COMPONENT ONTOLOGICAL REPRESENTATION OF FUNCTION FOR CANDIDATE DISCRIMINATION IN MODEL BASED DIAGNOSIS

by

Amruth N. Kumar

August 1994

A thesis submitted to the
Faculty of the Graduate School of State
University of New York at Buffalo
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

This research was partially funded by Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700 and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, as part of the Northeast Artificial Intelligence Consortium (NAIC).

Gratefully Acknowledging...

Shambhu Upadhyaya, my advisor, who was always encouraging, guiding and monitoring, yet never restricted or imposed.

Sargur Srihari, who not only introduced me to Model Based Reasoning but also played my silent guardian angel through the years.

Sreejit Chakravarty, who brought invaluable non-Artificial Intelligence perspective to my research.

Helene Kershner, who gave me all the breaks I needed to flesh out my Ph.D. experience.

Ken Smith and **Devon Bowen**, who were always there when I needed help.

My parents, **Saroja and Krishna Holla**, who were extremely patient, ever encouraging and never demanding of me. No thanks is enough for all that they have done and been to me over the years.

Kulki, **Safvi** and **Shaky**, my friends: they cheered me up, they pained me bad, but they never bored me!

Ellie, **Sally**, **Gloria**, **Jaynee**, **Leslie**, **Natalie**, **Heike** - the secretaries whose warmth and cheer I will sorely miss.

Contents

List of Figures

Abstract

This dissertation proposes a representation of function based on component ontology. Further, it proposes the use of function for suspect ordering in model based diagnosis.

In component ontological representation, function of a component is expressed in terms of the ports of the component. This representation has many advantages over the currently used state and process based representations: (i) Function of a component can be represented in isolation of the environment of the component. Therefore, libraries of function models can be built. These models are re-usable. (ii) Function model of a complex device can be built by composing the function models of its components. This ensures the fidelity of representation. Further, this process can be automated, and takes time linear in the number of components in the device. (iii) The representation is linear in space complexity. Function model of a simple device is linear in the number of its ports, and that of a complex device is linear in the number of its components. This compares favorably with state and process based representations which can be typically exponential in space complexity, and hence, intractable.

In the dissertation, the principles behind component-ontological representation of function are enunciated. Classes are presented as a representation based on these principles. Algorithms are developed to automatically compose the class model of a device from those of its components.

In addition, function is proposed to be used for candidate discrimination in model based diagnosis. It is as effective as the traditionally used fault probabilities, because both prompt for similar sequences of measurements to validate diagnosis. The advantages of using function instead of fault probabilities are: (i) It facilitates explanation generation based on causality during diagnosis. (ii) It is readily available from device design unlike fault probabilities. It can be used to diagnose devices whose fault probabilities are unavailable.

In the dissertation, two techniques are proposed for function based candidate discrimination. A diagnosis algorithm is developed, which employs these techniques to diagnose the class model of a device. The proposed techniques are device-independent.

Together with discrimination techniques, Classes serves as a computational model of function for diagnosis. Its scalability and domain-independence are demonstrated by applying it to an electronic printer buffer and an automobile carburetor.

Chapter 1

Introduction

Diagnosis is the problem of locating the faulty component in a device, which gives rise to the symptoms at the outputs of the device. Suppose a device with n components is to be diagnosed. Any of the n components could have failed. The objective of diagnostic reasoning is to find the failed component.

Classical electrical engineering techniques for diagnosis are based on test generation. They typically hypothesize both the nature (stuck-at/stuck-open) and location of the fault in a circuit. They then generate and apply suitable tests which can validate their hypothesis. These approaches have several drawbacks: They are restricted to diagnosing only known fault modes from a predetermined fault set. They do not easily scale up to complex devices. They have to exhaustively test all the locations in a circuit for faults. The resulting cost is exponential in the size of the device. Finally, it is not clear how they can be applied to other domains. Therefore, Artificial Intelligence (AI) techniques were proposed for diagnosis, which are domain-independent, scale up to complex devices and are more focused than exhaustive testing.

Rule-based [Buchanan and Shortliffe, 1984], Case-based [Hammond, 1989] and Model-based reasoning are some of the popular AI techniques used for diagnosis of devices. In rule based systems, empirical rules are used, which associate each symptom with a faulty component(s) in the device. When a symptom is observed, the rules pertaining to the symptom are fired, and the components indicated by those rules are suspected to be faulty.

Case based systems consult a database of previous case histories of the device to be diagnosed The symptoms of the device are used as keys to retrieve relevant cases from the database, and the diagnosis suggested by those cases is advised.

In model-based reasoning, first, a model of the device is used to simulate the correct behavior of the device. This is compared with the observed behavior, and symptoms are deduced based on the differences between the two behaviors. Next, a subset of the components of the device are suspected, which are structurally connected to the symptoms and can explain them. Finally, measurements are prompted for, across these components.

Model-based systems use knowledge of the correct behavior of the device rather than its fault model or fault history. Therefore, they have the following advantages over rule-based and case-based systems:

1. They can handle new devices and novel faults, for which no rules or case histories may be available. Rule and case based systems are said to be brittle on account of their inability to handle these situations.

2. The required knowledge can be easily culled from device design. Rules and case histories must be acquired by experience, and the acquisition is laborious and time-consuming.

3. The required knowledge of correct behavior is finite and can be certified to be comprehensive. On the other hand, rules and case histories cannot be comprehensively enumerated because of the possibility of unforeseen faults. This makes it harder to evaluate the competence of rule and case based systems.

Model based systems have been used for diagnosis in several domains such as analog electronic circuits [deKleer, 1976, Brown, 1976, Brown *et al.*, 1982, Cantone *et al.*, 1983, Pan, 1984, Dague *et al.*, 1987, Ben-Bassat *et al.*, 1988], digital electronic circuits [Davis, 1984, Genesereth, 1984, deKleer and Williams, 1987, Hamscher, 1990, deKleer and Williams, 1989], electrical devices [Struss and Dressler, 1989, Ginsberg, 1986, Reiter, 1987, Poole, 1989], physiology [Patil, 1981, First *et al.*, 1982] and fluid transfer [Scarl *et al.*, 1985].

1.1 Model Based Diagnosis

Representation and reasoning are the main issues in model-based diagnosis. Representation is the task of building a device model adequate for diagnosis. For a given device, it is a one-time effort. Reasoning includes the techniques and algorithms used to diagnose a device from its model.

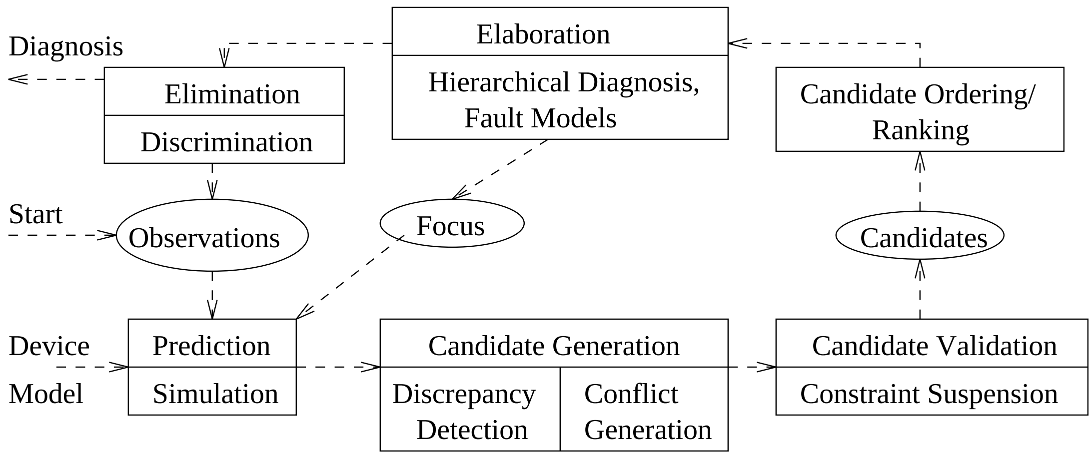


Figure 1.1: Architecture of Model Based Diagnosis Systems [Hamscher & Struss 90]

Traditionally, model of a device has consisted of descriptions of its structure and behavior [Davis, 1984, Genesereth, 1984, deKleer, 1984, Chandrasekaran and Milne, 1985, deKleer and Williams, 1987, Reiter, 1987, Davis and Hamscher, 1988, Hamscher, 1988]. Structure description consists of a listing of components in the device and their interconnections. Behavior is a specification of the relation between inputs and outputs of the device. For instance, behavior description of a combinational circuit in the electrical domain is its truth table.

The steps in diagnostic reasoning are shown in Figure 1.1, adapted from [Hamscher and Struss, 1990]. Initially, *observed* values at the inputs as well as outputs of the device are obtained. During *prediction*, device behavior is simulated using input values to calculate all *predicted* values at intermediate and output ports. All output ports where a discrepancy is detected between predicted and observed values are considered to be part of the symptom. Based on these symptoms, a list of suspect components called *candidates* are generated [deKleer and Williams, 1987, Reiter, 1987]. Next, it is verified whether and how each candidate could actually cause the symptoms at the outputs of the device, during *candidate validation* [Davis, 1984]. The validated candidates are *ordered* according to their likelihood of being faulty [Chen and Srihari, 1989, deKleer and Williams, 1987]. *Elaboration* may be carried out next, i.e., diagnosis may be continued at the next lower level of structural hierarchy within candidates. Alternatively, measurements may be initiated across candidates to either verify that they are indeed faulty or *eliminate* them from the candidate list. Traditionally, this last stage is called *discrimination*. In this dissertation, discrimination is meant to, in addition, include candidate ordering.

The techniques used for candidate generation may be based on device structure and/or behavior. Upstream Tracing [First *et al.*, 1982] as well as the classical electrical engineering concept of cone of influence [Kalme, 1986] are based on structure. In these techniques, all the components that are structurally connected to the symptom are generated as candidates.

Candidate generation techniques based on behavior include constraint satisfaction when behavior is modeled as constraints [Davis, 1984], resolution when behavior is represented using logical propositions [Genesereth, 1984] and truth maintenance [deKleer and Williams, 1987, Hamscher, 1988]. Candidate validation is performed by constraint suspension [Davis, 1984] on behavior.

Candidate ordering is performed to enable elimination of the more likely candidates first. A good candidate ordering strategy focuses diagnosis by reducing the number of components across whom measurements are carried out. Hence, it saves time and effort. However, candidate ordering is an *inherently heuristic* procedure [1].

Existing candidate ordering techniques are based on structure [Chen and Srihari, 1989], fault probabilities [deKleer and Williams, 1987] and ease of repair [Fink and Lusth, 1987]. Effectiveness of the structure-based technique is dependent on the topology, i.e., the number of fan-ins and fan-outs in the device. The technique based on fault probability requires collecting large bodies of probability information, which may be unavailable or hard to obtain. Ease of repair measures are similarly hard to collect, and the technique is subjective.

One of the proposals of this dissertation is to use function for candidate ordering. Some of the advantages of using function instead of the traditionally popular fault probabilities are: it is easily obtainable from the device design, is available for even novel devices, and facilitates generation of explanation during diagnosis.

1.2 Function

Function can be defined in two ways [Rosenman and Gero, 1995]:

- Operational: the relation between input and output of energy, material or information [Rodenacker, 1971];

[1] If the candidate most likely to be faulty can always be predicted correctly, there is no need to diagnose the device in the first place!

- Purposive[2]: the relation between the goal of the human user (purpose) and the behavior of the system.

Function of a component is a description of what it does in its environment, either as intended by its designer, or as interpreted by its user. A component (its environment) may be hardware (device), software (code), personnel hierarchy (organization), or any other system where the whole (i.e., environment) is a sum of its parts (i.e., components). Yet another computational definition of function of a component is that it is the effect of the behavior of the component on the behavior of the device in which it is embedded [Lind, 1994, Kannapan, 1995].

Function is distinct from behavior. Function of a component in a device is identified with respect to the device. It can vary from one device to another. On the other hand, behavior of a component is the relation between its inputs and outputs. It is independent of the device in which the component is embedded. It can be specified without considering any interaction among larger collections of components in the device. Hence, behavior is local in nature [deKleer, 1976], whereas function is non-local. For instance, the behavior of a D flip-flop is that its output follows its input. In a circuit, it may be used as a buffer. Buffering or storing data is its function in that circuit. Again, behavior of the ignition key of an automobile is to complete a circuit; its function is to start the car.

Function is an abstraction of behavior. Therefore, it can be used to organize and selectively access causal (behavior) knowledge of the component. For instance, it helps focus on missing causality during device redesign. When reasoning based on behavior gets bogged down by excessive details, function can be used to improve the focus of reasoning. For instance, it helps discriminate among suspects during diagnosis. Consequently, function may be used to address the scaling problem.

Function is being explicitly represented and used in applications as diverse as Diagnosis [Milne, 1985, Milne, 1987, Fink and Lusth, 1987, Brajnik *et al.*, 1990, Abu-Hanna *et al.*, 1994, Hawkins *et al.*, 1994, Lind, 1994], Device (Re)Design [Umeda *et al.*, 1992, Kannapan, 1995], Explanation Generation [Allemang and Chandrasekaran, 1991], Vision [Sutton *et al.*, 1993, Brand *et al.*, 1993], Naive Physics [Hodges, 1992],

[2]Some researchers distinguish between function and purpose. Purpose differs from function in that it relates to human notions of utility in a socio-cultural context. However, from a strictly mechanistic perspective, when function of a component is considered not with respect to a social context, but with respect to the system of which it is a part, there is no need to distinguish between these two terms.

Robotics and Natural Language Understanding [Pittges *et al.*, 1993]. These applications may be classified into three categories [Kumar, 1994b], viz., motivational: preceding an event; discriminatory: during the event; and explanatory: after the event. Function can be used to motivate a decision, discriminate among choices at hand, or explain an observation. Diagnosis is a discriminatory task.

Weak methods such as search, constraint propagation, and deduction dominate Artificial Intelligence reasoning because of their generality. Function as a source of knowledge is amenable to such methods of reasoning. Furthermore, function provides additional domain-specific knowledge which renders these weak methods strong. In most current applications, function has been used in this role of "additional" useful knowledge rather than as sole knowledge.

Fidelity, precision and efficiency are the attributes of a model [Hamscher, 1988]. Function is an abstraction of behavior. A function model is built at the expense of precision of the behavior model, i.e., the model's ability to make predictions that are strong enough to be falsifiable by observations of the actual device. Due to loss of precision, reasoning supported by function models is often heuristic, and based on abduction. However, function model preserves the fidelity of the behavior model, i.e., an abstracted function model still does not support incorrect predictions about the device. Therefore, function models continue to ensure correctness of reasoning.

1.3 Function in Model Based Diagnosis

Function knowledge has been proposed to augment model-based diagnosis [Milne, 1985, Fink and Lusth, 1987, Steels, 1989, Sticklen and Chandrasekaran, 1989, Brajnik *et al.*, 1990, Keuneke, 1991, Abu-Hanna *et al.*, 1991, Bradshaw and Young, 1991, Franke, 1991, Hunt and Price, 1992]. During model based diagnosis, function has been used for simulation [Sticklen and Chandrasekaran, 1989, Hawkins *et al.*, 1994, Sembugamoorthy and Chandrasekaran, 1986, Lind, 1994] and candidate validation [Fink and Lusth, 1987, Lusth, 1985]. During simulation, function models have been used to index behavior models. During candidate validation, function has been used to prompt for measurements.

In this dissertation, function is proposed for candidate discrimination, including suspect ordering. Every component in the cone of influence [Kalme, 1986] of a device output behaviorally contributes to the output. Therefore, most of the components can explain the symptoms at the output during candidate validation. However, only one or some of these components are designed to be functionally responsible for the various aspects of the output. For instance, in an automobile, all the following components contribute behaviorally to deliver fuel to the engine: gas tank, fuel line, fuel pump, fuel filter and carburetor. However, only the carburetor is responsible for the rate of fuel delivered. Therefore, for a given symptom at a device output, the functionally responsible components are far fewer than the components whose behaviors contribute to the output. This property is exploited to use function for candidate discrimination.

Function in model based systems has been represented in terms of states [Sembugamoorthy and Chandrasekaran, 1986, Sticklen and Chandrasekaran, 1989, Hawkins *et al.*, 1994, Iwasaki *et al.*, 1995] and processes [Brajnik *et al.*, 1990]. These state and process ontological[3] representations are intractable because of the potentially exponential number of states/processes in a device. Moreover, the limits of competence of these representations cannot be readily evaluated because of lack of guidelines for choosing appropriate states/processes and an inability to verify if the choices are comprehensive or even satisficing.

In this dissertation, function representation based on component ontology is proposed wherein, function of a component is represented in terms of ports of the component. This representation has several advantages including linear space and time complexity, re-usability, composability and domain-independence. It can be built systematically from design models. Hence, its competence can be evaluated.

Representational and inferential adequacy, and representational and acquisitional efficiency are some of the desired characteristics of knowledge representation schemes [Rich and Knight, 1991]. The proposed component ontological representation is shown to be representationally and inferentially adequate for diagnosis and representationally and acquisitionally efficient.

Finally, a computational model for function is proposed in this dissertation, called *Classes*. Classes incorporates the above two proposals, viz., function based candidate ordering and component ontological representation of function. As part of the computational model, function primitives called *classes*[4] are defined. Algorithms are developed to build class models of composite devices from those of their components.

---

[3]Consistent with its use in Computer Science [Bobrow, 1984], the word "ontology" is used to mean the study of the primitives in terms of which one views an object/world.

[4]A Note on Usage: The term *Classes* is used to specify the computational model, whereas *classes* is used to specify the function primitives in the computational model.

Two function based candidate ordering techniques are proposed. A model based algorithm is developed which uses these ordering techniques to diagnose the class model of a device.

1.4 Dissertation Organization

In the next chapter, literature is surveyed on representation of function and its use in model based diagnosis. Several associative as well as model based representations of function have been proposed by researchers. Among model based representations, state and process based ontologies have been used. In model based diagnosis, function has been used for simulation and candidate elimination. In the final section of the chapter, the work reported in this dissertation is motivated and its merits are listed.

In Chapter 3, principles behind component-ontological representation of function are enunciated. Next, Classes is proposed as a computational model for function, based on this ontology. Function primitives are proposed for components as well as subsystems of a device, and they are illustrated with a simple example of a doorbell. Finally, our claims regarding the advantages of component ontological representation, viz., re-usability, composability, and linearity of space and time complexity are justified. The representation facilitates model based diagnosis.

In Chapter 4, algorithms are developed to systematically build the function model of a device from those of its components. These algorithms illustrate the composable nature of component ontological models. Deterministic algorithms are described first, which assume that function models of components at the floor level of representation [Kumar and Upadhyaya, 1990] are provided. Next, heuristic algorithms are outlined, which assume that only inputs and outputs of components have been identified. The heuristic algorithms try to deduce function from structure.

Function based diagnosis of devices is described in Chapter 5. Two function based candidate discrimination techniques, viz., teleological and default order techniques are proposed. Next, a model based algorithm is developed, which uses these discrimination techniques to diagnose devices whose function models are constructed as described in Chapters 3 and 4. Finally, our discrimination techniques are qualitatively compared with traditional discrimination techniques, and the advantages of using function are analyzed.

In Chapter 6, our implementation of the diagnosis algorithm is reported. A printer buffer device, which has the complexity of a microcomputer is chosen as the test device for our computational model of classes. A complete class model of the printer buffer is built in order to demonstrate the scalability of Classes. Representation and diagnosis discussed in Chapters 4 and 5 respectively are applied to the Display Unit of the printer buffer. A sample run of its diagnosis is provided in Appendix A. In order to highlight domain-independence of Classes, the class model of an automobile carburetor is built and discussed.

In Chapter 7, the limitations of our computational model of Classes are listed. The implications of our work to the field of model based reasoning are noted. Our initial goals are revisited to examine how they have been met. Contributions of the dissertation are summarized. Pointers are provided on how Classes can be adapted and used by the reader. Topics of future work are listed.

Chapter 2

Background

Due to the leverage that function provides during reasoning, it is being applied in several fields and domains. In model based diagnosis, function has been used for simulation and candidate validation.

Function has been represented by association with components as well as by definition with respect to them. Among definitional representations, state and process ontologies have been used. Two clear schools of thought are identifiable among definitional representations: those that are based on flow models [Paynter, 1961], and those that developed out of the early seminal work done at Ohio State University on state based representation [Sembugamoorthy and Chandrasekaran, 1986]. A representative sample of literature from each of the above-mentioned schools, viz., associational, flow model and state based representations is examined in this chapter. Finally, the work reported in this dissertation is situated and motivated.

2.1 Associational Representations

In the following works, function is represented by associating it with components rather than by defining it with respect to them. Such a representation is device-specific. It is unsuitable for model based diagnosis because of lack of composability, as discussed in Section 3.1.

During reasoning, these works again use associations of functions with symptoms. The resulting diagnosis is brittle because it fails for symptoms which have not been explicitly encoded in rules.

Theory of Responsibilities

Milne's "theory of responsibilities" [Milne, 1985] is based on "second principles": principles used to describe subsystems of a device. Second principles are abstracted from "first principles" (behavior) applicable to a device. In the theory, rules are used to associate parts of an electrical circuit with parts of output (waveform) for which they are responsible. E.g., "The diode D1 is responsible for positive peak of the output of a rectifier circuit." Given that the symptom is some missing part of the output, the associated component is suspected. Milne's theory has been demonstrated for analog electrical circuits. It is not proposed how the theory can be extended to other domains.

Integrated Diagnostic Model (IDM)

In Integrated Diagnostic Model (IDM) [Fink, 1985, Lusth, 1985, Fink and Lusth, 1987], representation primitives such as transformer, regulator, reservoir, conduit and joint are proposed. These are primitives for function as well as structure of a device, and are used in a semantic network representation. Diagnosis is proposed based on "first principles", a term traditionally used to mean behavior. The diagnostic system consists of an "inference engine" which uses four general reasoning rules, such as, "If output of a function unit is unknown, ask the user". These rules, which are used for candidate elimination, i.e., to prompt measurements across components with unknown inputs/outputs, use only the structure content of the device model. Function content of the model is used to validate diagnosis of a rule based system, by qualitative simulation. The authors [Fink and Lusth, 1987] have applied their framework to electrical and mechanical domains.

