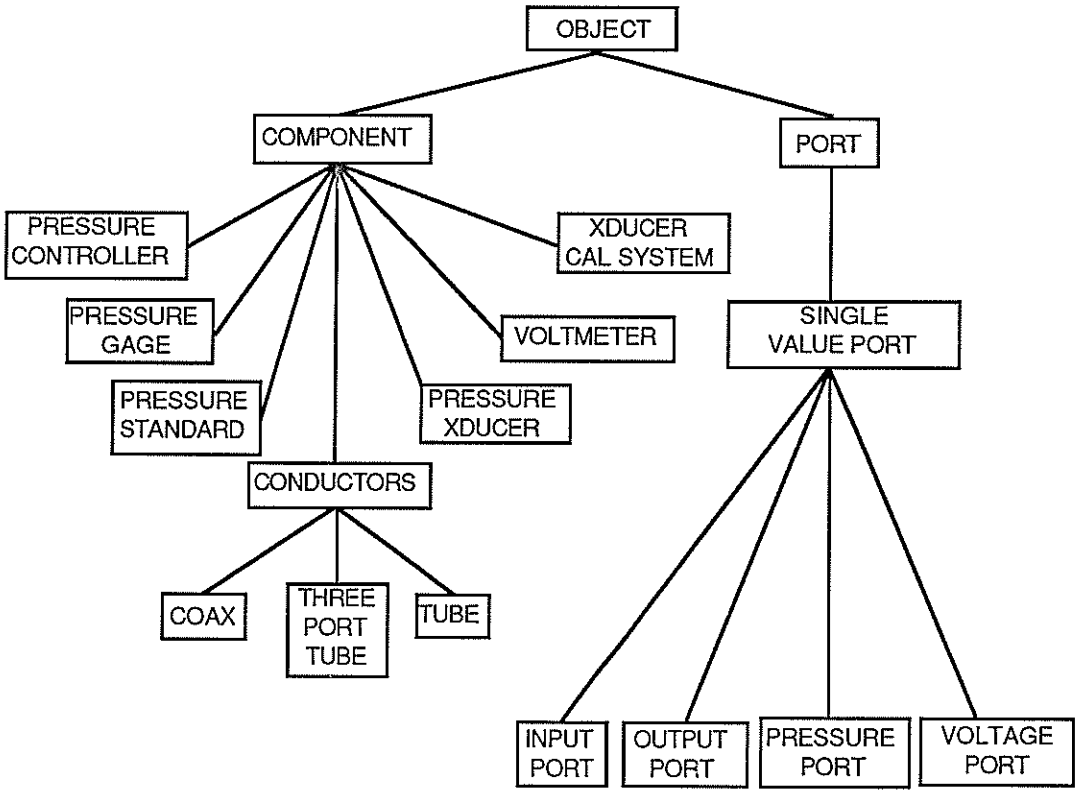

Figure 7-1 Class Hierarchy

Later into the development stage, the object model was further refined to be more hierarchical and self-contained for each component. For example, when a pressure standard object is created it instantiates a pressure generator, a pressure tube, and a pressure gage along with the proper ports and connections. The user is thereby freed from specifying all the details of internal components, ports and connections within a known module. This also simplifies the building of new models.

During the course of development, we experienced the unique features of object oriented programming. The tool we selected, Smalltalk/V, is relatively easy to use and offers an excellent programmer interface. The tools for inspecting objects and debugging methods proved to be very useful. However, the effects of the polymorphism of the language was, at times, confusing. Polymorphism, as explained in the Smalltalk/V manual, "is having different objects responding uniquely to the same message" (Smalltalk/V Tutorial and Programming Handbook, 1988). This offers a great convenience for the programmers to use the same method name for different objects to perform similar tasks. However, the hidden danger behind this is that the user assumes certain behavior for a method name and fails to check how the object really behaves upon receiving the message call. For example, we ran into problem with the "==" sign in our programming practice. The "==" sign checks if the entities on both side are equal for many data types. However, there is no method available for STRING type in Smalltalk/V to use the "==" sign. When "==" was used for string comparison, Smalltalk/V did not return predictable results.