Function of a device is decomposed into subfunctions, and subfunctions are associated with components or symptoms in yet another associative approach by Steels [Steels, 1989].

2.2 State Based Representations

State based representations are definitional in nature. Most of the following state based representations trace their origins to the early work on Functional Reasoning (FR) done at Ohio State University [Sembugamoorthy and Chandrasekaran, 1986]. They derive units of function representation on demand. These units are abstractions of behavior states. The semantics of each unit of function must be separately specified. An advantage of this approach is that, resolution of representation can be varied to suit the current needs by choosing the units appropriately.

If the chosen units are too fine grained, representation becomes intractable because of potentially exponential number of states. If the chosen units are too coarse, representation becomes brittle and ineffective during reasoning. The decision on what is a suitable resolution of representation, based on the tradeoff between complexity and efficacy, is often subjective and varies from user to user. Therefore, it is hard to evaluate the competence as well as limits of competence of state based representations.

State based representations meet the requirements of a knowledge representation scheme [Rich and Knight, 1991] as follows:

- Representational adequacy is good for simulation and explanation generation. However, it may be inadequate for diagnosis, because of the shallowness of its knowledge content that can help discriminate among devices.

- Acquisitional efficiency is low. Since the resolution of the model is variable, and is dependent on the task for which it is being built, models may not be reusable. Domain and task expertise is necessary to build them.

- Inferential adequacy depends on the choice of states. Hence, competence of the representation is hard to evaluate.

- Representational and inferential efficiency can be poor, since, potentially exponential number of states can exist in even a reasonably complex device.

State based representations have been predominantly used for simulation (information processing task [Marr, 1982]) rather than discrimination (classification task) during model based diagnosis. They have also been used for various other applications such as design and redesign [Goel and Chandrasekaran, 1989], Failure Modes Effects Analysis [Hunt and Price, 1992], debugging software [Allemang and Chandrasekaran, 1991], explanation generation during diagnosis [Keuneke, 1991] and natural language understanding of device design descriptions [Pittges *et al.*, 1993]. MDX2, FM, FBS, CFRL, TED and Musarela are some popular state based representations of function proposed in literature.

MDX2

In MDX2 [Sticklen and Chandrasekaran, 1989], the model of a device consists of a description of its structure and a list of its functions. Each of the listed functions is expressed in terms of pre-conditions, postconditions and the behavior used to realize the function. State based ontology is used for behavior representation: behaviors are represented as directed acyclic graph structures in which nodes represent states of the device, and links represent annotations explaining why transition from one state to another takes place.

During reasoning, function model is first used to derive a causal net structure, whose start is the set of preconditions that are satisfied by the current state, and goal is the postcondition of the function that needs to be realized. All required behaviors in the net are recursively expanded to the required level of detail. The resulting causal net is called a Particularized State Diagram (PSD) because it is tailored to a particular set of preconditions. Now, reasoning is carried out by "consequence finding" on PSD, which is a variation of simulation.

During diagnosis, the authors [Sticklen and Chandrasekaran, 1989] compile their function representation to generate an associational diagnostic expert system. The compilation generates a root node corresponding to the malfunctioning device. It generates a specialist/function checker for each function of the device and places these checkers as sons of the root node. Again, for each function checker, if the behavior required to realize the function has n states, the compilation generates $n-1$ function checkers corresponding to all state transitions. The generated tree of function checkers or specialists is the compiled expert system for the device. During diagnosis, this tree is traversed. The competence of the resulting diagnosis is hard to evaluate since it is dependent on the resolution of behavior descriptions. Besides, for reasonably complex devices, the expert system generated can be intractably large (since a specialist is required for each state transition in the device), and the generation can take potentially exponential time.

Functional Modeling: FM

FM [Sticklen *et al.*, 1993, Hawkins *et al.*, 1994] is an extension of the work on FR [Sembugamoorthy and Chandrasekaran, 1986] and MDX2 [Sticklen and Chandrasekaran, 1989]. In FM, the authors again propose direct and automatic generation of abductive diagnostic knowledge from function models. This knowledge captures two relationships between components and state variables, namely, *evokes* and *accounts-for*. They are the same as those used in INTERNIST [Pople, 1977], and are similar to the means-end relations of Multilevel Flow Modeling [Lind *et al.*, 1992].

During diagnosis, two step abduction is carried out on the compiled knowledge structure: all the hypotheses (components) that explain at least one symptom (in a state variable) are identified, and these hypotheses are grouped to form cover sets of hypotheses that together explain all the symptoms. Although motivation for the work is to compile efficient diagnostic knowledge, the authors admit that compilation is potentially intractable, as also the diagnosis algorithm they propose.

FBS Model

FBS (Function-Behavior-State) is a function representation scheme very similar to FR, proposed for Computer Aided Design [Tomiyama and Umeda, 1993]. Representation of function in FBS consists of a name in the form of "to verb object" (E.g., to move table), a decomposition of the function into subfunctions (along abstract-concrete or whole-part relations. E.g., to stop table, to move table by motor) and function-behavior relationships describing behaviors that can perform the function. Behavior and state are represented based on Qualitative Process Theory [Forbus, 1984]. The authors have used FBS to redesign devices for functional (as opposed to structural) redundancy, and to generate control software for electro-mechanical devices.

CFRL

CFRL (Causal Functional Representation Language) is a formalism for representing device function, and is an extension of FR [Sembugamoorthy and Chandrasekaran, 1986]. The highlight of CFRL is its formalization of the interpretation of function in terms of behavior.

Function is represented in CFRL as a triplet $\{D_F, C_F, G_F\}$, where D_F is the device whose function is F, C_F is the context (pre-conditions) in which the device functions, and G_F describes the function goal to be achieved (postcondition). G_F is an expression consisting of conditions, quantifiers, boolean connectives and CPDs (Causal Process Description), where a CPD is a directed graph with partial states as nodes, and causal and/or temporal relations as arcs. Justifications may be attached to causal arcs similar to annotations in MDX2 [Sticklen and Chandrasekaran, 1989]).

A CPD is an abstraction of behavior: it specifies some of the facts that should be true during the behavior and the causal/temporal ordering among them.

Behavior itself is represented as a linear sequence of states, called a trajectory (a path in an envisionment [Forbus, 1984]). Conditions are specified for matching a state in a trajectory with a node in a CPD. A trajectory is said to achieve a function F if G_F matches it and conditions C_F are satisfied.

Physical knowledge of the device is represented as a collection of model fragments, each of which represents a physical object or phenomenon. Associated with each state in its behavior model is a set of active model fragment instances representing the phenomena occurring in the state. These model fragments are evaluated to generate successive states in the behavior representation.

CFRL has been used to evaluate predicted behavior of a device and to determine if it achieves the device's intended function. CFRL is also being applied for design refinement of electromechanical devices.

TED

TED [Franke, 1991] is a language devised to represent device function for design and diagnosis. The language expresses function as behaviors prevented, guaranteed or introduced. It represents function in terms of variables, states, partial states and scenarios. TED has focused on representing and acquiring statements of purpose, and does not address the representation of causal descriptions.

MUSARELA

Musarela [Abu-Hanna *et al.*, 1991] introduces several innovative concepts and perspectives about function, symptoms and the epistemology of models. The epistemology of models consists of kernel models and interpretative models. Kernel models are independent of domain and application (diagnosis, design, etc.). Structure, behavior and topology (layout) models are all kernel models. Interpretative models are intended interpretations of behavior according to some criteria. Function is considered to be an interpretative model along with use (teleology as in TED [Franke, 1991]) and task-compiled models.

Function is used as an abstraction of behavior. The intended properties of a device, without which the device would be classified as faulty are termed functions. These properties are not necessarily functions, but in general, their realization indicates the realization of the functions to which they are related. In addition, functional parameters are defined with respect to functions and their values are determined either by direct qualitative simulation or by simulation of the underlying behavior. The identified functional parameters of a device are inter-related to generate a network, which is an abstraction of a state-based behavioral description of the device.

An innovation the authors of Musarela propose is to define a symptom as a discrepancy at the functional level rather than at the behavioral level. For instance, discrepancy in the voltage of output of a device need not necessarily indicate a symptom. However, at a more abstract level, if there is a discrepancy in the logic level of the output, the device is said to have a symptom. They claim that description of a symptom at the functional level is more natural to use. Function models are also proposed to be used for suspect generation because function can better focus the search for suspects.

Since the authors of Musarela represent function in terms of "chunks of behavior", efficiency of diagnosis is affected by the choice of chunks. As they admit, their functional models are neither unique nor specifically built for diagnosis. Therefore, care should be taken to appropriately select their chunks for diagnostic needs.

The authors of Musarela describe an elaborate scheme of cooperation among multiple models of a device: structure, behavior, function, topology, etc. They propose an ontological semantic network which is used as a means of communication between model types. They use an interesting classification of ontologies in each epistemological type (such as structure and behavior): model and domain ontology. E.g., model ontology for structure consists of terms such as component, terminal and conduit. Corresponding domain ontology terms in electrical domain include clock, pin and wire. Model ontology for behavior consists of parameter, state, constraint, operator, etc. Corresponding domain ontology for electrical domain includes terms such as logic-level, clock-state, =, and not. They also investigate issues in organizing hierarchical models by cost functions (such as observability and reachability) that affect diagnosis.

2.3 Flow Model Representations

Flow model representations are the other clear school of definitional representation of function, and are based on the concepts of flow and effort as used by Paynter [Paynter, 1961] and the Bond Graph community. In these representations, function is treated as a relation between input and output of energy, matter and information [Rodenacker, 1971]. A set of function primitives are defined, and functions of all the encountered components are cast in terms of these primitives.

One criticism leveled against flow model representations is that it is hard to list a complete and/or adequate set of function primitives for a domain. Completeness and adequacy of primitives depend on the desired resolution of representation. Yet, the competency of flow model representations can be better evaluated than that of state based representations because:

- Function primitives are high level abstractions of sequences of behavior states. They are at a coarser granularity than behavior states, and hence offer fewer choices to the modeler. (It is not surprising that various flow model representation formalisms define similar sets of primitives even though they were developed independently [Brajnik *et al.*, 1990, Lind, 1994, Kumar and Upadhyaya, 1995]. Such concurrence is hard to find among state based representations.) In order to evaluate the competence of a flow model representation, it is sufficient to clearly define the semantics of these primitives, which are finite in number.

  Furthermore, it is easier to list domain-specific function primitives than it is to list desirable states of behavior: behavior states are device-dependent whereas function primitives are not.

Flow model representations meet the requirements of knowledge representation [Rich and Knight, 1991] as follows:

- Representational and inferential adequacy are good for diagnosis, since, knowledge necessary to discriminate among devices is encoded and is available. Since flow model representation is at a coarser granularity than device states, it is not suited for simulation as well as state based representations.

- Acquisitional efficiency is moderate. Domain experts may still be necessary to build the model. But the model is reusable: function model of a device in a domain is relatively invariant with respect to the function primitives chosen for the domain.

- Representational efficiency is good, since the model is built in terms of a finite number of primitives. Inferential efficiency is also good because semantics need to be defined for only this finite set of primitives.

The work of Brajnik et al, and MFM are two flow model representations proposed in literature, and are described next. Our work on Classes described in Chapter 3 also belongs to this school of representation.

Brajnik et al

Multiple models are proposed by Brajnik et al [Brajnik *et al.*, 1990, Brajnik *et al.*, 1991, Chittaro *et al.*, 1989] to represent devices based on structure, behavior, function, teleology and empirical rules. Similar to MUSARELA [Abu-Hanna *et al.*, 1994] an elaborate framework is provided for cooperation among these five models. The authors distinguish between function and teleology: " teleology is a relation that considers function as a particular way to implement purpose" (page 75, [Brajnik *et al.*, 1989]). However, this relation is fixed for a device during design. Therefore, the distinction is irrelevant for diagnosis.

Function is treated as an abstraction of behavior, expressed in terms of generalized variables [Paynter, 1961]. A function model is built using *functional roles* and *processes*. Functional roles are function primitives defined on the basis of the four generalized variables in a device namely, flow, effort, impulse and displacement [Paynter, 1961]. The defined function primitives are *capacitor, inductor, resistor, ideal conduit, e-generator, f-generator* and *transducer*. They are defined for components in a device, independent of the device. Further, a "comprehensive" set of processes is proposed such as charging, discharging, transportation and transducing.

Therefore, building the function model of a device involves labeling the functional role of each of its components and identifying all possible processes in the device. This representation is not composable, i.e., function model of a complex device cannot be built by composing the function models of its components. In addition, representation based on process ontology is potentially intractable since, theoretically, each subset of components in the device can participate in a separate process.

During diagnosis, function is used for prediction by simulation. Given the function model of a device, and its inputs and initial conditions, simulation yields a description of the functioning of the device and a time diagram specifying when each process is active.

Teleological model is used in conjunction with function model for diagnosis. Based on the purposes in the teleological model which are not satisfied by a device, specific processes in its function model are suspected. However, no algorithm has been provided that explicates this cooperative reasoning. Further, purpose is defined as a subset of processes in the function model. Therefore, potentially exponential number of purposes can be identified with respect to the processes. Hence, representation of and reasoning with purposes and processes as proposed by Brajnik et al is computation-intensive. It also appears that knowledge used to associate purposes with processes is device-specific and not device-independent.

Multilevel Flow Modeling (MFM)

The model of a device in Multilevel Flow Modeling [Lind *et al.*, 1992] consists of a description of its components, their functions, and the goals that they serve in the device. Functions are expressed in terms of function primitives such as *source, sink, storage, transport, barrier* and *balance*, which are defined for flows of mass and energy. These primitives are defined in terms of states of devices as well as their port variables. Flow functions of components in a device are concatenated to build flow structures.

A set of goals are identified for each device. These goals are quantifiable and may be categorized as production, safety and economy goals [Stephanopoulos, 1983]. Flow structures of a device are connected to its goals by two means-ends relations called *achieve* and *conditions*. These relations provide the basis for reasoning with MFM models. MFM is called a multilevel model because of the several levels of abstraction it incorporates along the means-end axis (components, functions, goals) and the part-whole axis (component-device, subfunction-function, subgoal-goal).

Choosing goals at the appropriate level of abstraction for a device, and identifying functions which realize these goals requires expertise. As the author of MFM admits, it is hard to analyze the sensitivity of functional descriptions to chosen goal contexts. Therefore, it is hard to evaluate the competence of MFM models for reasoning about devices.

During diagnosis, an MFM model is automatically translated into a set of coordinated reasoning systems propagating values through a network of nodes and relations. Diagnosis is carried out as a search of this net [Larsson, 1993]. A goal is chosen to be diagnosed, and flow structures connected to it by means-end relations are suspected. Each flow function in the structure is now checked systematically for fault. Note that the choice of goals affects the competence of diagnosis.

Consolidation

The Consolidation method of qualitative reasoning [Bylander, 1988] is proposed for representation of behavior and not function. In Consolidation, primitives such as *allow*, *pump*, *create*, *move* and *destroy* are proposed to model behavior. Whereas our function primitives proposed in Chapter 3 model *cause*, these behavior primitives model *effect*. Cause - effect is one of the relations between function and behavior [Rosenman and Gero, 1995]. Consolidation is proposed to generate potential behaviors of a device from its structure and potential behaviors of its components. The author [Bylander, 1988] admits that behavior is not the required output for tasks such as diagnosis. It is not clear how Consolidation can be extended to diagnosis.

2.4 Summary

In summary, function has been represented by association with components as well as by definition with respect to them. The associative representations [Milne, 1985, Fink and Lusth, 1987, Steels, 1989] are device-specific and not composable, which makes them unsuitable for use in model based reasoning systems as will be argued in Section 3.1. The definitional representations use state [Sticklen and Chandrasekaran, 1989, Franke, 1991, Abu-Hanna *et al.*, 1994, Hawkins *et al.*, 1994, Iwasaki *et al.*, 1995, Lind, 1994] and process [Chittaro *et al.*, 1989, Brajnik *et al.*, 1990, Brajnik *et al.*, 1991] ontologies, i.e., they define function in terms of states and processes in the device. Therefore, these representations can be intractable. With few exceptions [Tomiyama and Umeda, 1993], these representations are not composable either. Finally, state based representation is subjective and its efficacy depends on the choice of states. Hence, the limits of competence of state based representations is hard to evaluate.

Most of the existing proposals for using function for diagnosis use function for qualitative simulation [Chittaro *et al.*, 1989, Brajnik *et al.*, 1990], candidate elimination by measurement [Lusth, 1985, Fink and Lusth, 1987] or compilation of causal networks that facilitate associative diagnosis [Sticklen and Chandrasekaran, 1989, Hawkins *et al.*, 1994, Lind, 1994]. Therefore, function knowledge is relegated to simulating or validating candidates rather than generating or discriminating among them. The proposals that use function for candidate discrimination are rule based [Milne, 1985, Steels, 1989]. The resulting diagnosis is brittle, i.e., it cannot diagnose faults that have not been explicitly accounted for in the knowledge base.

In this dissertation, the following function based proposals are made:

- Component ontological representation of function is proposed. The principles underlying component ontological representation are enunciated. This representation is linear in space complexity, and hence, efficient as compared to state and process based representations. The resulting function models are re-usable and composable. Due to composability, they are also suitable for use in model based systems.

- Function is proposed for candidate discrimination (as opposed to simulation) during model based diagnosis. It is just as effective as using fault probabilities for diagnosing devices because both prompt for similar sequences of validating measurements. It has several advantages over the traditionally used fault probabilities: it is easily available from device design, it is available for novel devices also, and facilitates generation of explanation during diagnosis.

- Based on the above two proposals, a computational model for function is proposed, called Classes. Representation in Classes is based on component ontology. Algorithms are developed that exploit/demonstrate the composable nature of Classes to automatically construct the function model of a device from those of its components. Two techniques are proposed which exploit function in the form of classes to discriminate among candidates during model based diagnosis. These techniques are device-independent, and not brittle. A model based algorithm is developed which applies these techniques to diagnose the class model of a device. Class models of an electronic printer buffer and an automobile carburetor are built to demonstrate the domain independence and scalability of Classes.

According to the spectrum of definitions of model based diagnosis proposed by Console and Torasso [Console and Torasso, 1991], our approach is abduction-based and utilizes the correct behavior of the device. In contrast, the attempts to improve candidate discrimination based on fault probability [deKleer and Williams, 1987, Hamscher, 1988, Farquhar, 1989, Bakker *et al.*, 1992, Poole, 1992] are consistency-based approaches which use a Truth Maintenance System (TMS).

Abductive problem solving has been proposed as both a unitary step from symptoms to diagnosis [Reggia *et al.*, 1985] and as a two step process [Josephson and Josephson, 1993]. In the two-step process, a set of hypotheses is generated such that each hypothesis accounts for one or more of the symptoms. Hypotheses in this set are now grouped to form cover sets of hypotheses which together explain all the symptoms. In this dissertation, a multi-step abductive diagnosis process is proposed wherein, abduction is applied repeatedly, to small fragments of the model at a time in order to focus diagnosis.

Structure, behavior, function and empirical rules are the four levels of model based diagnosis proposed by Milne [Milne, 1987]. The work reported in this dissertation encompasses the first three levels.

In the next chapter, component ontological representation of function is proposed. Principles underlying the representation are enunciated. The computational model of Classes is proposed based on these principles.

Chapter 3

Function Representation

State and process ontologies are currently used for representation of function. They are potentially exponential in complexity and their limits of competence are hard to evaluate. In this chapter, component ontology is proposed as an alternative to these ontologies for representation of function.

The principles behind component ontological representation of function are enunciated in Section 3.1. A computational model of function called Classes is proposed and described in Section 3.2. The advantages of using component ontological representation are analyzed in Section 3.3.

In brief, the computational model of Classes is as follows: flows of energy, matter and information in a device are identified as *signals*. Modular subsystems in a device corresponding to causal signals are termed *signal-lines*. A device may consist of several signal-lines and each signal-line may in turn be made up of several components. A *class* is defined as the function of a device/component or signal-line.

The class of a component C_1 is defined as the function of the component with respect to its ports (i_{11}, i_{12} and o_{11}) as shown by the solid arrows in Figure 3.1. A set of class primitives for components is defined in Section 3.2.1. The class of a signal-line S_1 (consisting of C_1 and C_2) is defined as the function of port i_{32} with respect to a connected component C_3 as shown by the dotted arrow in Figure 3.1. A set of class primitives for signal-lines is defined in Section 3.2.2.

The function model of a device is built by first identifying every signal-line in its structural model. Classes of components are identified with respect to the signal-lines they are part of, and classes of signal-lines are identified with respect to the components to which they provide input. Algorithms to automatically build the function model of a device are developed in the next chapter. During diagnosis, given

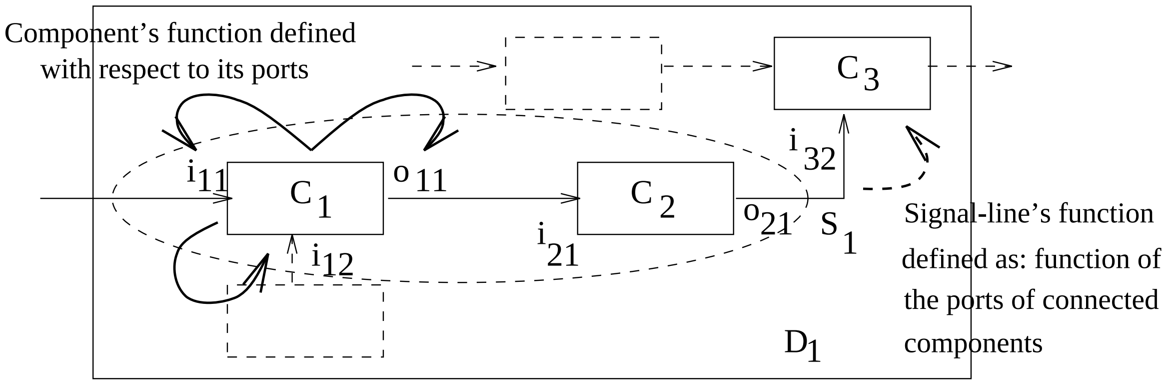


Figure 3.1: Defining Function of Components and Subsystems

a symptom in a device, a subset of signal-lines in it (or connected to it) is suspected to have failed, and given a symptom in a signal-line, a subset of components in it is suspected to have failed, based on its class. Diagnosis based on Classes is described in Chapter 5.

3.1 Component Ontology Principles

In this section, the nature of function is explicated and principles underlying component ontological representation of function are enunciated. For generality, the term "environment" is used in the following discussion, which may be understood to mean the "embedding device" in the case of device components.

Definition 1 *A function f_2 is defined to be specialized with respect to $f_1 = P_1 \wedge \ldots \wedge P_n$ (where P_i are predicates), if $f_2 = f_1 \wedge P_{n+1} \wedge \ldots \wedge P_{n+m}$ for $m > 0$. f_1 is in turn, generalized with respect to f_2. f_1* subsumes *f_2.*

Components are designed to be modular. Each component is provided with a set of ports through which it interacts with its environment:

- A component is designed to exhibit a behavior by means of which, it processes the signals at its ports. Therefore, it is designed to have specific functions with respect to its ports.

- The ports are expected to deliver specific signals to a component, from its environment. Therefore, they are designed to have specific functions with respect to the component.

Function of a Component:

Observation 1 *A component designed to carry out a function f can also carry out functions which are more generalized than f in specific environments, but not functions which are more specialized than f.*

Explanation: Suppose a component is designed to carry out a function $f = P_1 \wedge P_2 \wedge \ldots \wedge P_n$ where P_i are predicates. A part of the functionality of the component can be disabled so that the component carries out a new function $f_1 = P_1 \wedge P_2 \wedge \ldots \wedge P_{n-1}$ in a specific environment. Note that f_1 is generalized with respect to f. However, in order to use the component for a more specialized function $f_2 = P_1 \wedge P_2 \wedge \ldots \wedge P_n \wedge P_{n+1}$, it is necessary to incorporate additional hardware in the component which implements P_{n+1}.

It follows that, the function for which a component is designed is the most specialized function that the component can carry out. By definition, this is the function of the component with respect to its ports. All the functions which the component fulfills in specific environments can be expressed as generalizations of this function. Therefore, it follows:

Observation 2 *The function of a component in an environment can be expressed in terms of the function of the component with respect to its ports.*

Assuming that reasoning can be carried out correctly even when the function used for a component is the one defined with respect to its ports rather than the one defined with respect to its environment, we can now identify/specify the function of a component in isolation of its environment. Our assumption is borne out by analysis later in this section.

Principle 1 *The function of a component in any environment can be defined with respect to its ports.*

E.g., function of a switch in a table lamp is to turn the lamp on/off. In terms of its ports, function of a switch is to *control* what flows through the ports. Therefore, function of the switch in the table lamp may be said to be that of *control*.

Two alternative arguments for the above principle are as follows:

- Function of a component in a device is its purpose in the device, and is defined with respect to its environment, i.e., the device. However, a component is connected to its environment through its ports. Its function with respect to its environment is realized through these ports. Therefore, function of a component with respect to its environment may be specified in terms of function of the isolated component with respect to its ports.

- Function is an abstraction of behavior and is expressible in terms of behavior. Furthermore, it is convenient to express function in terms of behavior because behavior is amenable to computational techniques on account of its quantitative nature. Since behavior of a component is specified with respect to its ports, function may also be specified with respect to its ports.

Note that, whereas behavior of a component has been traditionally represented in terms of values at the ports of the component, principle 1 above proposes to represent function of a component in terms of its ports and not values at its ports. Further, this representation is independent of the environment of the component. For instance, in Figure 3.1, the function of C_1 embedded in the device D_1 is expressed in terms of its own ports i_{11}, i_{12} and o_{11} as shown by the solid arrows.

Function of a Subsystem: Function is also associated with subsystems in a device and subsystems connected to a device[1]. A subsystem can be encapsulated into a component. From principle 1, the function of the subsystem can now be expressed in terms of its ports. These are the ports of the subsystem connected to (the ports of) components causally adjacent to the subsystem.

The following property among behaviors of components in a device is relevant to the expression of functions of subsystems:

Observation 3 *In the direction of causality, the range of behavior of a component is a subset of the domain of behavior of the successive component.*

Explanation: From Figure 3.2, note that, for proper operation of the composition of components C_1 and C_2, C_2 must be able to accept and process any and all outputs generated by C_1. In other words, non-shaded portion of the *effective*[2] range of behavior $b(C_1)$ should be empty. Therefore:

[1]The term subsystem is used to refer to a causally modular collection of some of the components in a device, and their interconnections. Subsystems are later referred to as signal-lines in this dissertation.

[2]Effective range of a device behavior is the range through which the device is configured to behave in a specific environment. It is a subset of the device's actual range. E.g., when a transistor is used as an amplifier, it is operated only in its linear range, which is its effective range. The actual range

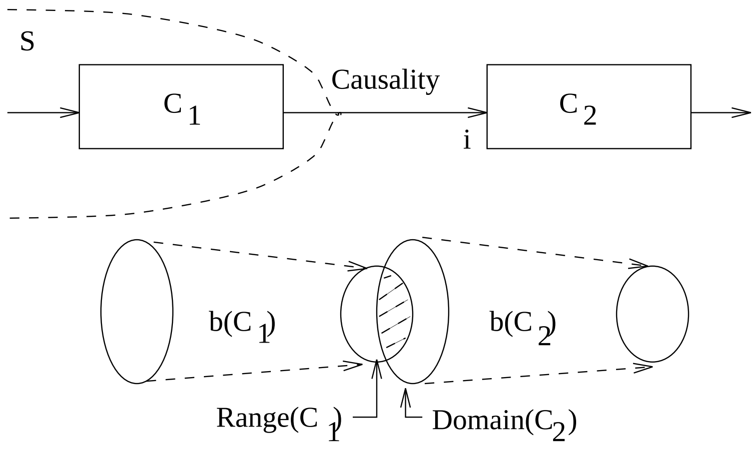


Figure 3.2: Subsumption in the Direction of Causality

$$Range(b(C_1)) \subseteq Domain(b(C_2))$$

This implies that in order to quantify the causal effect of C_1 on C_2, it is sufficient to consider the domain of C_2.

Assume that C_1 is the final component in a subsystem S connected to the input i of C_2. C_2 was designed to expect/accept a specific function f at its input port i. Irrespective of the output delivered by the subsystem S at i, it is admitted as function f by C_2. f is the function of port i with respect to component C_2. Hence, the following principle:

Principle 2 *The function of a subsystem in a device can be specified as the function of the port(s) of the component(s) causally adjacent to it.*

For instance, referring back to Figure 3.1, function of subsystem S_1 consisting of components C_1 and C_2 is specified as function of port i_{32} of C_3. The function of port i_{32} is in turn defined with respect to component C_3 as shown by the dotted arrow.

Component Ontological Representation: Even though function of a component was defined as being relevant with respect to its environment in Section 1.2, it follows from Principles 1 and 2 that:

of $b(C_1)$ considered in isolation could be a superset of the domain of $b(C_2)$. However, when C_1 is connected to the input of C_2, it is by design configured to operate such that its effective range is a subset of the domain of $b(C_2)$.

1. function of a component can be specified with respect to its own ports, and in isolation of its environment;

2. function of a subsystem connected to a component can be specified as function of the component's port connected to the subsystem.

Since function is represented in terms of ports of the component, these principles are the basis of *component ontological representation.* Since function is represented in terms of ports in lieu of (and, in isolation of) the environment, functions of components can be statically compiled into device libraries. They need not be built dynamically, on demand, and tailored to the specific environment at hand. The compiled function models can be retrieved and used as necessary.

Function model of a complex/composite device can be built by composing function models of its components as follows:

- Subsystems in the device are identified.

- Functions of these subsystems are determined to be *functions of the ports of the components* to which they are connected.

- Within each subsystem, functions of components are determined to be *functions of those components with respect to their ports*, through which they are connected to the subsystem.

Hence, functions of subsystems as well as components are determined in terms of function models retrieved from the component library. The above steps are recursively applied to all the levels of structural hierarchy of the device.

Compositional nature is an advantage inherent to component ontological representation. Composability also implies that representation can be automated. Principles 1 and 2 provide dual basis for automatically building the function model of a device from those of its components.

Device-Independence of Model Based Reasoning: The following observation is useful to highlight another advantage of using component ontological representation:

Observation 4 *Model based reasoning is device-independent compared to rule based reasoning only insofar as the device knowledge, i.e., the model it uses is compositional in nature.*

Note that the techniques used in model based reasoning (constraint suspension, dependency tracing, etc.) as well as those used in rule based reasoning (deduction, forward/backward chaining) are domain-independent. Diagnostic rules used in rule based systems are device-specific, but so are the device models used in model based systems. The difference between the two systems is that, the device knowledge used in model based systems is compositional in nature, i.e., model of a device can be built from models of its components by composition. But, diagnostic knowledge used in rule based systems, i.e., associations between symptoms and faults cannot be derived for a device by composing the diagnostic rules of its components.

To elaborate, device knowledge used in model based reasoning consists of structure and behavior models of the device. A library of such component models is built and made available to the reasoning system. When a model based reasoning system considers a complex device, it builds a model of the device by composing the models of its components retrieved from the library. However, if a rule based system is to diagnose a device, it has to first build the rule base pertinent to the device from scratch. It cannot compose or re-use any diagnostic rules available for the components of the device. Therein lies the device-independence of model based systems as compared to rule based systems.

Note that, the above discussion is about models versus rules and may not be misconstrued as the form versus content debate [Davis, 1989, Keller, 1989]. Our argument is not about the **form** of representation. For instance, device models used in model based reasoning may themselves be represented in the form of logical rules and resolution may be used to reason with them [Genesereth, 1984]. Our argument applies to the **knowledge content** used in model based and rule based diagnosis systems: structure and behavior models are composable whereas empirical rules associating symptoms with faults are not.

Since composability is one of the primary representational advantages of model based systems over rule based systems, a representation is deemed adequate for model based reasoning only if it is composable. Hence, the principle:

Principle 3 *Representation of function should be composable for use in model based reasoning.*

Component ontological representation is composable and hence, suited for model based reasoning whereas, most associative and state based representations are not.

Accounting for Context Sensitivity of Function: Function of a component may differ from one environment to another. It may also change in a given environment, depending on the signals at the ports of the component. For instance, an AND gate may be used to "control" (by using one of its inputs to regulate the passage of signal at the other input) or "convey" (by setting one of its inputs to a permanent high). In component ontological representation, specialization relation among functions of a component is exploited to account for these changes.

Suppose a component can assume one of several functions $f_1, \ldots, f_n$ depending on its environment. Since all these functions are abstractions of the same component behavior and must be realized through that behavior, they are generalizations or specializations of each other. Alternatively, since they are all generalizations of the function of the component with respect to its ports as argued earlier, they are generalizations or specializations of each other.

Observation 5 *Even if the function assigned to a component's model is more specialized than its function in the current environment, the correctness of reasoning is ensured.*

Explanation: Let the current function of a component C in a device where it is embedded be:

$$f_1 = P_1 \wedge P_2 \wedge \ldots \wedge P_n$$

where P_i are predicates such as *ToMake* and *Guarantee* clauses [Sticklen and Chandrasekaran, 1989, Keuneke, 1991]. Let the function actually assigned to the component's model be f_2. If f_2 is more specialized than f_1, by Definition 1 it includes at least one additional conjunct P_{n+1}:

$$f_2 = f_1 \wedge P_{n+1}$$

Failure of any of the predicates $P_1, \ldots, P_{n+1}$ results in an output symptom. Therefore, function f_2 fails whenever f_1 fails. But, the vice versa is not true. It follows that, during function based diagnosis, component C is suspected in a larger number of cases (symptoms) if f_2 is assigned to it than if f_1 is assigned to it, and these cases cover all the cases corresponding to f_1. In other words, if a more specialized function f_2 is assigned instead of the actual function f_1, the component is never classified to be correct when it could be faulty (cases $P_1, \ldots, P_n$), but the vice versa classification is possible (case P_{n+1}). This ensures the correctness of diagnostic reasoning. However, this could result in a loss of efficiency during diagnosis because component C is suspected in even some cases (P_{n+1} above) which are not warranted by its appropriate function f_1.

As an illustration, the switch in a table lamp can be rigged to be constantly on: to behave like a piece of wire, a *conveyor* of electricity. But, assigning to the switch its most specialized/subsumed function of *control* ensures correctness of diagnosis even when it is only used as a *conveyor*.

Recall that a component can assume different functions in different environments, which are generalizations or specializations of each other. Therefore, the principle follows:

Principle 4 *In order to ensure the correctness of reasoning in all environments, the function statically assigned to a component's model must be the most specialized function of the component.*

Again, recall that the most specialized function of a component is its function with respect to its ports.

Although our explanation is specific to diagnosis which is a discriminatory application of function [Kumar, 1994a], the above principle applies to motivational and explanatory applications as well. Consider design, which is a motivational application of function. If f_1 (as defined above) is assigned to the model of a component whose most specialized function is f_2, the component will be considered inadequate for a situation where function f_2 is required, and will not be used. On the other hand, if f_2 is assigned to the component, the function required during design is f_1, and the component can be configured to serve f_1, it can still be considered/included, although at the expense of cost of design. Therefore, in order to ensure that a component is not eliminated from consideration incorrectly during design, it must be assigned its most specialized function. The case of explanatory application of function (such as explanation generation) may be argued similar to diagnosis.

To summarize:

1. From Observation 1, it is clear that a component can assume only those functions which are generalizations of its function with respect to its ports.

2. From Principle 4, it is clear that assigning the most specialized function to a component, i.e., the function of the component with respect to its ports, will ensure correctness of reasoning even when the component's function in the current environment is more generalized.

Therefore, in component ontological representation, function is assigned to a component statically with respect to its ports rather than dynamically with respect to its environments without any loss of correctness of reasoning, and in spite of the context sensitive nature of function.

3.2 Classes

In this section, the computational model of Classes is proposed as a device and domain independent representation of function based on component ontology principles. In the computational model, function primitives called classes are defined for both components and subsystems.

Causal interactions in a device may be expressed in terms of *effort* for and *flow* of energy, matter and information in the device [Paynter, 1961]. For instance, electricity in digital devices, gasoline, air and electricity in automobile engines, and heat in heaters are all flows. Heat, gasoline, electricity and air are flows of energy or matter in different forms. Each such form is termed an *element*.

An *intended element* in a device is an element which participates in the behavior of the device by design. An *unintended element* is a side-effect: it does not contribute to the behavior for which the device is designed. Gasoline, air and electricity are some intended elements in an automobile engine. Heat is an unintended element in it. Note that, although an unintended element does not participate in the behavior of the device, it may still contribute to symptoms in the device by side-effect.

A *signal* is defined as a causal flow of energy, matter or information in a device. A signal could flow along several paths from the device's inputs towards its outputs. Therefore, several signals may exist in a device.

A *signal-line* is defined as the sequence of components along the path from the origin to the use of a signal. Each component in the path causally contributes to the signal. Signal-lines in a device represent subsystems in it, and are logical entities. They are usually linear in structure. Signal-lines may share components. A component may belong to different signal-lines with respect to its different ports.

For instance, the signal-line for fuel flow in an automobile is the sequence of components: fuel tank, fuel line, fuel pump, fuel filter and carburetor. A *unit* is the set of components in a signal-line. The unit of the fuel flow signal-line is the set of all the above-mentioned components.

By definition, a signal is restricted to only one element in a device. Signal-lines with different elements may share components too. E.g., a power transistor may belong to both electrical and thermal signal-lines in an electronic device.

3.2.1 Classes of Devices/Components

Following Principle 1, class of a component is defined with respect to its ports. The class of a component with respect to its ports is also the function of the component with respect to the signal-lines connected to those ports.

Definition 2 *The class of a device/component with respect to a signal S at its port is defined as follows (See Figure 3.3):*

1. *Producer*: produces the signal S. Its function is

   $$f_p(x) = S$$

   where either $x = \emptyset$, i.e., input is non-existent or the element of x is different from that of S.

   E.g., solar cells produce electrical signal from solar energy input. Similarly, power cube in a digital device, and battery in the electrical unit of an automobile are producers.

2. *Consumer*: consumes the signal S. Its function is

   $$f_t(S) = y$$

   where either $y = \emptyset$, i.e., the device has no output corresponding to the input signal S or the element of output y is different from that of S. In the latter case, consumer transduces between different elements.

   E.g., acoustic speakers consume electrical signal to produce sound. Bulbs and automobile wheels are consumers of electrical and torque signals respectively.

3. *Control*: transforms the signal S. Its function is

   $$f_c(S, C) = S'$$

   where transformation from S to S' is dependent on input C. However, C is optional. If it exists, the component is a consumer with respect to it.

E.g., transistor amplifies an electrical signal. NOT gate transforms a signal from level-one to level-zero. Choke in the automobile carburetor regulates the volume of air flow into the carburetor based on input from the thermostat.

4. *Address*: chooses among several signal-lines Y_i for an input signal S or among several input signals S_i for a signal-line Y. The chosen signal is passed unchanged. Address component's function may be written as either of the following (or a combination of both):

$$f_a(S, A) = S = Y_i$$

$$f_a(S_i, A) = Y = S_i$$

   where Y is a signal at the output and i is some function of the address input A. However, input A is optional, and the component is a consumer with respect to it.

   E.g., multiplexers and demultiplexers in a digital electronic device. Distributor in the automobile ignition unit selects one of several cylinders to deliver sparks.

5. *Data*: conveys the signal S unchanged. Its function is

$$f_d(S) = S$$

   E.g., wire in a digital device conveys electricity and fuel line in the automobile fuel unit conveys fuel unchanged. Data may be further classified as follows:

   (a) *Ideal*, i.e., involving no dissipation.

   (b) *Non-Ideal*, i.e., involving dissipation.

6. *Store*: behaves as a repository for the signal S until it is retrieved. Its function is

$$f_s(S_{t_o}) = S_{t_o+t}$$

   for some time instant t_o and elapsed time $t \geq 0$. It may have an optional control input which assists in storing the signal S, i.e., reading and writing, or saving and releasing the signal.

   Store may be characterized as data with an associated time delay. E.g., memory chips are store components.

Note that, functions are treated as abstractions of behavior in the above definitions of classes. These abstractions may be modeled as in [Giunchiglia and Walsh, 1992]. E.g., in a NOT gate which is a control component, $S' = \overline{S}$ and $C = \emptyset$. (These classes are illustrated with the example of a door bell in Section 3.2.3.)

It is not required that all the above classes of components occur in a unit. However, all the components in a unit must be classifiable as belonging to at least one of the above classes.

A component could belong to many classes with respect to its different inputs and outputs, as seen in the definitions of address and control classes. A component could also belong to a combination of classes with respect to the same input and/or output. For instance, memory chips are a combination of store and address (to select the appropriate store cell) classes.

Note that, in the above definitions, control is the only class which modifies its output. This is too restrictive. Some specializations of control class which may also be considered are:

- *Attenuator*: reduces the magnitude of its input signal.
- *Amplifier*: increases the magnitude of its input signal.
- *Switch*: conveys or inhibits its input signal depending on its control signal.
- *Modulator*: modifies some attribute (such as magnitude, frequency) of its input signal according to its control signal.
- *Demodulator*: extracts variations in some attribute (magnitude, frequency) of its input signal.

3.2.2 Classes of Signal-lines

Following Principle 2, class of a signal-line is defined with respect to the components to which it provides input. It is defined as the function of the port(s) of the component(s) to which it is connected.

Definition 3 *The class of a signal-line connected to the input of a device/component D is defined with respect to D. Let x be an input, y an output and b the behavior of device D (See Figure 3.3):*

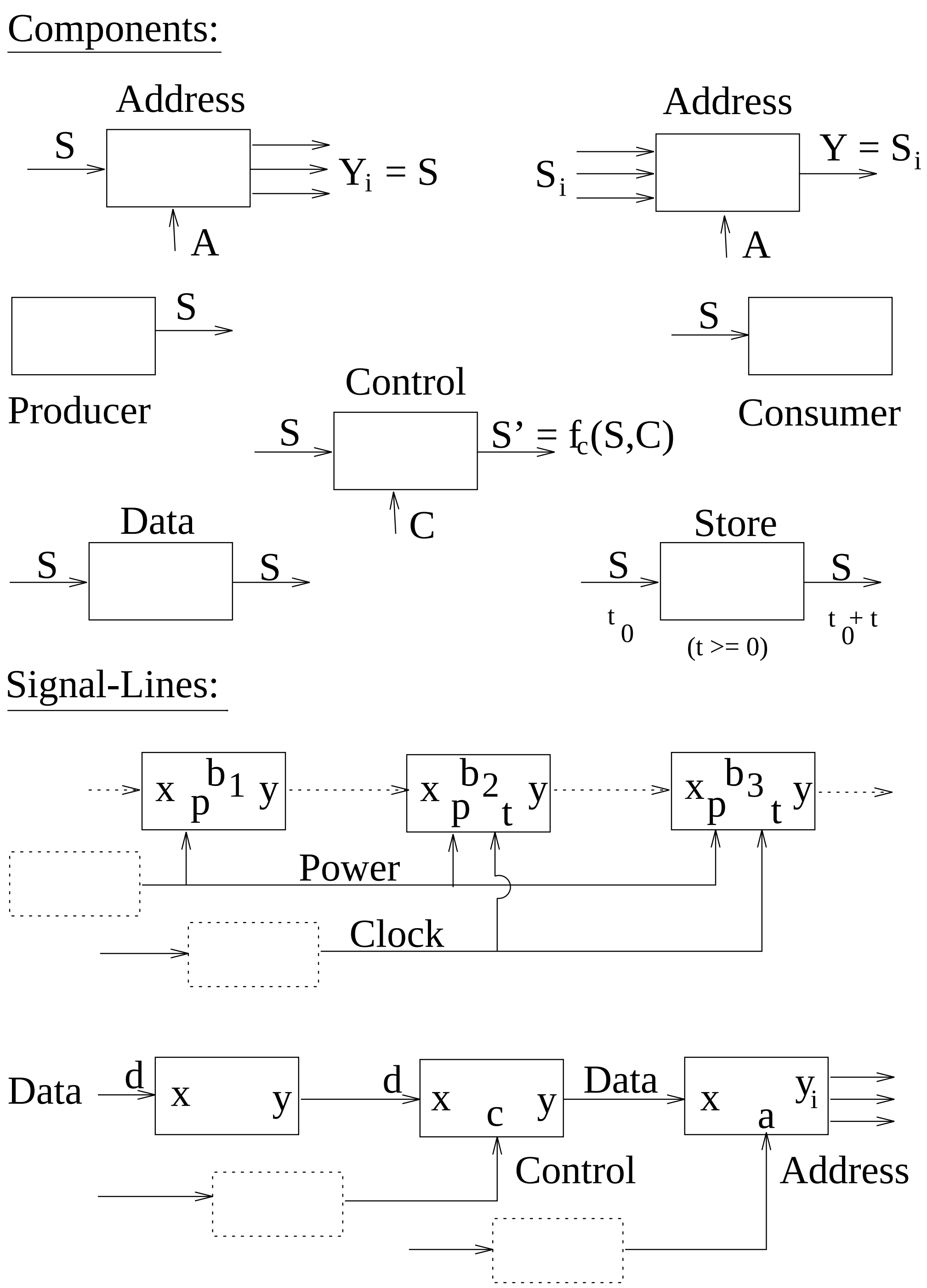


Figure 3.3: Classes of Components and Signal-lines

1. *Power* signal-line p is the input necessary to realize the behavior b of the device, i.e., $(p = 0) \Rightarrow b$ is not realized and y cannot be observed. However, p is not output by D modified or otherwise. E.g., power cord of an electrical device is part of its power signal-line.