We had heard that Smalltalk/V was a good prototyping tool as well as an excellent way to learn object oriented programming techniques and we found both to be true. Although at times Smalltalk/V seems to be limited, it is also powerful. One is given access to all of the internal code of Smalltalk/V. This library of objects is available to use or to redefine - if one dares. The greatest drawback of Smalltalk/V is the poorly designed manual. Syntax and commands are often difficult to find. During implementation of the user interface, these problems were especially frustrating. Another irritation was the flawed implementation of the Macintosh user interface. However, overall Smalltalk/V has proven to be a good tool for our project.

We have learned some lessons working together as a team. First of all, having the project well defined before development started played a crucial role in our success. Having a consensus of the system design minimized the integration efforts and helped to preclude the possible problems further into the development cycle. We took a special approach in our coding process. When we implemented the model and simulation modules of our prototype, all the team members were able to participate by using a Macintosh which was connected to a large screen projector. Coding standards, design issues, and object interactions were discussed and refined in this process. The result was tremendous; the whole team went through the development process together for the model, which is most important module in our prototype. This practice not only enabled us to learn the tool together but also ensured that each team member had a thorough understanding of the model, how it was implemented and the interaction among the objects. Next, we broke the prototype into logical modules and started to develop the program in parallel. Modular programming allowed us to move at a faster pace and enabled us to easily modify code once the validation and verification process began. We feel that our approach greatly enhanced team communication and minimized the integration problems.

One interesting problem encountered involved diagnosing the lower levels of multiple level systems. Initially, our model was designed in a way that all the predicted and observed values were reset before the diagnosis and then values would propagate through the model during diagnosis. However, when we designed this, we only considered the top level of our model for value propagation. When the model started diagnosing at a lower level, all the values were still reset first. This way, our model lost the input values to this lower level component and was unable to produce the proper results. The design of the model was modified so as to retain the input values of the lower level components when the diagnosis proceeded into deeper levels of our model. After this modification, the system behaved correctly during diagnosis of multiple layer models.

Our prototype demonstrates the benefits of model-based reasoning. We learned that designing a sound model was the most critical part of a model-based reasoning approach. A good model serves as a foundation for reasoning and makes the rest of the development process smooth. Once we finished modeling the component and port behaviors, we were able to diagnose systems with any combination of the implemented components and ports. If we had taken the rule-based approach, we would have had to implement individual diagnosis software for each system. However, the model-based reasoning approach enabled us to implement a knowledge-based system that can be applied to various measurement instruments with relatively little effort.

8. FUTURE DEVELOPMENT ISSUES

The most important feature lacking in MoBaD is the ability to perform hypothesis discrimination as described in the background section. The ability to suggest additional measurements in order to pinpoint the problem would make the program a valuable tool in the absence of an experienced engineer or technician.

Two features that would be relatively easy to implement in MoBaD include:

- Add a slot to the port class to specify the tolerance of the value of the port. This would permit a greater degree of inexactness in the system, which would make it more closely model the real-world.

- Give the user the ability to access more of the system from the graphical user interface. For example, add the ability to specify the positions the components and ports on the screen. Currently, these positions are hard coded in the class definitions.

The usefulness of this system would be improved by implementing an automatic probing and testing feature. This would provide the diagnosis system with the ability to probe a measurement system without the aid of an human.

MoBaD does not have a way of handling the rules-of-thumb often necessary to diagnose a measurement system. Perhaps MoBaD could be modified to call small rule-based expert systems to handle these special cases.

Currently, hypothesis generation is exhaustive (i.e. all components at the current level of diagnosis are tested). The program's efficiency would be improved if it only tested those components which can influence the port where an inconsistency is detected. This would preclude following irrelevant data paths through the structure. Also, if multiple discrepancies are detected, this information could be used to limit hypothesis generation.

MoBaD considers only the first of the layered category of failures listed in the background section: localized failure of function. A significant improvement would be realized if one or more of the other categories of failure were detectable during diagnosis.

Modeling of dynamic behavior has not been addressed by this project. Since certain measurement systems operate dynamically, and since some static systems exhibit dynamic behavior when malfunctioning, the ability to model and diagnose a dynamic system would be advantageous. This might require the application of qualitative physics theory, which is the modeling of physical processes without the use of physics equations (de Kleer and Brown, 1984).