2. *Clock* signal-line t is the input used by D to delay, time or synchronize the output y. The behavior of D may be written as: $b = b'(t)$, i.e., b is some function of the temporal input t. The input t is not output by D modified or otherwise. E.g., timer in a toaster is part of its clock signal-line.

3. *Control* signal-line c provides control input to a control component, i.e., the control component D obtains its output y by modifying its input x according to input c. In other words, $y = b(x, c)$ and $\frac{y}{x}$ is in some way proportional to c. E.g., idle speed screw in an automobile carburetor is part of the control signal-line connected to its throttle.

4. *Address* signal-line a provides address input to an address component, i.e., the address component D selects its output y from among its several inputs x_i or channels its input x to one of its several outputs y_i based on input a. In other words, $b(x_i, a) = y = x_i$ or $b(x, a) = x = y_i$, where i is some function of a. In an automobile distributor, electronics controlling the firing sequence is part of the address signal-line.

5. *Data* signal-line d is the input that D was *designed* to process (to control, address, store, etc.), i.e., $d = x$ itself. y is the corresponding data output of D. E.g., fuel-air mixture is one of the data signal-lines connected to an automobile engine.

A signal-line could belong to different classes with respect to different components. For instance, the same signal-line could provide data input to device $D1$ and control input to device $D2$. Note that, data is the only input which may be passed on through a corresponding output of the device D. All the other inputs are consumed by the device. Data inputs/outputs are the ones chosen by an address component, based on its address input. Data input is the one modified by a control component, based on its control input. Data input is the one which is conveyed unmodified, by store and data components to their data outputs. Data input and output are not directly affected by power and clock inputs. However, power enables D to process its data input and clock times this processing.

The above classes of signal-lines may be categorized as follows:

1. Active classes are those that enable device D to operate, and hence, contribute "effort":
   (a) Power provides spatial effort
   (b) Clock provides temporal effort
2. Passive classes are those that provide "flows" which are processed by the device:
   (a) Primary class provides the flow which is processed by components in the signal-line:
      i. Data
   (b) Secondary class provides the flow which is used to process some other flow:
      i. Address
      ii. Control

In components with state, state is treated as a pseudo-input. It is a data input/signal-line with respect to the component, which is in turn a data component with respect to it.

3.2.3 Illustration: The Door Bell

The door bell [Milne, 1985] is a suitable example to illustrate Classes. Operation of the door bell shown in Figure 3.4 is as follows:

When the switch is pressed, the electric circuit is completed through contact, spring and the electromagnet. The electromagnet is magnetized and it attracts the armature. As the armature moves towards the electromagnet, the hammer strikes against the gong to produce sound. However, when the armature moves towards the electromagnet, the spring separates from the contact, and the circuit is broken. The electromagnet is no longer magnetized. The armature-hammer assembly rebounds due to the spring, thereby re-connecting the contact with the spring. As long as the switch is held down, the above sequence repeats, resulting in an on-again-off-again sound.

It may be noted that the door bell has several elements in it: electrical, magnetic, mechanical and acoustic. Signal-lines in electrical and mechanical elements are modeled as shown in Figure 3.4. Magnetic and acoustic elements are ignored since faults in these elements require material defects.

In the electrical signal-line, contact, spring and wires convey electrical signal and are hence, data components. Battery produces electrical signal and is hence, the producer. Its input is chemical, which is ignored in the present illustration. The electromagnet consumes the signal, its output being in magnetic flow. The switch regulates the signal based on manual input and is hence, control.

In the mechanical signal-line, electromagnet is the producer. It produces motion by attracting the armature when magnetized. The spring transforms linear motion into vibration and is control. Its data input is mechanical motion and its output is motion modified to vibration. Its control input is the material resilience of the spring. Gong and contact are consumers of motion since they stop the swing developed in the hammer. Hammer and armature are data components which convey motion unaltered.

There are three other signal-lines in the device, which are not of interest to diagnosis:

1. The magnetic signal-line with the electromagnet as producer and the armature as consumer. Since malfunction in this signal-line requires a change in the material composition of the electromagnet and/or armature, it is ignored.

2. The acoustic signal from the bell to the listener. Bell is the producer of sound by impact of hammer on gong. The diagnostician or user is its consumer.

3. The mechanical (motion) signal-line with the diagnostician's or user's hand as producer and the switch as consumer. Note that, if diagnosis yields the switch as the prime suspect, this signal-line may be examined after taking into account a more detailed model of the switch.

3.3 Discussion of Classes

Lemma 1 Universality of Data Signal-line: *Every signal-line in and by itself is a data signal-line.*

Proof: Without loss of generality, suppose a signal-line S_1 provides control input to a component C_1 in a device D. By definition, C_1 is also a consumer of the signal S carried by S_1. Each of the components in S_1 (other than C_1 and any producer) must have an input and a corresponding output port connected to the signal-line S_1.

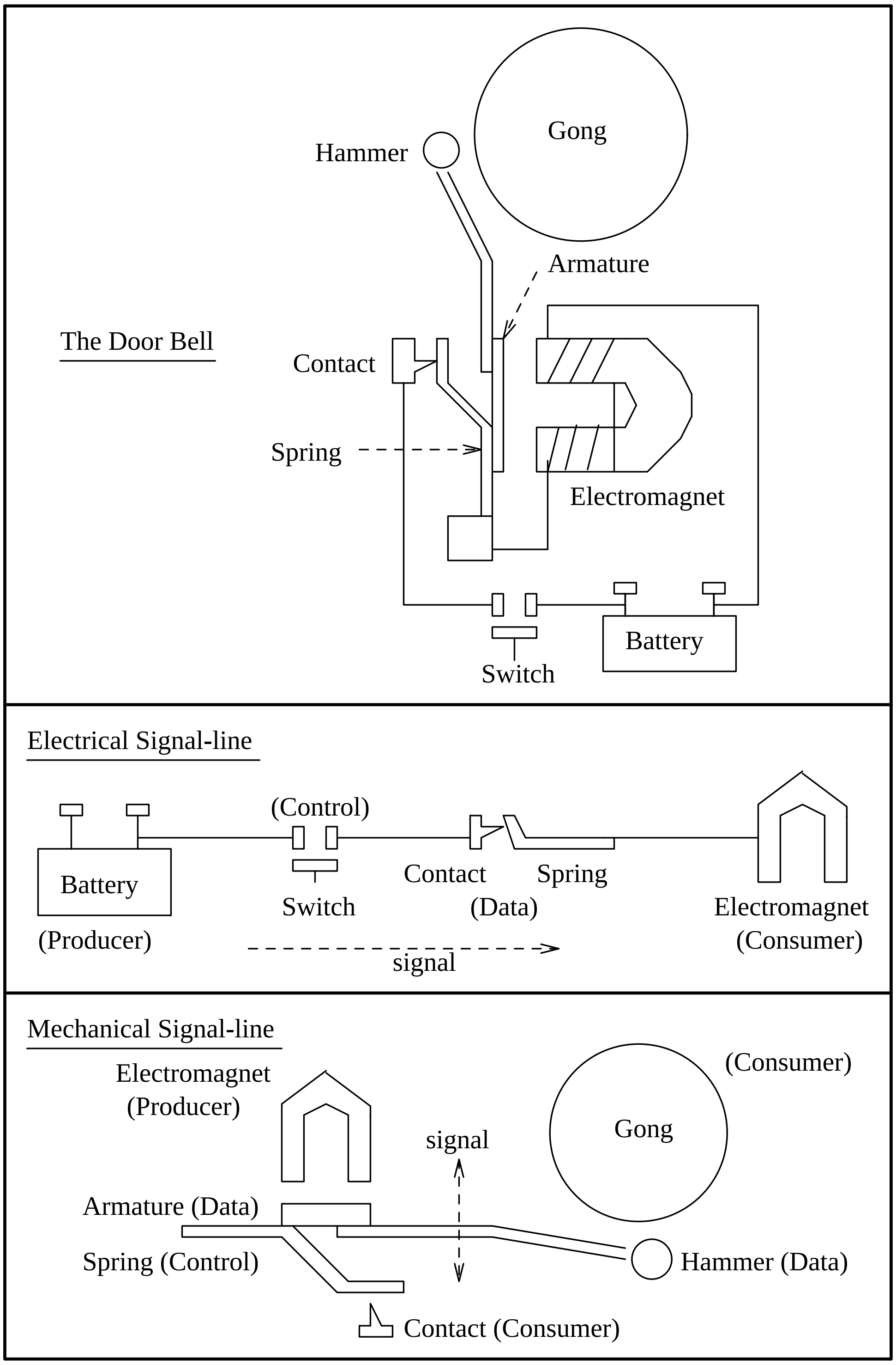


Figure 3.4: The Door Bell and its Class Model

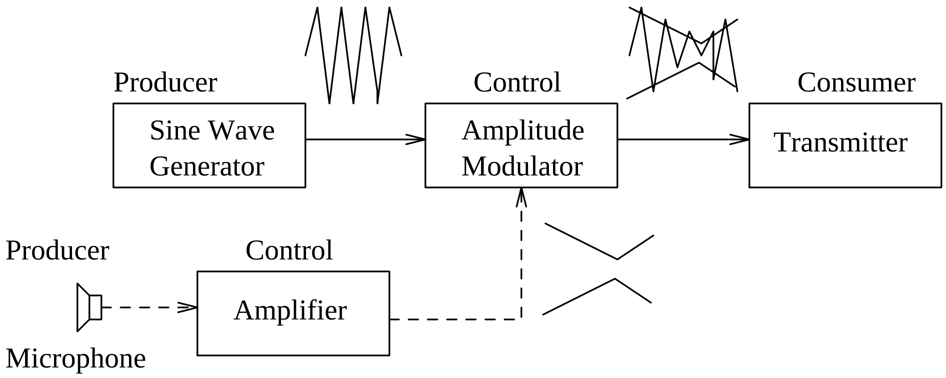


Figure 3.5: Amplitude Modulator System

Therefore, signal S provides data input to each of these components. It follows that, S_1 is a data signal-line with respect to these components, although it is a control signal-line with respect to C_1. It is also trivially a data signal-line with respect to any producers and some consumers (E.g., mechanical signal-line with respect to gong in the door bell) in its unit. Hence:

Corollary 1 *A signal-line is a data signal-line with respect to all the components in its unit except some consumers.*

Consider the amplitude modulator system in Figure 3.5. It consists of five components, viz., a sine-wave generator, amplitude modulator, transmitter, amplifier and a microphone. The sine-wave generator, amplitude modulator and transmitter constitute a signal-line S_1 corresponding to the carrier signal. The microphone, amplifier and amplitude modulator constitute a signal-line S_2 corresponding to the information signal. With respect to S_1, the sine-wave generator is a producer, amplitude modulator is control and transmitter is a consumer. Input from the sine-wave generator is a data input of the amplitude modulator and the output to transmitter is its corresponding data output. Input from the amplifier is its control input. Therefore, S_2 is a control signal-line with respect to the modulator. However, considered in isolation of S_1, S_2 is itself a data signal-line with the microphone as producer, amplifier as control and modulator as the consumer. The universality of data signal-line implies that, it is sufficient to be able to diagnose data signal-lines.

The definitions of *classes of signal-lines* and *classes of components* are *duals* of each other. Classes of components are identified with respect to signal-lines, and classes of signal-lines are identified with respect to components. The dual perspectives are used to represent and reason about functions of devices systematically within a model based paradigm. Function model of a device is built by identifying every signal-line in its structural model, and categorizing each component/signal-line into classes as defined above. During diagnosis, given a symptom in a device, a subset of signal-lines in it (or connected to it) is suspected, and given a symptom in a signal-line, a subset of components in it is suspected based on its class.

Representation in the computational model of Classes may be summarized as follows:

- Subsystems in a device are treated as signal-lines;
- Function primitives called classes are defined for both components and signal-lines;
- Dualism between classes of components and signal-lines enables systematic traversal of a device during representation and reasoning;
- Every signal-line is a data signal-line.

The class model "chunks" or fragments the structure model into modular causal substructures called signal-lines. Considering one signal-line at a time helps focus diagnosis. Whereas hierarchical representation traditionally used in model based reasoning improves the focus of reasoning by suppressing unnecessary structure, class representation does so by suppressing irrelevant behavior. Therefore, the representation per se offers advantages for use in diagnostic reasoning. In Chapter 5, discrimination techniques are proposed, which complement the support of Classes for diagnosis: they help navigate through a class model during diagnosis.

The proposed function primitives are not claimed to be comprehensive or necessary in all domains. Our objective is to demonstrate the merit of component-ontological representation using Classes rather than to promote a particular choice of function primitives. Any suitable choice of domain-dependent primitives may be plugged into the above computational model and used for function based reasoning [3]. It is important however, to identify a reasonable number of primitives in order to take advantage of Classes for diagnosis. Note that, the primitives proposed above are adequate for digital electronics domain.

[3] An excellent source of function primitives is the book "Living Systems" by J.G. Miller [Miller, 1978]. Among the primitives proposed by the author for processing matter and energy are: distributor, converter, producer, storage, reproducer, boundary, ingestor, extruder, motor and supporter.

3.3.1 Tractability/Efficiency

If a device D has m inputs and n outputs, the space complexity of its function model is $O(m+n)$, i.e., the complexity is linear in the number of input/output ports. This is the complexity of class models of all atomic components as well as components at the *floor* level of representation [Kumar and Upadhyaya, 1990] of a device. It compares favorably with state based representations, which can be potentially intractable because a device can have exponential number of states.

If a composite device D' has m' inputs, n' outputs and p components in it, an estimate on the upper bound of the number of signal-lines in it is given by:

$$z + kp$$

where k is a constant corresponding to the number of signal-lines (other than data) connected to each of the p components. The value of k is determined by the number of classes defined for signal-lines. $k = 4$ for the set of classes proposed in this dissertation. The value of z depends on the topology of the device. If the device has:

- neither fan-ins nor fan-outs, $z = m' + n'$.
- only fan-ins or fan-outs, $z = m'n'$.
- reconvergent fan-outs, z is potentially exponential in p. However, in practice, at least one component is unique to each signal-line because:
  - Due to subsumption in the direction of causality argued in Figure 3.2, it is unlikely (except in trivial cases) that two signal-lines will share the same set of components (units), but not their sequence.
  - If two signal-lines share the same sequence of components, they can most likely be merged to form a single signal-line.

  Therefore, $z = m'n'p$.

Hence, the number of signal-lines in a complex device is linear in the number of its components. This means that the space complexity of its function model is also linear, and not exponential as in the case of state and process based representations.

The **DS** algorithm proposed in Section 4.1.1 identifies a complete signal-line in time which is linear in the length of the signal-line. The **DR** algorithm developed in Section 4.1.2 builds the function model of a complex device by calling the **DS** algorithm as many times as there are signal-lines in the device, which is linear in the number of components. Therefore, the time required to build the function model of a complex device from those of its components is linear in the number of its components. This compares favorably with state based representations which can take exponential time to build, because of potentially exponential number of states in a device.

A summary of the advantages of component ontological representation and hence, class models vis-a-vis state and process ontological representations is as follows:

- The space complexity of representation of an atomic component is linear in the number of ports of the component. The space complexity of representation of a complex device is linear in the number of components in the device. Hence, the representation is space-wise efficient.

- Due to compositional nature, the component ontological function model of a complex device can be constructed by composing the function models of its components. As discussed above, this can be done in linear time. Hence, representation in Classes is time-wise efficient too. Further, this process can be automated (as discussed in Chapter 4) to ensure the fidelity of representation.

- Function model of an atomic component is defined in isolation of its environment as argued in Section 3.1. Hence, it can be compiled and stored in device libraries. It need not be built afresh every time it is used; it can be re-used to build function models of complex devices by composition. This results in considerable savings of time and effort during device representation.

- Component ontological function model of a device is fixed with respect to its ports and is determined from its design. It is not subjective, and will not vary from user to user. Hence, the competency of component ontological representation is easy to evaluate.

In comparison, state and process ontological representations are potentially exponential in space and time complexity (based on the number of possible states/processes), have to be built from scratch on demand (and not statically compiled or composed), are subjective, and cannot be automated or re-used.

Class representation meets the four requirements of knowledge representation [Rich and Knight, 1991] as follows:

- Representation Adequacy: It is adequate for diagnosis. The representation of function in terms of ports highlights the difference in the behaviors of the various ports with respect to their device. This perspective of function is suitable for candidate ordering. On the other hand, it is not suitable for simulation as are state based representations [Sticklen and Chandrasekaran, 1989, Vescovi *et al.*, 1993].

- Representational Efficiency: It is efficient because it has linear space and time complexity.
- Inferential Adequacy: It is adequate for diagnosis because it enables candidate discrimination based on function, as will be discussed in Chapter 5.
- Acquisitional Efficiency: Class models of atomic devices can be built from abstractions of their behaviors. These models can be compiled into device libraries for future re-use. Class models of complex devices can be built by composing the class models of their components. This process can be automated as demonstrated in the next chapter. Hence, class representation is acquisitionally efficient.

3.3.2 Scalability

Practical devices are often transdomain devices, i.e., they incorporate several elements. Since Classes is based on the concepts of flow and effort [Paynter, 1961], it can be applied to any domain. More interestingly, it can be used to represent transdomain devices (E.g., the door bell in Figure 3.4) without any switch in notation between domains. This compares favorably with the multiple models previously proposed to represent transdomain devices [Struss, 1988, Gallanti *et al.*, 1989, Weld, 1990, Liu and Farley, 1990, Bandekar, 1991, Brajnik *et al.*, 1991], because Classes is unified and systematic.

Practical devices often involve side-effects also, i.e., causal interactions that were not provided for in their design. Representation of side-effects is necessary for complete diagnosis [Davis, 1984]. The function model proposed by Classes can be used to model side-effects in devices at the functional level. Therefore, Classes provides a means to completely represent a device, while at the same time holding down the complexity of representation by virtue of its uniformity. It does not tradeoff complexity for completeness. The following result specifies how side-effects, i.e., unintended elements in a device must be modeled in terms of its intended elements in order to facilitate diagnosis.

Lemma 2 *In a device D with the set of elements $E = E_U \bigcup E_N$ (E_U being the subset of unintended elements and E_N the subset of intended elements), the behavior of $E_u \in E_U$ must be modeled in terms of all $E_n \in \Gamma_u$, where*

$\Gamma_u = \{E_i \mid E_i \in E_N \bigwedge U_u \bigcap U_i \neq \emptyset\}$

and U_u and U_i are the sets of components in the signal-lines of the elements E_u and E_i respectively.

For instance, an electromechanical device might have:

$E_N = \{$ Electrical, Mechanical $\}$

$E_U = \{$ Thermal, Electromagnetic $\}$

The thermal element must be modeled in terms of every element $E_n \in E_N$, with which it shares at least some components, i.e.,

$U_{thermal} \bigcap U_n \neq \emptyset$

In this case, the thermal element shares a power transistor with the electrical element and hence, thermal side-effect must be modeled in terms of electrical components. However, if it does not share any component with the mechanical element, it need not be modeled in terms of mechanical components.

Therefore, a consequence of domain-independence of Classes is that, it is systematic and uniform in handling both transdomain devices and devices with side-effects. Hence, it is scalable to representation of complex devices.

The class model of a device is its structure model with function annotations. Hence, hierarchical structure organization solves the problem of scaling the function representation to large devices. Moreover, the function model of a complex device can be generated automatically from those of its components. Hence, Classes is scalable to representation of large devices.

3.4 Summary

In this chapter, the principles behind component ontological representation of function have been enunciated. These principles provide the basis for defining the function of a component independent of its environment. The resulting function model has been proved to ensure correctness of reasoning. Component ontological representation is composable, a trait *required* of models used in model based systems.

Classes has been proposed as a computation model for function based on component ontology principles. As part of the computation model, function primitives called classes have been proposed for components as well as subsystems (signal-lines). The claims of linear complexity and scalability of class representation have been justified. Class representation facilitates model based diagnosis.

In the next chapter, composability of Classes is demonstrated by developing algorithms to build the class model of a complex device from those of its components.

Chapter 4

Automating Representation

As discussed in Chapter 3, one advantage of using component ontological representation is that it is composable, i.e., function model of a device can be built by composing the models of its components. Furthermore, this process can be automated. As a result of composability and the ability to automate representation:

1. It is sufficient to build and ensure the correctness of function models of components at the floor level of representation [Kumar and Upadhyaya, 1990]. Since function models of complex devices, which are harder to build and verify, are systematically composed from these models, their fidelity is ensured.

2. Considerable effort and time are saved during representation of complex devices. Further, function models of atomic components can be compiled into device libraries and re-used, thereby obviating any duplication of effort.

Hence, the ability to automate representation ensures the fidelity, and improves the efficiency of representation. It also improves the scalability of representation.

In this section, algorithms are developed to automate function representation. In Section 4.1, deterministic algorithms are described, which build the class model of a device from the class models of its components. In Section 4.2, heuristic algorithms are provided, which do not assume the availability of class models of components. They build the function model of a device from its structure.

Both the algorithms assume that, prior to their application, principles of structural composition [Hamscher, 1988], viz., feedback loop encapsulation and reconvergent signal simplification are applied to the device. To begin with, a few necessary definitions:

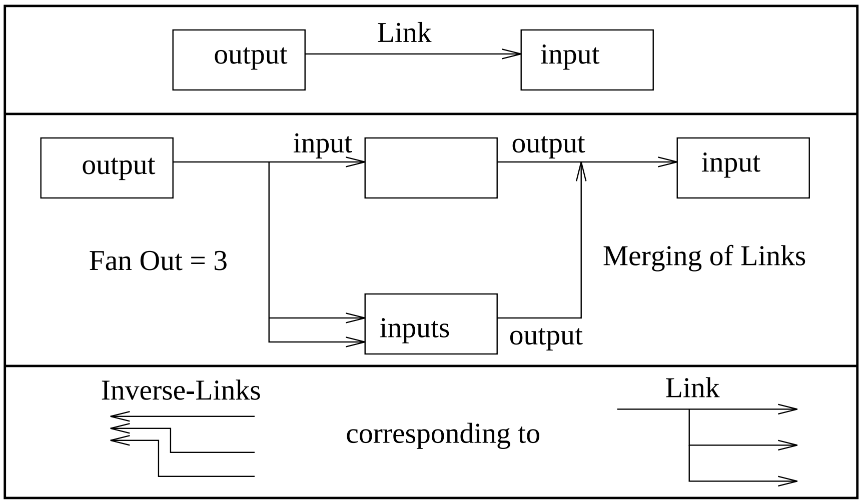


Figure 4.1: Links and Inverse-links

Definition 4 *A complete signal-line in a device is a signal-line which:*

1. originates with the producer of the signal or some device input through which the signal is obtained;

2. terminates in a consumer of the signal or some device output through which the signal is passed out;

3. includes all the intermediate components. From Lemma 1, it provides data input to each of these components, and is a data signal-line with respect to them.

Definition 5 *A link is a directed connection between the output of a component and the input of an immediately causally following component in a device.*

In this discussion, inputs of a device are considered *virtual outputs*, and its outputs are considered *virtual inputs*. A link may fan out (See Figure 4.1). A fan-in at the input of a component is termed *merging of links*. Links have been defined in the direction of causal interaction.

Definition 6 *An inverse-link in a device is a directed arc from the input of a component to the output of an immediately causally preceding component in the device.*

Several inverse-links may combine to form a link in a device (See Figure 4.1).

The structure of a device D is given by

$$D = \ < C, L >$$

where $C = \{D_1, \ldots, D_\alpha\}$ is the set of components in D and $L = \{L_1, \ldots, L_\beta\}$ is the set of inverse-links in it at a given level of structural hierarchy.

4.1 Deterministic Algorithms

Two algorithms are developed in this section: the Deterministic-Signal-line (**DS**) algorithm to identify complete signal-lines in a device, and the Deterministic-Representation (**DR**) algorithm to build the class model of a device using DS algorithm. Both the algorithms assume that class models of components at the floor level of representation [Kumar and Upadhyaya, 1990] are provided. For instance, the floor level of representation for a digital electronic device could be at the level of its gates, chips or boards. If the floor level is at a higher level than the level of atomic components, **DS** and **DR** algorithms may also be used to build the class models of devices at the floor level from the class models of atomic components.

Class model of an atomic component is built by abstraction, from its behavior model. Abstractions of behavior expressions of the component may be generated as discussed in [Giunchiglia and Walsh, 1992]. These abstractions are matched with definitions of classes provided in Section 3.2, to label:

1. the component into classes with respect to its ports;

2. each port into a class with respect to the component.

4.1.1 Identifying a Complete Signal-line

A complete signal-line is identified by beginning with its terminating point, and tracing it against the direction of causality.

Algorithm 1 (DS): *The deterministic algorithm to identify a complete signal-line in a device, terminating at a point O in the device is as follows:*

1. Start with an inverse-link corresponding to one of the links merging at O. Include it in the signal-line.

2. Let the inverse-link be connected to the data output O_1 of a component D_1. Include D_1 in the signal-line.

3. Repeat Steps 1 and 2 starting with the data input in D_1 corresponding to data output O_1. Note that this correspondence is provided as part of the class model of D_1. A complete signal-line is identified in either of the following cases:

   (a) No inverse-link is found in Step 1 because the last component included in Step 2 is the producer of the signal, and it has no data input corresponding to output O_1.

   (b) No component is found in Step 2 because the last inverse-link included in Step 1 is connected to an input of the device (virtual output).

Theorem 1 *Algorithm 1 is correct, i.e., it identifies a complete signal-line ending at point O in the device.*

Proof: The algorithm terminates only when it reaches a producer or a device input. It considers only those components to which it provides data input/output. Therefore, if the point O is a device output or a consumer, from the definition of a complete signal-line, it is clear that the algorithm is correct.

Note that, the device is finite and all the loops in it have been encapsulated. Therefore, the algorithm terminates by reaching a producer or a device input. It considers each and only the components in the signal-line being constructed. Hence, it takes time linear in the number of components in the signal-line.

4.1.2 Building the Class Model

Algorithm 2 (DR): *The algorithm to build the class model of a device D, given the class models of its components $D_1, \ldots, D_\alpha$ is as follows:*

1. Let the outputs of the device D be given by the list $O = \{O_1, \ldots, O_n\}$.

2. Starting with each $O_i \in O$, identify all possible complete signal-lines using **DS** Algorithm. In Step 1 of **DS** Algorithm, recursively identify signal-lines starting with each merging link. Let the resulting set of signal-lines be: $S = \{S_1, \ldots, S_m\}$, where $m \geq n$.

3. Identify the unit U_i of each signal-line $S_i \in S$. Let unit U_i have p_i components in it, i.e., $U_i = \{D_1^i, \ldots, D_{p_i}^i\}$. With respect to each component D_j^i:

(a) If the component has power and clock inputs which have not already been modeled, recursively apply the algorithm from Step 2 onwards for these inputs.

(b) If D^i_j is an address component with respect to the signal-line S_i, let i^i_j and o^i_j be its data input and output corresponding to S_i. Recursively apply the algorithm from Step 2 onwards for all the address inputs of D^i_j that have not already been modeled with respect to i^i_j/o^i_j. Similarly, if D^i_j is a control or store component, recursively apply the algorithm for all of its control inputs.

4. For each component D^i_j which is not at the floor level of D, recursively apply the algorithm at the component's next lower level of structural hierarchy.

Note that Step 2 models all the data signal-lines in the device. Step 3 models the signal-lines other than data, connected to each component in the device.

Theorem 2 *Algorithm 2 is complete, i.e., it builds a complete class model of the device D.*

Proof: From Step 4, it suffices to prove that the algorithm completely represents a device at one level of structural hierarchy. From Step 3, whenever a signal-line is identified, all the components in its unit are modeled. Further, power, clock, address and control signal-lines connected to these components are also modeled. Therefore, it suffices to prove that all complete signal-lines in the device are identified. In Step 2, all the signal-lines terminating in a device output are identified. In Step 3, all the signal-lines terminating in a consumer component are also identified. Hence, this is true.

DR algorithm has two levels of recursion: one between levels of structure hierarchy (Step 4), and one at a given level of hierarchy (Steps 2 and 3). The recursion between structure levels terminates upon reaching atomic components. The recursion at a given level terminates upon exhausting all complete signal-lines (Step 2), and modeling all the signal-lines connected to each component (Step 3). Since the number of components and signal-lines in a device is finite, the algorithm terminates.

Representation of function of a complex device can be automated using **DS** and **DR** algorithms. **DR** algorithm calls **DS** algorithm (which has a complexity linear in the number of components in a signal-line) for every complete signal-line in the device. The number of complete signal-lines in a device is linear in the number of components

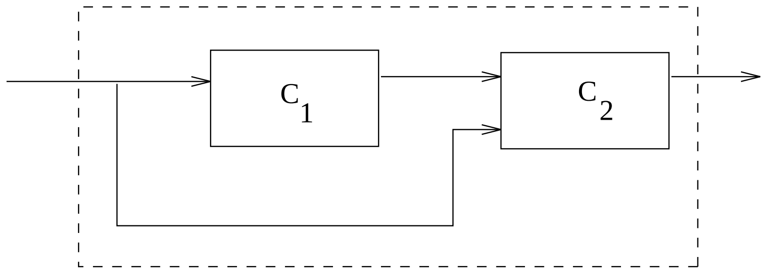


Figure 4.2: Data Signal-line with Fan-out, Providing Input to two Components

in the device, as seen in Section 3.3.1. Therefore, **DR** algorithm takes time linear in the number of components in the device. Hence, automated representation of complex devices can be carried out in linear time using Classes. An illustration of using deterministic algorithms to build the class model of the Display Unit of a printer buffer is provided in Section 6.3.1.

4.2 Heuristic Algorithms

Two algorithms are developed in this section: the Heuristic-Signal-line (**HS**) algorithm to identify complete signal-lines in a device, and the Heuristic-Representation (**HR**) algorithm to build the class model of a device using **HS** algorithm. These algorithms do not assume availability of class models of components. They are based on device structure, and assume that inputs and outputs of components have been identified. They are based on the following heuristic rule:

Heuristic Rule 1 *Causal links in a data signal-line do not fan out. If they do, they do not provide inputs to more than one component in the signal-line.*

The topology of data signal-lines precluded by the above heuristic rule is shown in Figure 4.2. When such topology occurs in a device, it can be encapsulated as shown by the dotted lines in the figure before applying the following heuristic algorithms.

4.2.1 Identifying a Signal-line in a Device

Identifying a complete signal-line terminating at a point in a device is equivalent to finding a path from the root to a leaf in a directed (incomplete and possibly cyclic) tree T built with the terminating point as its root, components and links as nodes, and producers and device-inputs as its leaf nodes.

In the following algorithms, a signal-line is represented as a list of links and components. The notation "a $\in$ b" means that a is an element of list b, and $[a|b]$ means that element a is appended to the list b. The function $in(D)$ yields the set of all the inputs of a device D.

Algorithm 3 (HS): *The heuristic algorithm to identify signal-lines in a device D, connected to a point O in it is as follows:*

Initialization:

- Let the set of signal-lines in D, $DSLSet = \emptyset$. When the algorithm terminates, it returns a list of complete signal-lines of D terminating in O, in $DSLSet$.
- Let the signal-line currently being identified, $DSL = []$.
- Let the set of inputs of all the components already included in DSL, $DIS = \emptyset$.
- Let relaxation figure $\gamma = 0$. This guides application of Heuristic Rule 1.
- Let $LinkSet =$ the set of all the links ending in O.

The algorithm:

1. Reorder $LinkSet$ in the order of fan-out of each link. If n is the smallest fan-out in $LinkSet$, delete all the links in $LinkSet$ with fan-out greater than $n + \gamma$. Let L_1^{-1} be an inverse-link corresponding to L_1, the first link in the reordered $LinkSet$, such that $L_1 \notin DSL$ (i.e., it has not already been considered), and L_1 is not a branch of any input in DIS except those of the most recently added component (the exception accounts for multiple inputs to a component from the output of a causally preceding component). Now, $DSL = [L_1|DSL]$ and $LinkSet = LinkSet - L_1$.
   - If no such link can be found because the last component to be included does not have any inputs, it is a producer. If so, add DSL to DSLSet, i.e., $DSLSet = DSLSet \bigcup \{DSL\}$ and backtrack to the most recent selection of link, to check if any longer signal-line can be identified.
   - If the last component to be included has inputs, but none of them satisfy both the above criteria, backtrack to the most recent selection of link in order to find a valid signal-line.

2. Let the component at the end of inverse-link L_1^{-1} be $D1$. If $D1 \notin DSL$, then $DSL = [D1|DSL]$ and $DIS = DIS \bigcup in(D1)$.

   - If no such component can be found because L_1^{-1} ends in a virtual output, add DSL to DSLSet, i.e., $DSLSet = DSLSet \bigcup \{DSL\}$ and backtrack to the most recent selection of link. Since the algorithm has confined its search to links with the smallest fan-out and has reached a virtual output, this is a valid signal-line. Backtracking at this stage is to search for any longer signal-line.
   - If L_1^{-1} ends in a component which does not satisfy the above criteria, i.e., the component is already in the signal-line, a loop has been encountered. Backtrack to the latest selection of link in order to find a valid signal-line.

   Otherwise, recurse from Step 1 with $LinkSet$ = links at the inputs of $D1$.

3. When the algorithm terminates, $DSLSet$ is a list of possible signal-lines in D connected to the point O in it.

   If $DSLSet = \emptyset$, i.e., no valid signal-line has been found, increment the relaxation figure γ and repeat the algorithm. This step is carried out until either a valid signal-line is found, or the value of γ equals the maximum fan-out in the device.

Consider the device in Figure 4.3. Suppose a data signal-line is to be identified with respect to its output $Out2$. The link *D7-Out2* or *D11-Out2* may be considered. Suppose *D7-Out2* is chosen. Component *D7* is included in DSL. The links at the inputs of *D7* are: *D6-D7*, *D1-D7* and In3-D7. Links *D1-D7* and *In3-D7* are disregarded in favor of *D6-D7* because of their higher fan-outs. Link *D6-D7* and component *D6* are included in DSL. Again, among the links at the inputs of *D6*, *D1-D6* and *In3-D6* are discarded in favor of *D5-D6* (or *D8-D6*) because of their higher fan-outs. Suppose *D5-D6* and component *D5* are included. Link *In2-D5* completes the signal-line because it begins in virtual output $In2$. Therefore, the first data signal-line generated is *In2-D5-D6-D7-Out2*. Now, the algorithm backtracks to also generate the signal-lines *In2-D8-D6-D7-Out2* and *D9-D10-D11-Out2* for $Out2$.

Since the **HS** algorithm is based on Heuristic Rule 1, it is heuristic. As a result, it is necessary for a domain expert to examine its output and eliminate inappropriate signal-lines. (E.g., Suppose the control input to a control component is not a link with fan-out. The algorithm will incorrectly identify a data signal-line with respect to the control input in addition to identifying one with respect to the data input of the

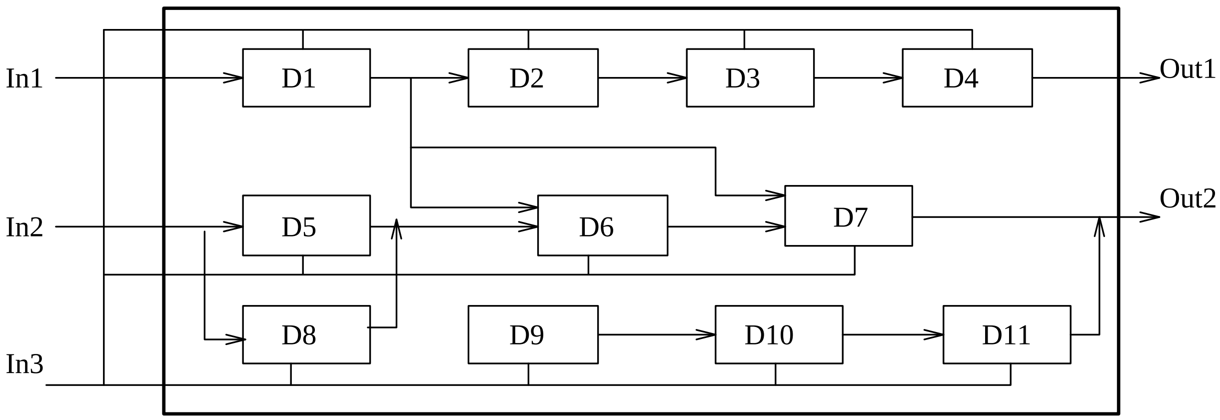


Figure 4.3: A Device with Fan-ins and Fan-outs

control component.) However, the algorithm greatly reduces the labor of identifying signal-lines, and can be gainfully used with human aid, or as human aid.

The **HS** algorithm may be augmented with domain-specific knowledge where necessary. For instance, when considering electrical domain, a complete signal-line must necessarily ensure a closed circuit, whereas in fluid flow, this is not required. Therefore, in the door bell of Figure 3.4, irrespective of whether the switch is to the left or right of the battery, only one complete signal-line should be identified and the switch must be included in it.

4.2.2 Building the Class Model

The following **HR** algorithm builds the class model of a device based on its structure. It uses **HS** algorithm to identify all data signal-lines in the device. The following definition is used to specify its termination condition:

Definition 7 *A trivial signal-line in a device D is one that comprises of either only one link, or a concatenation of links and components such that, no component has a signal-line at its inputs which has not already been modeled.*

Algorithm 4 (HR): *The heuristic algorithm to build the class model of a device $D =< C, L >$ based on its structure is as follows:*

Initialization:

1. Let the set of complete signal-lines already generated for device D, $DSLGen(D) = \emptyset$.

Algorithm:

1. Let the outputs of D be $O = \{O_1, \ldots, O_n\}$. With respect to each $O_i \in O$:

   (a) Obtain $DSLSet$, the list of all possible data signal-lines, using *HS* algorithm. Eliminate any incorrect signal-lines (as discussed before) from $DSLSet$.

   (b) With respect to each link merging at O_i:

   - Sort all the signal-lines identified for it, by decreasing order of their lengths.
   - Consider the signal-lines in the above sorted order: If a signal-line DSL is a proper sublist of some signal-line in $DSLGen(D)$, discard it. Otherwise, add it to $DSLGen(D)$, i.e., $DSLGen(D) = DSLGen(D) \bigcup \{DSL\}$

   Let the set of data signal-lines added to $DSLGen(D)$ be: $S = \{S_1, \ldots, S_m\}$, where $m \geq n$.

2. With respect to each signal-line $S_i \in S$:

   (a) Identify its unit U_i.

   (b) Categorize each component D_j in the unit U_i into classes with respect to S_i.

   (c) With respect to each component D_j:

   i. Perform the substeps of Step 1 for each input $I_k \in \{in(D_j) - I\}$, in order to identify the data signal-lines in D terminating in I_k. (I is the input of D_j connected to S_i.) Let the set of signal-lines generated be S'.

   ii. Categorize each signal-line $S'' \in S'$ into classes with respect to the component D_j.

   iii. If any signal-line S'' is non-*trivial*, recursively apply Step 2 onwards, to the signal-line.

3. Apply the algorithm recursively to each device D_j in each unit U_i, at its next lower level of structural hierarchy.

Note that, Steps 1 and 2 of the algorithm recursively build the class model at a given level of structural hierarchy. Step 3 accounts for multiple levels of hierarchy in the device.

Consider again, the device in Figure 4.3. The signal-line *In1-D1-D2-D3-D4-Out1* is generated with respect to $Out1$. Each of $D1$, $D2$, $D3$ and $D4$ are categorized into classes with respect to the signal-line. The trivial signal-lines *In3-D1*, *In3-D2*, *In3-D3* and *In3-D4* are identified and categorized into classes with respect to $D1$, $D2$, $D3$ and $D4$ respectively. All the three signal-lines generated with respect to $Out2$ (from the previous example) are included in $DSLGen(D)$. With respect to component $D7$, the trivial signal-lines *In1-D1-D7* and *In3-D7* are also identified and categorized into classes. Similarly, signal-lines are identified with respect to the other inputs and components in the device.

Theorem 3 *HR Algorithm is complete, i.e., the output of the algorithm is a complete representation of the device at all levels of structural hierarchy.*

Proof: From Step 3, it suffices to prove this for one level of structural hierarchy. Every component in a device is causally connected to at least one output of the device directly or indirectly. Therefore, generating signal-lines with respect to all the merging links at all the outputs (Step 1) and with respect to all the inputs of all the components (Step 2(c)i) ensures that a complete function model of the device will be built. Human intervention is however, necessary in both Steps 1a and 2b.

4.3 Summary

In this chapter, two sets of algorithms have been developed to build the class model of a complex device from those of its components. Deterministic algorithms assume that class models of components at the floor level of representation [Kumar and Upadhyaya, 1990] are provided. Heuristic algorithms assume that only inputs and outputs of components are identified. Heuristic algorithms try to deduce function from structure, and need human intervention. The correctness and completeness of these algorithms were proved where applicable.

Deterministic and heuristic algorithms demonstrate the composable nature of class representation. Deterministic algorithms may be automated. Automating the representation of function of complex devices ensures the fidelity of the models built, and improves the efficiency and scalability of representation. Implementation of similar algorithms for behavior representation has been reported elsewhere [Kumar and Upadhyaya, 1990]. Techniques to discriminate and diagnose class models of devices are discussed in the next chapter.

Chapter 5

Function Based Diagnosis

In this chapter, function based candidate discrimination is proposed for model based diagnosis. The basis of function based candidate discrimination is *abduction*. Suppose the function of a component P in a device is to ensure a feature Q of the output of the device. P is "responsible" for Q by design. In other words, if Q is not observed, P may or may not be faulty. But, if Q is observed, component P is working. It follows that:

$$Q \longrightarrow P$$

By contraposition,

$$\neg P \longrightarrow \neg Q$$

i.e., if component P is not working, the output feature Q will be incorrect. Now, given that the symptom observed at the output is:

$$\neg Q$$

By abduction, it may be concluded:

$$\neg P$$

i.e., the component P is faulty. For instance, in the amplitude modulator system of Figure 3.5, suppose a symptom is observed in the carrier wave frequency. It is the function of the sine-wave generator to ensure the frequency of the carrier wave. Therefore, the sine-wave generator is suspected before the transmitter or the amplitude modulator.