9. CONCLUSIONS

The final product of our project is a prototype tool that uses model based diagnosis concepts to diagnose measurement systems. Our approach followed closely with the research that was done by Randy Davis. In addition, we implemented our prototype to be general enough to allow a variety of systems to be built using the model. Although most of the model based reasoning concepts were implemented in our prototype, further development is still needed to explore many related issues as discussed in the above section. When fully refined, this tool will significantly improve the equipment troubleshooting process without the extensive development time associated with building rule-based expert systems. In conclusion, we have proven the usefulness of model-based reasoning in diagnosis applications for measurement instrumentation.

REFERENCES

Davis, R., Reasoning from first principles in electronic troubleshooting. *Internat. J. Man–Mach. Stud.* **19** (1983) 403-423.

Davis, R., Diagnostic reasoning based on structure and behavior, *Artificial Intelligence* **24** (1984) 347-410.

Davis, R. and Hamscher, W. C., Model-based reasoning: Troubleshooting, Memo 1059, MIT Artificial Intelligence Laboratory, March 1988.

De Kleer, J. and Brown, J. S., A qualitative physics based on confluences, *Artificial Intelligence* **24** (1984) 7-83.

De Kleer, J. and Williams, B. C., Diagnosing multiple faults, *Artificial Intelligence* **32** (1987) 97-130.

Genesereth, M. R., The use of design descriptions in automated diagnosis, *Artificial Intelligence* **24** (1984) 411-436.

Shortliffe, E., *MYCIN: Computer-Based Medical Consultation* (American Elsevier, New York, 1976).

Smalltalk/V Mac Tutorial and Programming Handbook, Digitalk Inc., 1988.

APPENDIX A USER'S MANUAL

A-1 STARTUP

Load all three of the MoBaD disks that were supplied to you by your friendly neighborhood MoBaD dealer. Put everything contained on these disks in a separate folder on your hard drive. Click on the V.Image file that you wish to run. The examples below are all based on the Pressure System image file. Upon entering Smalltalk/V, the system will be displayed as shown in Figure A-1.