The objective of this research is to use structure and function for candidate generation and ordering, and behavior for candidate validation. The fault must lie within the cone of influence [Kalme, 1986] of the symptom. Therefore, structure can be used during candidate generation to eliminate the components which are not in the cone of influence. Typically, too many components fall into the cone of influence, and are generated as candidates. Both function and behavior can be used to discriminate among these candidates. As argued in Section 1.3, the components functionally responsible for an output are far fewer than those that behaviorally contribute to it. Therefore, function is better suited to discriminate among candidates. Finally, candidates must be validated. Validation requires quantitative evaluation of the candidates and must be based on behavior.

Structure, behavior, function and empirical rules are the four levels of model based diagnosis proposed by Milne [Milne, 1987]. The work reported in this dissertation encompasses the first three levels.

In this chapter, diagnosis in the computational model of Classes is described. Two device-independent techniques of candidate discrimination are proposed, viz., teleological technique in Section 5.1.1 and default order technique in Section 5.1.2. A model based algorithm is developed in Section 5.2, which applies these discrimination techniques to diagnose the class model of a device. The discrimination techniques and the diagnosis algorithm are evaluated in Section 5.3.

5.1 Discrimination Techniques

Traditionally, the term *discrimination* is used to mean eliminating candidates or ordering future elimination. It is carried out after elaboration (See Figure 1.1) by obtaining measurements across the candidates from the user. In this dissertation, the term *discrimination* is used slightly differently as follows:

- It includes candidate ordering in addition to elimination.
- It is carried out during candidate generation and not after elaboration.

5.1.1 Teleological Technique

In teleological technique, symptoms are expressed as missing features of outputs. Associations between features and classes are specified as part of the device model. Components that belong to the classes associated with features missing in the current symptom are suspected first.

Symptom Expression

Outputs of a device are expressed in terms of *features*. These features comprehensively characterize the outputs and are distinguishable from each other. During diagnosis, the symptom in a device is expressed in terms of missing feature(s).

In this discussion, consider the static *value* and dynamic *transition* (i.e., change) of an output of a device. With respect to these, the following three features are used:

1. *Existence* specifies whether the output is observable. This assumes the feasibility of observing the output if it exists.

2. *Validity* specifies whether the output is within the range of behavior of the device.

3. *Correctness* specifies whether the output is as expected, based on the behavior and current inputs of the device.

Among the above features, existence is observed. Correctness can be computed from inputs and the device behavior. Validity of an output is ascertained by verifying that it is within the valid range of outputs, or by verifying that there is at least one valid input for which the device generates that output:

- If the behavior of the device is invertible, inputs corresponding to the output are computed. The validity of these inputs is verified from the design of the device.

- If the behavior of the device is not invertible:

  - If the range of output values of the device is contiguous (if discrete device) or continuous, only boundary values of the range are stored in the component library. Current output is checked to lie between these boundary values.

  - If device output is monotonic with respect to its inputs, but not contiguous/continuous, binary search is carried out on its input range to locate the input corresponding to the current output.

  - In the worst case, a list of output values of the device is exhaustively computed, and stored in the component library in a sorted order. The current output is checked for membership in this list.

Teleological Technique for Suspect Ordering

Simple, yet powerful heuristic associations between features and classes are used to order suspects. Components belonging to the classes corresponding to missing feature(s) are suspected first.

Following are some sample feature-class associations:

1. Power signal-line and producer are responsible for the existence of value.
2. Store and data are responsible for the validity of value.
3. Address (component and signal-line) and producer are responsible for the correctness of value.
4. Producer is responsible for the existence of transition.
5. Store and control signal-lines are responsible for the validity of transition.
6. Control and producer are responsible for the correctness of transition.

Following are the steps in teleological technique of candidate ordering:

1. The symptom(s) is translated into missing features.
2. Classes responsible for the missing features are determined based on feature-class associations.
3. The list of components/signal-lines is reordered, with components/signal-lines belonging to the above classes placed at the top of the list.

Generation of Teleological Associations

Feature-class associations are generated using qualitative simulation. Recall that function is an abstraction of behavior. Function of a component is realized through its behavior. Therefore, the behavior based technique of constraint suspension [Davis, 1984] is used to observe the effect of malfunction/suspension of the behavior of a component on the behavior of the embedding device. This effect is expressed in terms of missing features, and the suspended constraints are abstracted to obtain the responsible class. Together, these form a feature-class association. As explained later, these associations generalize to other devices in the domain.

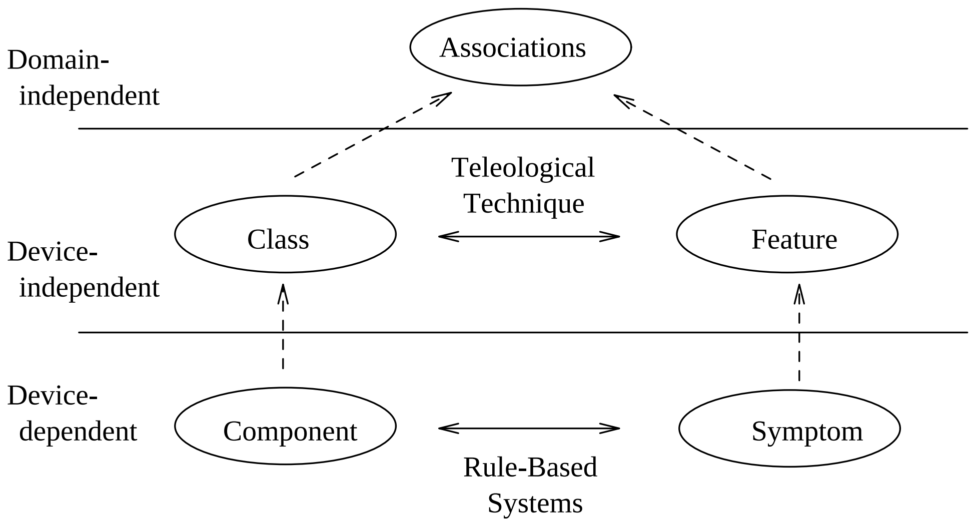


Figure 5.1: Teleological Reasoning Versus Rule Based Systems

The above algorithm resembles Failure Modes Effects Analysis (FMEA), which involves assessing the effects of all possible failure modes on a device/system. Function has been applied before to facilitate/automate FMEA [Wirth and O'Rorke, 1993, Hunt *et al.*, 1995]. Note that feature-class associations are a succinct representation of the results of FMEA.

Rather than exhaustively list every failure mode and its effects as in FMEA, function enables us to enumerate 'classes' of failures. For instance, in an AND gate with inputs p and q, FMEA would consider six cases in all: stuck open, stuck-at-1 and stuck-at-0 failure modes for each input. However, note that each input is a control input with respect to the other. The behavior of an AND gate can be re-written as $y = dc$, where data $d = p(q)$ and control $c = q(p)$ respectively. Now, when control c is stuck-at-1, $y = d$. When it is stuck-at-0, $y = 0$. When it is stuck open, the result depends on the technology used to implement the gate. In this case, function enabled us to list failures in half the space required by FMEA.

Analysis of Teleological Technique

The rules used in rule based systems associate components with symptoms. Hence, they are device and symptom-dependent. The feature-class associations used in teleological technique associate features with classes. Therefore, teleological technique may be considered to be an abstraction of rule based reasoning (See Figure 5.1).

Feature-class associations need not be exhaustively specified for all possible symptoms as in rule based systems. Since they are specified in terms of features which are higher level abstractions of symptoms, they cover a wider range of faults, and are not brittle. Being fewer in number than rules, they require less space.

Unlike symptoms, features are independent of the device. Therefore, feature-class associations may be applied for diagnosis of all the devices in a domain which have the same output features. Hence, feature-class associations are device-independent. As a result, time and space are saved while acquiring and storing them: they need not be acquired or stored individually for each device.

In addition to their device-independence, it is our hypothesis that the same feature-class associations can be applied to isomorphic function topologies in different domains. For instance, a producer-control-consumer structure, whether in a circuit or an automobile, obeys the same set of teleological rules. To the extent that this hypothesis is true, feature-class associations are also domain-independent.

Critical to the success of teleological technique of candidate discrimination are availability of:

- Appropriate feature-class associations for the device and domain. These are obtained by constraint suspension as described before.

- A *comprehensive* vocabulary of features to express outputs and symptoms. This may be obtained from literature on qualitative physics [deKleer and Brown, 1984, Forbus, 1984, Kuipers, 1986]. The features provided in this dissertation are those that are most widely applicable.

Teleological technique is better suited for continuous outputs or parametric observations of outputs than discrete outputs (e.g., better for observations of voltage than logic levels in digital circuits) because they have more features.

Milne is one of the early proponents of using teleological associations [Milne, 1985]. He proposes that components in a device "responsible" for different parts of the output be identified in the model, and suspected first during diagnosis. He uses associations between parts of analog output waveforms and components in an analog device. These associations are specific to analog devices and are device-dependent. Brajnik et al [Brajnik *et al.*, 1991] have also attempted to use teleology for diagnosis in the form of teleological models. However, the associations they propose between teleological and function models are also device-specific. Our contribution here has been to propose a framework for device and symptom-independent expression of associations.

The class model of a device is hierarchical. Teleological reasoning is applied repeatedly to various hierarchical levels of a class model. It uses constraint suspension [Davis, 1984] to derive symptoms at intermediate nodes at a given level, as well as to transfer diagnostic control from one level to the next lower level. Hence, it is scalable.

Extensions of Teleological Technique

So far, it has been assumed that a symptom translates to a single missing feature. If a symptom translates to a conjunction of missing features:

$$\neg f_1 \wedge \neg f_2 \wedge \ldots \wedge \neg f_n$$

the set of classes associated with all these features is used. Further, this set is sorted in descending order of the number of missing features among $f_1, \ldots, f_n$ associated with each class.

So far, particular classes have been associated with features. A better technique would be to associate particular orders of classes with features. Then, if one class fails to yield the faulty component, teleological reasoning can continue with the next one in the list associated with the missing feature.

In the preceding discussion, features were chosen to be canonical terms such that symptoms can be described in terms of (as a conjunction of) them. Note that functions may themselves be expressed as conjuncts of predicative terms. For instance, address class may be described as a conjunction of "preserving magnitude of signal" and "redirecting signal". It is advantageous to devise a set of terms which can be used to express symptoms as well as functions. When features, functions and symptoms all share the same vocabulary of terms, feature-class associations need not be compiled and stored any more. Instead, during diagnosis, symptoms can be translated to features, and these features can in turn be translated to classes.

5.1.2 Default Order Technique

In default order technique, components are ordered according to some reasonable order established among their classes. This order of classes should be device-independent, and being a default strategy of discrimination, should yield a reasonably good order of candidates in the absence of any symptom-, device- and domain-specific knowledge.

Towards establishing such an order among classes, two heuristic rules of reliability are proposed. Domain experts may order classes based on these rules, according to domain-specific information. However, in the absence of such expertise, a *default* order of classes of components and signal-lines is also proposed based on the rules, which is domain independent. The term "specialization" used in the following heuristic rules is understood as defined in Section 3.1.

Default Order of Components

Heuristic Rule 2 *The greater the specialization of a component in a signal-line, the greater its complexity, and hence, the greater the chances that it will break down.*

Therefore, the default order of classes of components is by decreasing order of their specialization. Relative specialization of the defined classes is characterized as follows:

- *Data* makes flow possible. Flow is conveyed unaltered.

- *Store* delays flow. Flow is conveyed unaltered, but some time delay is introduced.

- *Address* redirects flow. Flow is conveyed unaltered, but a destination is selected for it.

Therefore, address and store are specialized with respect to data. The following classes are more specialized than data, store and address since they alter flow:

- *Control* alters flow.

- *Consumer* consumes, and thereby alters flow. Flow is altered to be non-existent in its element.

- *Producer* generates flow. Flow is "altered into existence" in its element.

Therefore, producer and consumer are specialized with respect to control.

Hence, the default order of classes of components is a partial order as illustrated by the lattice structure in Figure 5.2. The order is producer and consumer, control, address and store, and data.

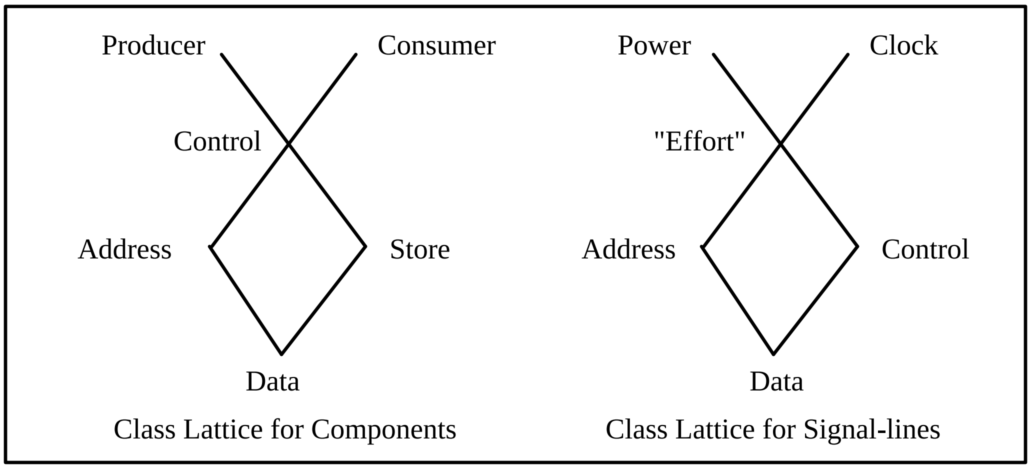


Figure 5.2: Default Order of Classes

Default Order of Signal-lines

Heuristic Rule 3 *The more specialized the class of a signal-line, the fewer the components in it. The more the components in a signal-line, the greater the chances of a fault lying in it.*

Therefore, the default order of classes of signal-lines is by increasing order of their specialization.

Effort enables flow. Hence, signal-lines conveying effort, i.e., active classes are more specialized than those conveying flow, i.e., passive classes.

Among flow signal-lines, i.e., passive classes:

- Data is the flow input.
- Address is the input which redirects a flow unaltered. It selects among different permissible destinations for the flow.
- Control is the input which alters a flow. It selects among different permissible transformations of the flow.

  Therefore, address and control are specialized with respect to data.

Among effort signal-lines, i.e., active classes:

- Power is the spatial effort that makes flow possible.
- Clock is the temporal effort that times a flow.

  Therefore, power and clock are specialized with respect to effort.

The default order of classes of signal-lines is again a partial order as illustrated by the lattice structure in Figure 5.2. The order is data, control and address, and power and clock.

Default Order for Suspect Ordering

Given a default order of classes of components and signal-lines for a domain, candidates are ordered as follows:

- Candidates are ordered according to the default order of their classes.
- Candidates in each class are in turn ordered according to other criteria such as the order in which they were generated or their fault probability.
- Candidates which belong to more than one class are ranked according to the earliest of their classes in default order.

5.1.3 Comparison of the Techniques

Candidate ordering based on default order is not sensitive to the symptom observed in the device. Diagnosis based on this technique translates to a fixed traversal of the diagnostic search space. Therefore, a device's components and signal-lines can be compiled into a search list according to the default order, and stored along with its device model.

Teleological technique is sensitive to the symptom, i.e., discrimination among suspects is performed based on the symptom at the outputs of the device. Hence, it is preferable to default order technique. If teleological technique suspects more than one class for a symptom, default order can be used to resolve the order among those classes. Default order technique can also be used to order components belonging to the unselected classes in teleological technique.

Heuristic rules 2 and 3 are not specific to any device or domain. Hence, the default orders of components and signal-lines are applicable to any device or domain. Since the order is based on principles of reliability, it is our hypothesis that it closely resembles the order suggested by fault probabilities of components [deKleer and Williams, 1987]. An advantage of using default order technique instead of fault probabilities is that function is readily available from design whereas fault probabilities are hard to obtain.

Teleological technique is device-independent by virtue of device-independence of the feature-class associations used by it. Feature-class associations and default orders of devices express knowledge about faulty behaviors of devices, which is traditionally used in rule based systems. In Classes, they are proposed to be used in addition to correct behaviors of devices, which are traditionally used in model based reasoning. Device-independence implies that they can be acquired and stored efficiently, unlike rules used in rule based reasoning which have to be acquired separately for each device.

The proposed discrimination techniques may be used for either candidate generation or only candidate ordering, as follows:

1. *Function-based Diagnosis:* Function is used to generate and order candidates as described in Section 5.2. Behavior-based constraint suspension is used to validate these candidates. Since function based diagnosis is heuristic, backtracking is incorporated into it, so that in the worst case, it degenerates to unfocused behavior-based diagnosis. However, in average case, it can be expected to focus diagnosis because functionally responsible components are far fewer than those that behaviorally contribute to a symptom.

2. *Function-based Ordering:* Dependency tracing or truth maintenance is used to generate candidates and constraint suspension is used to validate them. Validated candidates are now ordered based on function, using either of the discrimination techniques described before.

5.2 Diagnosis Algorithm

In this section, a model based algorithm is developed to diagnose a device based on its function. It uses the techniques described in Section 5.1 for candidate discrimination and steps through the class model of a device built using the algorithms in Chapter 4.

The algorithm assumes the existence of a single non-intermittent fault in the device. It also assumes that the device being diagnosed has only one element in it. It may be extended to diagnose devices with multiple elements by incorporating element-level discrimination techniques.

Algorithm 5 *Given the class model of a device D, and a symptom S at its output O, the algorithm to diagnose the device is as follows:*

1. All the signal-lines that terminate in output O are enumerated. Let the corresponding list be $SL = (S_1, \ldots, S_n)$.

2. The signal-lines in SL are ordered by either of the discrimination techniques. Let the ordered candidate-list of signal-lines be $SR = (S_1, \ldots, S_m)$. SR can be obtained by either of the following two approaches:

   - *Eager Approach*, where only the signal-lines belonging to classes suspected by the discrimination technique are retained in SR $(m \leq n)$.
   - *Safe Approach*, where all the signal-lines in SL are retained in SR, although reordered $(m = n)$.

3. The following steps are applied to the first signal-line $S_1 \in SR$. If it is found that the fault does not lie in S_1, the algorithm backtracks to consider $S_2 \in SR$ and so on. If necessary, the algorithm backtracks until all the signal-lines in SR are considered.

   (a) Let the unit of the signal-line under consideration be $CL = (C_1, \ldots, C_p)$.

   (b) The components in CL are ordered by either of the discrimination techniques. Let the ordered candidate-list of components be $CR = (C_1, \ldots, C_q)$. CR can be obtained by either of the following two approaches:

      - *Eager Approach*, where only the components belonging to classes suspected by the discrimination technique are retained in CR $(p \leq q)$.
      - *Safe Approach*, where all the components in CL are retained in CR, although reordered $(p = q)$.

      Candidates in CR may now be validated by function simulation [Fink and Lusth, 1987, Sticklen and Chandrasekaran, 1989, Brajnik *et al.*, 1990] or behavior based constraint suspension [Davis, 1984], to eliminate those candidates which cannot explain the symptom.

   (c) The following steps are carried out on the first component $C_1 \in CR$. If C_1 is found to be fault-free, the algorithm backtracks to consider component $C_2 \in CR$ and so on. If necessary, the algorithm backtracks until all the components in CR are considered.

      i. *Peripheral Diagnosis:* Other signal-lines connected to the inputs of C_1 are checked. All the signal-lines at the inputs of C_1 are listed except

S_1. Let the list of signal-lines be AL. The algorithm is recursively applied from Step 2 onwards to all the signal-lines in AL that have not already been considered. Note that, whereas AL is the set of signal-lines connected to a device at its level of structural hierarchy, SL is the set of signal-lines *within* the device at its next lower level of structural hierarchy.

ii. *Component Diagnosis:* If the component C_1 is not atomic or it is not at the level of desired granularity of diagnosis, it is checked for fault recursively at its next lower level of structural hierarchy. First, the symptom S' at output O' of the component C_1 which can explain the symptom S at the output O of the device D is computed by constraint suspension [Davis, 1984]. Next, the diagnosis algorithm is recursively called by substituting S by S', O by O', and D by C_1.
If C_1 is atomic, i.e., it cannot be decomposed into subcomponents, or if it is at the desired granularity of diagnosis, measurements are prompted for across it, in order to verify if it is faulty.

Peripheral and component diagnoses can be performed in either order:

- *Lazy Order:* Perform peripheral diagnosis before component diagnosis, i.e., check if any signal-lines connected to the component are faulty before checking if the component is itself faulty. This order is based on the assumption that the cost of constraint suspension is less than the cost of probing for measurements. It is ideal for address and control classes of components since they have several inputs which could be faulty.

- *Greedy Order:* Perform component diagnosis before peripheral diagnosis, i.e., check if the component is itself faulty, before checking if any signal-lines connected to it are faulty. This order assumes that probing for measurements is cheaper than constraint suspension. It is ideal for producer, consumer, data and store classes of components since they tend to have few inputs which could be faulty.

5.2.1 Single Fault Assumption

Assuming a uniform probability p of failure of components, the probability of m independent faults occurring in a device which has n components is:

$$C_m^n p^m (1-p)^{n-m}$$

Usually, $p \ll (1-p)$. Therefore, single faults are more likely to occur than multiple independent faults.

Even when multiple faults occur in a device, single fault diagnosis algorithms may be repeatedly applied until all the faults in the device are located. It is our belief that this strategy can effectively diagnose most cases of multiple faults. (An exception is the relatively rare case where multiple faults together appear as an unrelated single fault.)

Finally, theoretical results show that diagnosing multiple faults is a problem of exponential complexity [Reggia *et al.*, 1985, Reiter, 1987]. Because of the above considerations, our diagnosis algorithm assumes the existence of a single fault.

Note that multiple faults have been considered in model based diagnosis in systems such as GDE [deKleer and Williams, 1987] and XDE [Hamscher, 1988]. These systems consider subsets of components as suspects and order them based on fault probability. However, they assume that the multiple faults are independent. In reality, this is rarely the case. E.g., a fuse blows, resulting in a burned out bulb. Adapting these systems to treat dependent multiple faults is an onerous task since they then need to consider a combinatorial explosion of combined probabilities.

5.2.2 Highlights of the Diagnosis Algorithm

The diagnosis algorithm exploits the dual nature of Classes. In Steps 1 and 3(c)i, given a device, the algorithm suspects a signal-line, whereas in Step 3a, given a signal-line, it suspects a device. This dual approach enables the algorithm to systematically step through the class model of a device. It ensures that, if necessary, all the components causally connected to a symptom are eventually considered during diagnosis. The order in which components/signal-lines are considered is based on causality and function.

The diagnosis algorithm may be characterized as a depth-first search of the component tree (See Figure 5.3), coupled with function based heuristic criteria to prune (order and eliminate) choices at each level of the tree. This perspective is described in greater detail in Section 5.3. In literature, pruning of choices has been proposed based on structure [Chen and Srihari, 1989] and fault probabilities of components [deKleer and Williams, 1987]. Function-based pruning of the search tree is our novel contribution.

Abductive problem solving has been proposed both as a unitary step from symptoms to diagnosis [Reggia *et al.*, 1985] and as a two step process [Josephson and Josephson, 1993]. Our diagnosis algorithm is a multi-step process wherein, abduction is applied repeatedly (Steps 2 and 3b) until the faulty component is found. Each application of abduction uses the results of the previous application as its starting point. The utility of class representation is in fragmenting a device into modular subdevices, appropriate and convenient for application of abduction based on function.

The diagnosis algorithm is device and domain-independent. It is model-based and is part of the computational model of Classes summarized in Section 3.3. The discrimination techniques it uses are also device-independent, and hence, space-wise efficient as discussed in Section 5.1.3.

The diagnosis algorithm is scalable to large and complex devices so long as they are represented hierarchically: during *component diagnosis* (Step 3(c)ii), it recursively considers lower levels of structural hierarchy of the component. Since the class model of a device built using the algorithms in Chapter 4 is itself hierarchical, Classes is a scalable computational model for function based diagnosis.

During diagnosis, only complete signal-lines (as defined in Chapter 4) are considered. From Lemma 1, every signal-line is a data signal-line. Therefore, the task of diagnosing a device is *elaborated* into diagnosing complete data signal-lines in it.

Lemma 3 *The data signal-line is always suspected first with respect to a component.*

Proof: The algorithm begins by considering only data signal-lines connected to the symptom. From then on, a component is suspected only when a symptom is observed or inferred in the data signal-line to which it belongs.

Lemma 4 *If a consumer component exists in a signal-line, it is always suspected first.*

Proof: A signal-line is suspected only when a symptom is observed or inferred at the device output or consumer component at which it terminates.

Lemma 5 *The safe approach of diagnosis is complete.*

Proof: Safe approach does not eliminate components/signal-lines at any stage. It only reorders them. If a suspect is found to be not faulty, the diagnosis algorithm backtracks to consider the next candidate in the suspect list. Hence, in the safe approach, in the worst case, all the components/signal-lines in a device are eventually examined.

5.2.3 Diagnosing the Door Bell

In this section, three scenarios are presented for the diagnosis of the door bell described in Section 3.2.3. The first one illustrates the working of the diagnosis algorithm and uses default order technique for discrimination. The other two scenarios demonstrate the teleological technique.

Scenario 1: Suppose the bell does not ring. The mechanical signal-line is suspected, since it is the only signal-line connected to the symptom at the gong. The following ordered candidate-list is generated: electromagnet (producer), contact and gong (consumer), spring (control), armature and hammer (data). The first suspect is the electromagnet, a producer. Since greedy order is followed for producers, testing of the electromagnet is initiated. If the electromagnet is found to be not faulty, by peripheral diagnosis, the only other signal-line connected to it is suspected, i.e., the electrical signal-line in which it is a consumer. In the electrical signal-line, the ordered candidate-list is: battery (producer), switch (control), contact, spring and wires (data). The first candidate is battery, and it is tested as per the greedy order. If it is not faulty, the algorithm backtracks to consider, in order: switch, contact, spring, wires, armature and hammer. Note that the diagnosis algorithm orders the candidate-list when generating it. The order is based on function, and enables examination of the candidates which are more likely to have caused the symptom (battery) before those that are less likely (hammer and wires).

Scenario 2: Suppose the door bell "clanks" but does not ring repeatedly. This translates to valid but incorrect transition in mechanical flow. Based on feature-class associations of teleological technique, mechanical control is suspected first. A test of the spring is initiated.

Scenario 3: Suppose the door bell rings ceaselessly. The symptom translates to incorrect transition in the mechanical signal. By feature-class associations, mechanical control and producer are suspected first. Suppose the control component, i.e., spring is found to be fault-free. The producer, i.e., electromagnet is now diagnosed by component and peripheral diagnoses. During peripheral diagnosis, the electrical signal-line is considered. Incorrect transition in mechanical flow is translated to incorrect transition in electrical flow. Therefore, the control component in it, i.e., the switch is suspected first. Since control class is diagnosed by lazy order, mechanical signal-line from the user's hand to the switch is considered next.

5.3 Evaluation of the Discrimination Techniques

In this section, our two discrimination techniques are compared with two other techniques proposed in literature: Chen and Srihari's technique for candidate ordering based on structure [Chen and Srihari, 1989] and deKleer and Williams' minimum entropy technique for probe selection based on fault probability [deKleer and Williams, 1987, deKleer, 1990].

The ordering technique based on structure [Chen and Srihari, 1989] is dependent on device topology. It works well for devices with large number of fan-out connections, but fails for devices with linear topology. It yields different results depending on how the components in a device are aggregated [1]. In comparison, topology does not affect candidate ordering based on classes.

Minimum entropy technique [deKleer and Williams, 1987, deKleer, 1990] is based on fault probabilities. Two typical cases are considered for comparison of our discrimination techniques with minimum entropy technique: a single signal-line and a device composed of several signal-lines. A single non-intermittent fault is assumed to be present in the device.

Case 1- Single Signal-line: Discrimination among components in a signal-line is carried out as follows:

- Default Order technique always suspects components in the default order: consumer, producer, control, address, store and data. Therefore, if the fault lies in a consumer component, only one measurement is required to locate it. On the other hand, if the fault lies in one of the data components, it is located after several intermediate measurements.

[1] For instance, in the printer buffer display described in Section 6.3, it yields different results depending on whether each LED segment is considered a separate device (order: *RP1*, *CPU*, *Buffer1*, *Decoder*, Segment(s) at fault ...) or all eight parts (seven segments and a decimal point) of the LED are aggregated into a single device (order: *LED1*, *RP1*, *RP2*, *CPU*, ...).

- Teleological technique suspects components which belong to a particular class(es), based on the symptom and applicable feature-class associations. For instance, if the symptom is incorrect transition of the output, control and producer components are suspected first, based on the rules in Section 5.1.1. If the control/producer component(s) is indeed faulty, the fault is located with few measurements. Otherwise, other components are checked using the default order technique.

- Minimum entropy technique reduces to half-split search when all the components are assumed to have the same fault probability. If some component is assumed to have a greater fault probability than the rest, minimum entropy suggests measurements at the ports of that component before prompting for measurements elsewhere. If the assignment of fault probabilities to components reflects the default order of classes of those components, minimum entropy technique translates to default order technique. This scenario is quite likely since, default order is dependent on relative specialization, complexity of construction and hence, theoretical fault probabilities of components.

Case 2- A Composite of Signal-lines: The diagnostic search space of a complex device consisting of several inter-connected signal-lines is shown in Figure 5.3. The root of the tree is the symptom. Nodes at the first (and every subsequent odd) level in the tree are components in a signal-line connected to the symptom. Nodes at the second (and every subsequent even) level are signal-lines connected to a component. Leaf nodes are classes of atomic components in the device.

Note that, this tree is not the same as the structural hierarchy of a device. In fact, it corresponds to only one level in the hierarchy. The links in the tree represent structural connectivity and should not be mistaken for component-device relationship.

In the tree, all the classes of components/signal-lines are shown at each level, although, in a device, some of the classes may not exist. Further, in a device, more than one component/signal-line may belong to the same class. At the even levels, data signal-line is not shown because components at the preceding odd level constitute the data signal-line. At the odd levels, consumer is the left-most sibling since symptoms are usually observed in consumer components. In order to facilitate comparison of our techniques, siblings at each level of the tree are ordered by default order.

Diagnosis using default order technique is a depth first search of the above tree. Therefore, it is systematic. However, its efficiency depends on where the solution lies

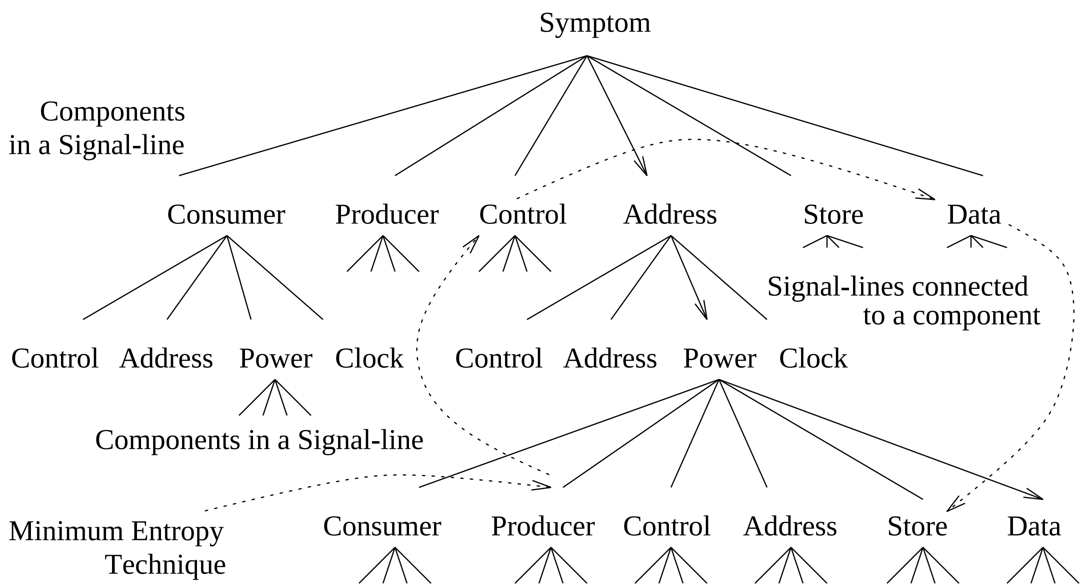


Figure 5.3: Search Space for Diagnosis

in the search space. Diagnosis using teleological technique is a depth-first traversal of the tree, although it is not carried out in the left-to-right fashion of depth-first search. For instance, arrows in the figure from the symptom to an address component, power signal-line and a data component depict a typical traversal of the tree by diagnosis using teleological technique. The diagnosis algorithm pursues its choice till it reaches a leaf node. If the component corresponding to the leaf node is faulty, the diagnosis is complete. The number of measurements requested is proportional to the topological distance of the fault from the symptom, measured in terms of signal-lines. If the leaf node is not faulty, the diagnosis algorithm backtracks to its parent node and performs a default order, i.e., depth first search of the rest of the tree rooted at the parent node. This continues until, in the worst case, diagnosis backtracks to the root node.

Minimum entropy technique considers only components (and not signal-lines). Further, it considers all intermediate points/components for each decision, i.e., it always considers all the nodes at all the odd levels of the tree. A typical search path followed by it is shown by dotted arrows in the figure. As is evident from the figure, it is hard to explain the sequence of components considered by it, based on causality. E.g., why is a control component considered after the producer in the power signal-line connected to an address component, when the two are not causally related?

On the other hand, our diagnosis algorithm systematically traverses the search tree. Every component/signal-line it considers is causally related to the signal-line/component that it last considered. Therefore, the order in which it examines components can be explained based on causality. Such explanations are necessary for humans to understand the diagnosis carried out by the system. The cognitive coherence of the output of a system influences how well the system is accepted by humans, especially in critical applications where human computer interaction is vital and frequent. In diagnosis, understandability of the output of a system leads to reassurance in the competence of the system.

Minimum entropy technique is based on information-theoretic principles. The sequence of components considered by it can be explained on the basis of finding the next measurement which yields the most information about the device being diagnosed, but not on the basis of causality. The resulting explanation is unintuitive.

In Figure 5.3, suppose the data component identified by diagnosis using teleological technique is indeed faulty. Suppose again that fault probability assigned to the component is the highest among all components. Minimum entropy technique would be able to identify the fault directly, i.e., without any intermediate measurements required by our diagnosis algorithm. Other than being unintuitive to the user, this result also highlights the dependency of minimum entropy technique on empirically obtained fault probabilities rather than on causal topology of the device. Often, these probabilities are unavailable, unknown or hard to obtain. In comparison, function exploited by our discrimination techniques is readily available from the design of a device.

Easy availability is a winning advantage of using function instead of fault probability. Consider the effectiveness of minimum entropy and teleological techniques. Both these techniques are knowledge-based. Both are only as good as the knowledge (fault probabilities, feature-class associations) provided to them. Therefore, easier availability of the necessary knowledge makes teleological technique the better choice between the two.

Since fault probabilities are hard to obtain, deKleer has proposed a more practicable version of minimum entropy technique in which, exact probabilities need not be known and all components are assumed to fail with equal probability [deKleer, 1990]. He generalizes this to have *partitions* of all the components with the same fault probability, such that components in one partition are more likely to fail than those in another partition. *Classes, by definition, provides such partitions based on component function.* (Since single fault is assumed in Classes, $q = 1$ in [deKleer, 1990].) The implications of this are two-fold:

1. Note that default order of classes was derived based on principles of reliability. Therefore, we hypothesize that it is similar to the order of the partitions arranged by their fault probabilities.

2. Since function is readily available whereas fault probabilities are not, it can be used as a substitute for them. It can also be used to generate partitions of devices, which can in turn be used in the revised version of minimum entropy technique. This is especially useful for devices whose fault probabilities are unknown (novel devices), unavailable (classified data), or required only at a coarse level of resolution.

Function is obtained from design whereas fault probabilities are obtained from experience. In today's technology, where novelty to obsolescence is measured in months rather than years, the advantage of using function for diagnosis can hardly be exaggerated.

To summarize, Chen and Srihari's technique works well for topologies with large fan-outs. Minimum entropy technique is applicable to devices with standard components which have readily available fault data. Our discrimination techniques are applicable to standard as well as novel devices, and are not affected by device topology. They are as effective as using fault probabilities, i.e., can be expected to prompt for similar sequences of measurements because:

- Default order is based on principles of reliability and resembles theoretical fault probabilities.

- The effectiveness of teleological technique and minimum entropy technique are comparable in that both are only as good as the knowledge provided to them. This knowledge is easier to obtain for teleological technique.

Our diagnosis algorithm using these techniques has the advantage that it facilitates generation of explanations during diagnosis. Default order may even be used to partition components for the revised version of minimum entropy technique [deKleer, 1990].

Function based diagnosis is better at diagnosing modular devices than those where a single function is shared among several components and/or a single component handles several functions. This is encouraging: modular device designs are popular because they tend to facilitate diagnosis and repair by replacement.

5.4 Summary

In this chapter, function has been proposed to be used for candidate discrimination in model based diagnosis. Two function based candidate discrimination techniques were described, viz., teleological and default order techniques. A model based algorithm was developed, which uses these techniques to diagnose the class model of a device. Finally, our discrimination techniques were compared with traditional techniques and their advantages were analyzed.

In the next chapter, our implementation of the diagnosis algorithm is reported. It is used to diagnose a printer buffer. The scalability and domain-independence of Classes are demonstrated by building class models of a printer buffer and an automobile carburetor.

Chapter 6

Implementation and Examples

In this chapter, our implementation of Classes is described. Two devices have been chosen to demonstrate Classes: a printer buffer and an automobile carburetor.

Scalability of Classes is demonstrated by building the class model of a printer buffer in Section 6.2. Representation and reasoning in Classes are illustrated using the Display Unit of the printer buffer in Section 6.3. A sample run generated by our implemented system diagnosing the Display Unit is listed in Appendix A. Finally, domain-independence of Classes is illustrated by building the class model of an automobile carburetor in Section 6.4.

6.1 System Description

The diagnosis system is implemented in Common Lisp. It includes the diagnosis algorithm and default order technique of candidate discrimination. Components and devices are represented as s-expressions in Lisp.

Structure and function of devices are modeled using slot-filler representation [Minsky, 1981], where all relevant information is attached to a device model through property lists of Lisp. Generic properties of a device are stored separate from its specific properties. For instance, the library file for AND gates stores the behavior of AND gates, number of ports, etc. The specific file for an AND gate stores such details as the instance of the chip in which it is housed.

Simulation and inference behavior of devices [Davis, 1984] are represented procedurally using LISP functions rather than declaratively as in DART [Genesereth, 1984] and GDE [deKleer and Williams, 1987]. Provision is made to store three values for every port of every component: expected values obtained by simulation, validation values obtained by constraint suspension [Davis, 1984] and measured values obtained from the user. Not all the values may be present for every port.

Initially during diagnosis, measured values of inputs and outputs of the device are obtained from the user. Measured values of inputs are assigned to be their expected values also. Simulation of the device is carried out using expected values of inputs to calculate expected values for all intermediate and output ports of the device.

Measured values of outputs are assigned to be their validation values also. Constraint suspension is performed using these validation values. Proceeding from outputs to inputs, validation values are calculated for all intermediate and input ports of the device. These values specify how each component could explain the symptoms of the device. Components with conflicting validation values are marked for elimination from the candidate list. Finally, function based diagnosis is performed as described in Chapter 5.

Validation by constraint suspension is performed for all the components before function based diagnosis because:

- It is more efficient to perform constraint suspension for all the components in one pass through the device rather than on demand.
- Validation values provide "symptoms" at the outputs of intermediate components. These symptoms are used by teleological technique of candidate discrimination.

6.2 The Printer Buffer

The Heathkit printer buffer (See schematic in Figure 6.1) provides interface between a computer and a printer. It receives files from the computer, stores them if necessary, and prints them when the printer is free. It has both serial and parallel input and output ports and can be interfaced between two computers and two printers simultaneously (See Block Diagram in Figure 6.2.).

The Central Processing Unit (CPU) of the printer buffer is an 8-bit CMOS microprocessor (Z64180) running on a 6.144 MHz clock. Its system program is 4 Kbytes long and is stored on a Read Only Memory (ROM). The data bus is 8-bit wide while the address bus is 19-bit wide and allows up to 512 Kbytes of Random Access Memory (RAM) to be used. It comes with two options in RAM size: storage of 64 Kbytes or

Figure 6.1: Schematic of the Heathkit Printer Buffer

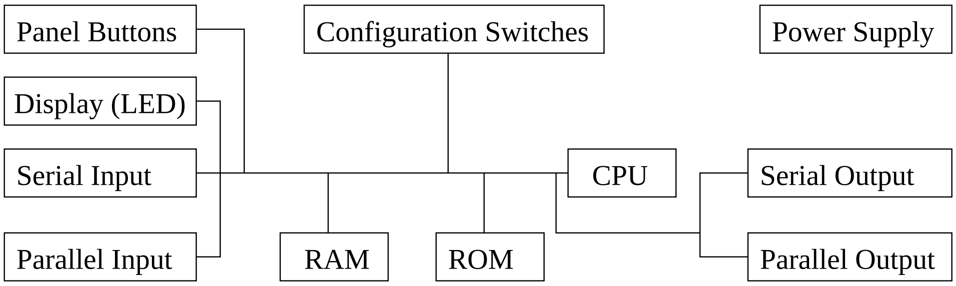


Figure 6.2: Block Diagram of the Printer Buffer

512 Kbytes. The printer buffer can accept data[1] at up to 38.4 Kbaud (serial) or 100 KCPS (parallel) speed.

The printer buffer displays the amount of free memory on front panel Light Emitting Diodes (LED). It runs a self-diagnostic program to check its memory when powered on. It provides panel buttons to specify functions such as repeat printing, priority printing, dumping the buffer, etc. It also provides internal configuration switches to set baud rates, parity, handshake modes, etc.

In the next several subsections, various units[2] of the printer buffer are modeled using Classes in order to demonstrate the scalability of Classes. Note that in the following models, capacitors and pull-up resistors have been ignored for the sake of simplicity. The ubiquitous power signal-lines are not shown in most class models in order to improve clarity of the figures. In the figures, only classes of components with respect to their inputs have been labeled. Classes of signal-lines with respect to their components have been omitted, but are explained in the accompanying text.

6.2.1 The ROM Unit

The Read Only Memory (ROM) Unit stores the firmware necessary to operate the printer buffer. It consists of ROM chip, ROM Decoder and CPU components. Its class model is shown in Figure 6.3.

The ROM chip resides in the first 4 Kbytes of address space of the machine. The CPU selects the ROM chip through the ROM Decoder. When the ROM Decoder is

[1]This refers to bit-wise information and not the data class. Although "data" is used in both these senses in this chapter, its correct meaning should be clear from the context.

[2]In order to simplify the presentation of the class model of the printer buffer, it has been broken down into several subdevices, which are referred to as "units" in this chapter. These are not the same as the units defined in Section 3.2. These do not correspond to single signal-lines, and may each consist of several signal-lines.