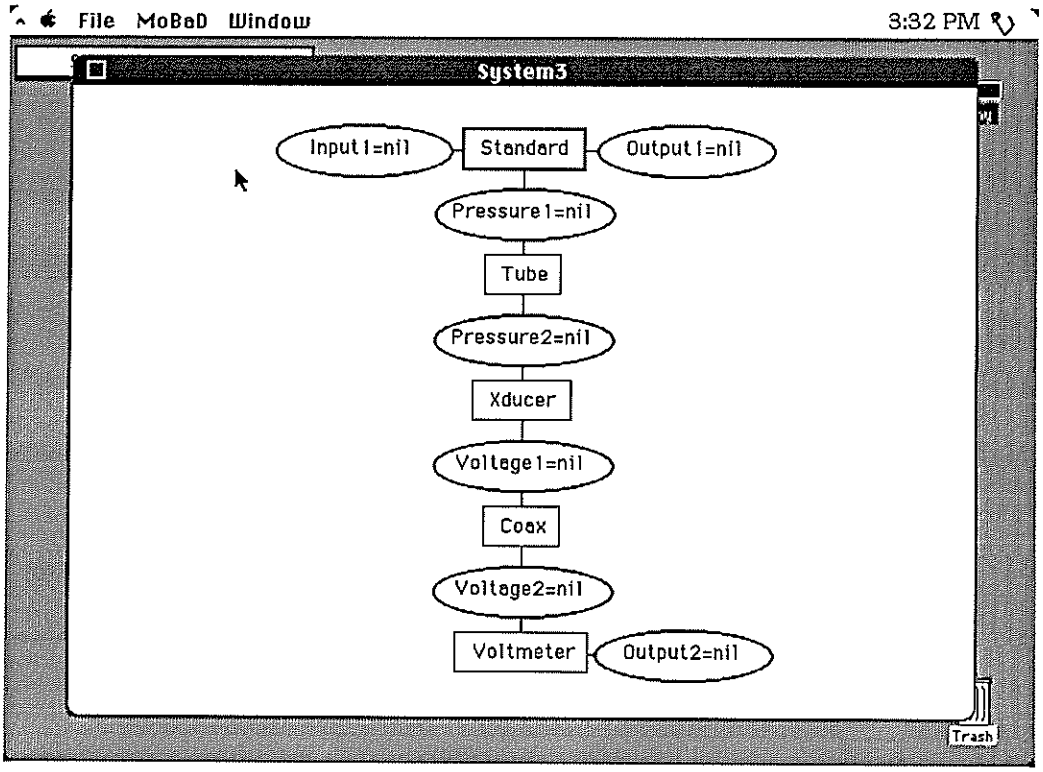

Figure A-1 Top Level System Display

The top level of the system is displayed. Each component is represented with a rectangle and each port is displayed as an ellipse. Any component with a darker outline is also represented at a lower level, since the system is represented hierarchically. To display the contents of any such component simply click on it. These lower level windows can be closed to return to the top level window. If the main window is closed then the system can be restarted by performing the 'File In' command (under the File menu) on the appropriate source file (PressureSystem or FullAdder).

A-2 INPUT VALUES

Before diagnosis the user must enter observed values. Usually, all input and output values are initially entered. To enter an observed value, click on a port. A message box prompting the user to input the observed value appears. After entering an observed value, click the "Accept" button as shown in Figure A-2. You will see that the value is changed in the system diagram.

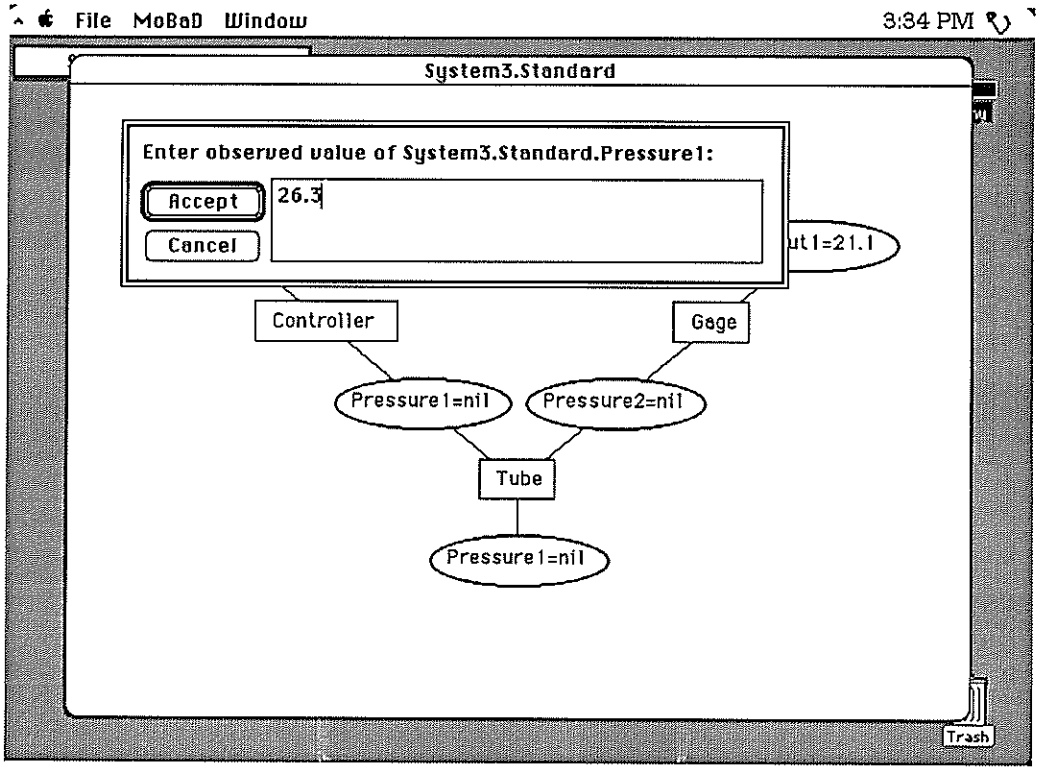

Figure A-2 Input Window for Observed Values

At any time all the values can be reset to nil by choosing "Reset Values" under the "MoBaD" menu. It is important to know that this will not only reset all observed values but also all the system's predicted values.

A-3 DIAGNOSIS

When you have entered the observed values you are ready for diagnosis. Under the "MoBaD" menu select "Diagnose System." The "MoBaD Diagnosis" window will pop up as seen in figure A-3. MoBaD initially diagnoses at the top level. The window will indicate if a fault is discovered and which components are suspects. If any component that is represented at a lower level is suspected, MoBaD will

continue diagnosis within that component. The data on that component will also be displayed.

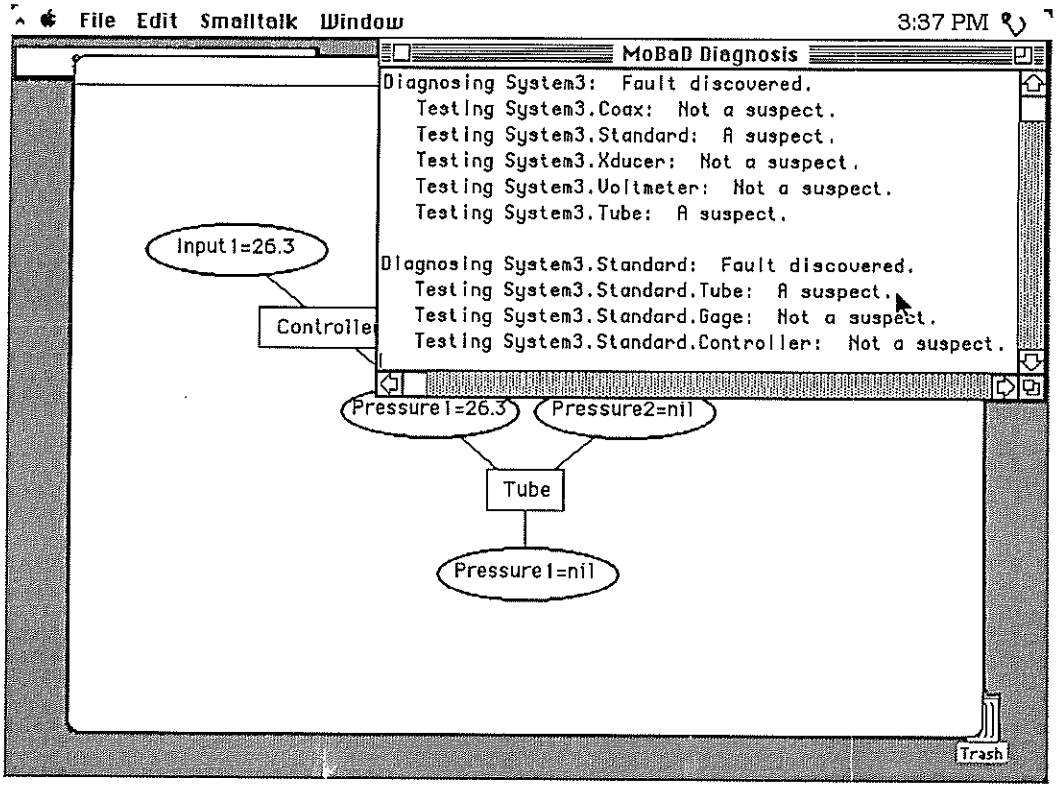

Figure A-3 Diagnosis Results

If you wish to monitor the port values, select "Port Monitor" under the MoBaD menu prior to beginning a diagnosis. Now when you diagnose, the window entitled "Port Monitor" will appear with the "MoBaD Diagnosis" window. The port monitor window will display the observed and predicted values for each port.

A-4 PRESSURE SYSTEM

The pressure system that is modeled has one input and two outputs. The output of the PressureStandard should be the same as the input value when the system is working properly. The output of the Voltmeter should be exactly one-fifth of the input value.

A-5 DIGITAL SYSTEM

The digital system modelled is a full adder. It has two data inputs, a carry input, a data bit output, and a carry bit output. All observed voltages entered should be either 5.0 volts or 0.0 volts.

APPENDIX B CODE LISTING

The code listing is available on request by contacting Mark Maxwell at (314) 232-5660 (daytime) or (314) 731-7028 (evenings). Or you may write to:

McDonnell Douglas Corp.
Attn. Mark Maxwell, MC 1022192
P. O. Box 516
St. Louis, MO 63166