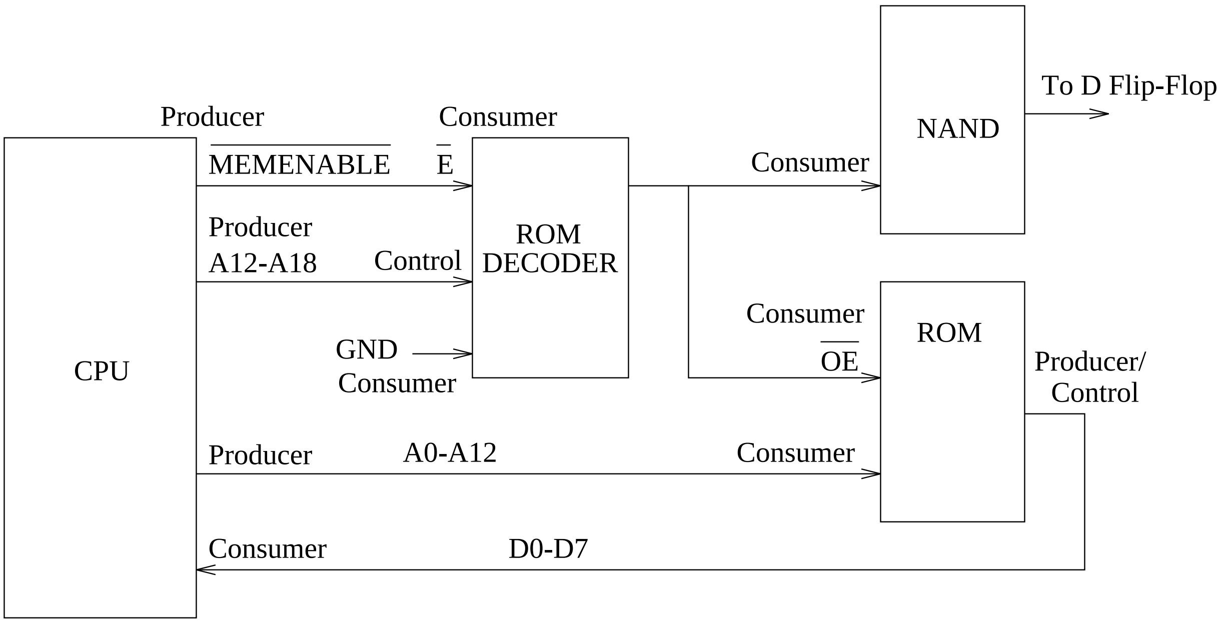


Figure 6.3: Class Model of the ROM Unit

enabled by a low $\overline{MEMENABLE}$ signal from the CPU, and address lines A12-A18 are all zeros, it outputs 0. This low output is used to enable the ROM chip through its Output Enable ($\overline{OE}$) input. The CPU selects specific locations in the ROM chip through the address lines A0-A12.

Class model of the ROM Unit illustrates the subtleties of Classes well (See Figure 6.3). Several signal-lines interact in the class model:

- CPU is the producer of signal A0-A12 which is consumed by the ROM chip. This is an address signal-line with respect to the ROM chip.

- ROM chip is the producer/control of signal D0-D7 which is consumed by the CPU. This is a data signal-line with respect to the CPU.

- CPU is the producer of $\overline{MEMENABLE}$ output which is consumed by the ROM-Decoder. This is a power signal-line with respect to the ROM-Decoder because the ROM-Decoder will not function without this input. (In addition, the ROM decoder has another power signal-line that conveys power to the chip, which is not shown in the figure.)

- The Ground signal is consumed by the Decoder. It is a control signal-line with respect to the Decoder. The Decoder compares its A12-A18 input with this signal and outputs 0 only when the two are the same.

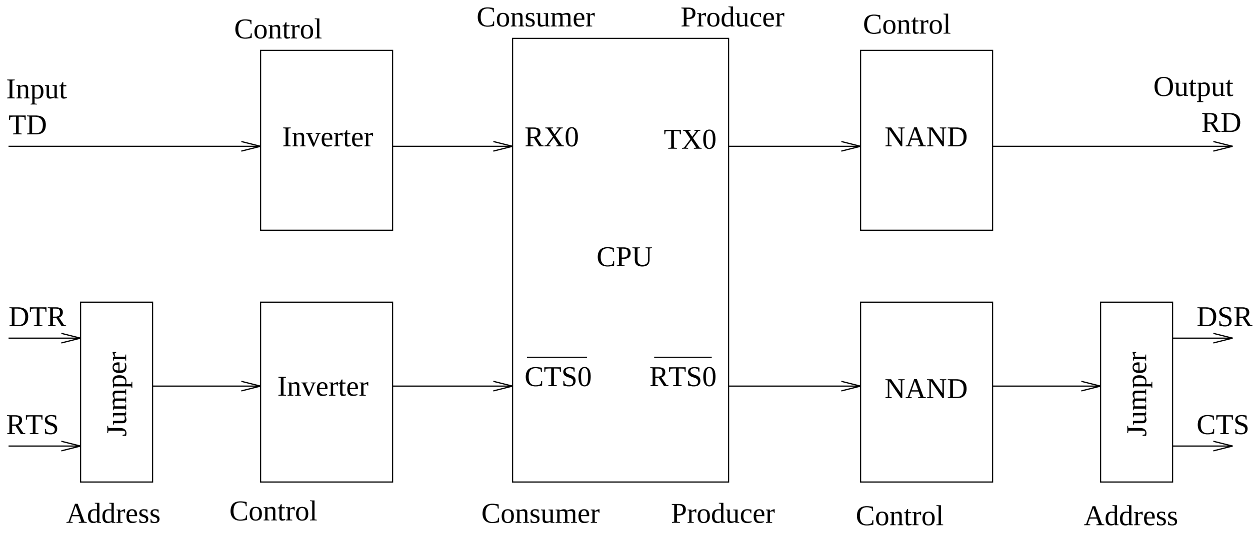


Figure 6.4: Class Model of the Serial Input Unit

- CPU is the producer of the signal A12-A18. ROM Decoder is its control component. This signal is consumed by the ROM chip at its $\overline{OE}$ input. The resulting signal-line *CPU-ROM Decoder-ROM* is a control signal-line with respect to the ROM chip, since it enables the ROM to output its contents. The output of the ROM Decoder is also consumed by a NAND gate, which is part of the RAM Refresh Unit and will be discussed later in this chapter.

6.2.2 The Serial Input Unit

The Serial Input Unit receives files from the computer in serial fashion. It consists of CPU, Inverters (RS232 Receivers), NAND gates (RS232 Transmitters) and Jumpers. Its class model is shown in Figure 6.4.

The Serial Input Unit uses TD input to receive data and RD output to transmit data[3] as a series of pulses. These are its communication channels.

Transfer of data between the computer and the printer buffer through the serial port is coordinated by *handshaking* protocols. Handshaking may be done through software using only TD and RD ports, or through hardware. The Serial Input Unit supports two types of hardware handshaking: either through DTR/DSR (Data Terminal Ready/Data Set Ready) ports, or through RTS/CTS (Request To Send/Clear To Send) ports. In the case of DTR/DSR handshaking, the printer buffer sets DSR

[3]The input unit has the capability to transmit data to the computer.

to low to indicate that it is ready to receive data. When the computer senses this low, it sets DTR to high to indicate that data is ready to be sent. It sends the data through TD input after this handshaking. After receiving the data, the printer buffer sets its DSR signal to high in order to prevent the computer from sending more data till it is ready. Similarly, in the case of RTS/CTS handshaking, the computer sends a signal on RTS requesting an okay to send data. When the printer buffer senses this request, it signals that it is ready to receive data through CTS. Data transfer through TD input follows.

Class model of the Serial Input Unit consists of several signal-lines (See Figure 6.4):

- TD input is controlled by the Inverter and consumed by the CPU. This is a data signal-line with respect to the CPU.
- The CPU produces TX0 signal which is controlled by the NAND gate. It is connected to RD output.
- The Jumper is a manually switched address component with respect to input handshake signals DTR and RTS. The Inverter controls the signal output by the Jumper and the CPU consumes it. The resulting signal-line *Jumper-Inverter-CPU* is a control signal-line with respect to the CPU.
- The CPU is the producer of $\overline{RTS0}$ signal which is controlled by the NAND gate. The Jumper is an address component with respect to this signal and connects it to outputs DSR and CTS.

6.2.3 The Serial Output Unit

The Serial Output Unit sends files to the printer in serial fashion. It is connected to the second of the two serial ports supported by the CPU. It consists of CPU, Inverters (RS232 Receivers), NAND gate (RS232 Transmitter) and a Jumper. Class model of the Serial Output Unit is shown in Figure 6.5.

The printer buffer's Serial Output Unit receives data through RD input [4] and transmits data to the printer through TD output as a sequence of pulses. Its handshaking protocol is similar to that of the Serial Input Unit, except that it is one way:

[4]The output unit has the capability to receive data from the printer.

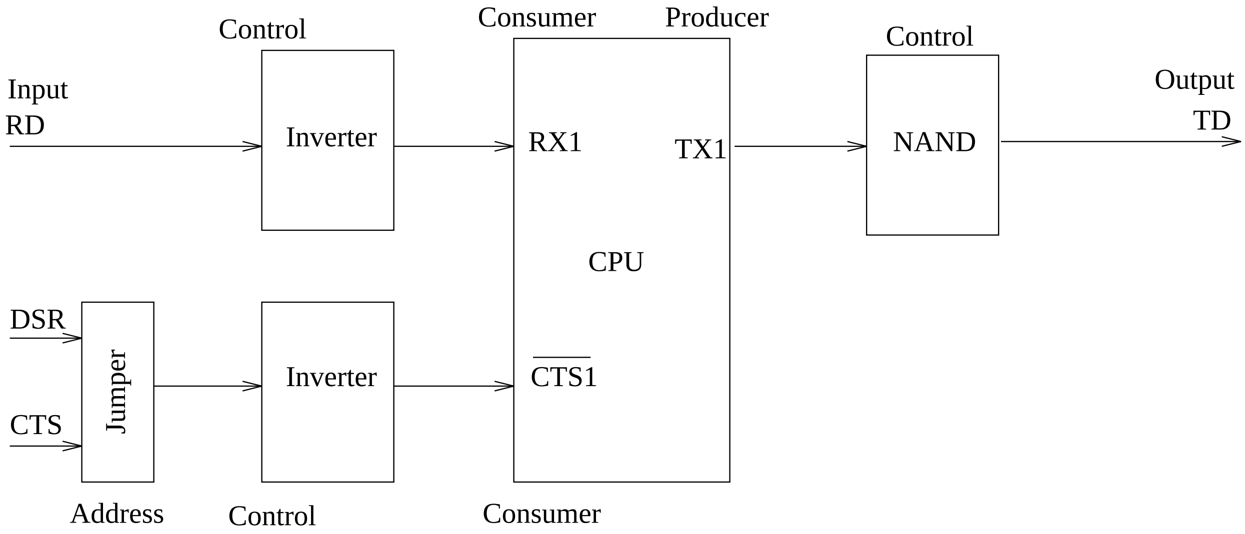


Figure 6.5: Class Model of the Serial Output Unit

it only receives handshaking from the printer. When the printer sends a low DSR or CTS signal, the unit sends data through the TD output.

Class model of the Serial Output Unit has the following signal-lines (See Figure 6.5):

- The RD input is controlled by the Inverter and consumed by the CPU. This is a data signal-line with respect to the CPU.
- The Jumper is a manually switched address component with respect to the input signals DSR/CTS. The Inverter controls the signal output by the Jumper and the CPU consumes it. The resulting signal-line *Jumper-Inverter-CPU* is a control signal-line with respect to the CPU.
- The CPU produces the TX1 signal which is controlled by the NAND gate. It is connected to the TD output.

6.2.4 The Configuration Switches Unit

The Configuration Switches Unit enables the user to set communication parameters. It consists of Tri-state Buffers, a 3-to-8 line Decoder/Demultiplexer, Switches and CPU. Class model of the Configuration Switches Unit is shown in Figure 6.6.

The user can manually configure the switches to set communication parameters such as baud rate, word length, parity status (on/off), parity (odd/even), number of

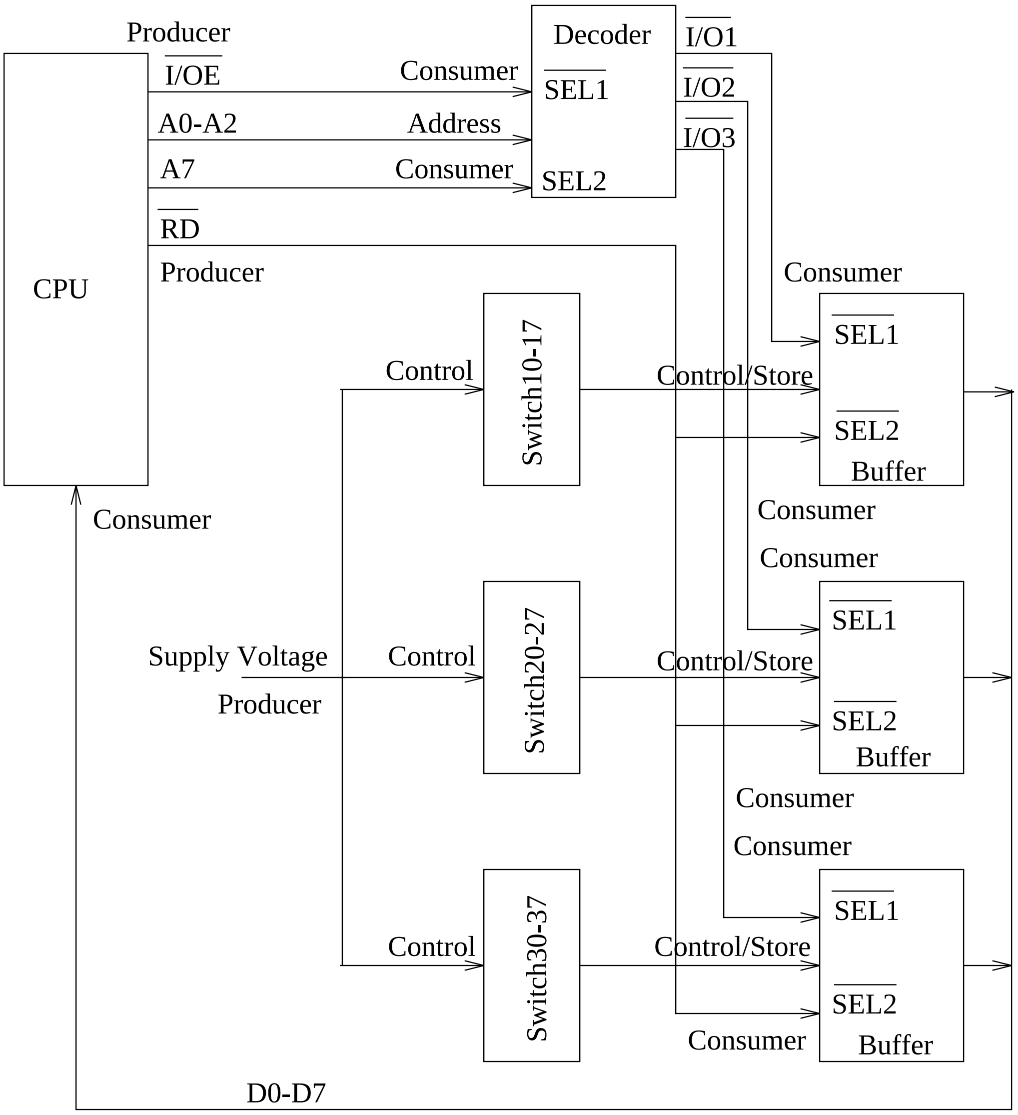


Figure 6.6: Class Model of the Configuration Switches Unit

stop bits, and handshaking and its polarity. An open switch outputs 1 and a closed switch outputs 0. The CPU reads the settings of these switches through three Tri-state Buffers as follows: It outputs address A0-A2 which is used by the Decoder to select a particular Buffer. When the selected Buffer also receives a low $\overline{RD}$ signal from the CPU, it loads the settings of the 8 switches connected to its input, i.e., one byte of data, on to the data bus D0-D7 for the CPU to read.

In Figure 6.6, class model of the Configuration Switches Unit is shown:

- The CPU is the producer of signal $\overline{I/OE}$ consumed by the Decoder. This is a power signal-line with respect to the Decoder.

- The CPU produces signal A7 which is consumed by the Decoder. This is also a power signal-line with respect to the Decoder.

- Signal A0-A2 is produced by the CPU. The Decoder is an address component with respect to this signal. The Buffers are consumers of this signal in the form of $\overline{I/O}$ input. The signal-line *CPU-Decoder-Buffer* is a control signal-line with respect to the Buffers.

- $\overline{RD}$ signal produced by the CPU is consumed by the Buffers. This is also a control signal-line with respect to the Buffers.

- Supply Voltage is the producer of the signal which is controlled by the manual Switches Switch10-17, Switch20-27 and Switch30-37. Buffers act as control/store components of the signal and the CPU consumes it through the data bus D0-D7. The resulting signal-line *Supply Voltage-Switch-Buffer-CPU*[5] is a data signal-line[6] with respect to the CPU.

6.2.5 The Parallel Input Unit

The Parallel Input Unit receives files from the computer in parallel fashion. It consists of CPU, Schmitt Trigger Inverter, Resistor Pack (RP), D Flip-Flops, OR gate,

[5]This actually refers to three different signal-lines passing through Switch10-17, Switch20-27 and Switch30-37. In this discussion, for simplicity, the device suffixes have been dropped to obtain one name that stands for all three signal-lines. This convention is used where necessary, through the rest of this chapter.

[6]Technically, this is a control signal-line with respect to the CPU because the CPU decides which software routines to use for communication, based on this signal. However, since software of the printer buffer is not being modeled as part of this effort, this subtlety is being ignored here.

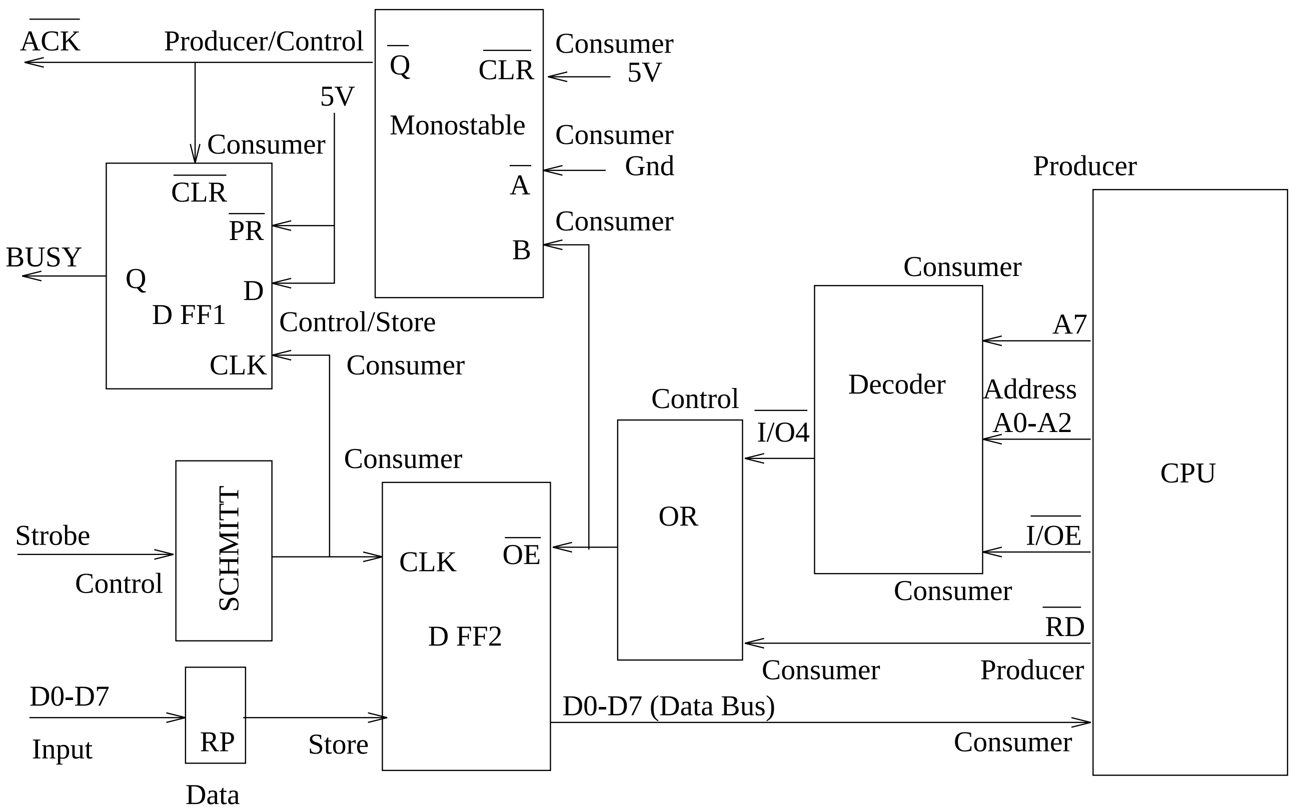


Figure 6.7: Class Model of the Parallel Input Unit

Monostable Multivibrator and Decoder. It receives data through inputs D0-D7 and uses Strobe, BUSY and $\overline{ACK}$ signals for handshaking. Class model of the Parallel Input Unit is shown in Figure 6.7.

When the computer wants to send data to the printer buffer in parallel fashion, it checks the BUSY and $\overline{ACK}$ signals issued by the Parallel Input Unit. If BUSY signal is high, the computer waits. Otherwise, if $\overline{ACK}$ signal is low, the computer sends one byte of data in parallel on inputs D0-D7 and pulses the Strobe line low. This Strobe line in its inverted form is used by D Flip-Flop2 (D FF2) to latch the data present on input D0-D7. The same inverted Strobe is used by D Flip-Flop1 (D FF1) to issue a high BUSY signal.

The CPU reads the data stored in D Flip-Flop2 by selecting it with a low $\overline{OE}$ input. The low $\overline{OE}$ is generated as a conjunction of a low $\overline{RD}$ signal issued by the CPU and a low $\overline{I/O4}$ signal generated by the Decoder as described before in the Configuration Switches Unit. The CPU transfers the data from D Flip-Flop2 over the data bus D0-D7 to its RAM Unit. After the transfer, it unselects D Flip-Flop2, and $\overline{OE}$ input of D Flip-Flop2 rises to 1. This rising signal triggers the Monostable Multivibrator into issuing a low pulse of fixed time duration. This is the low $\overline{ACK}$ pulse which signals that the Parallel Input Unit is ready to accept a new byte of data. This signal is used by D Flip-Flop1 to also reset BUSY signal to 0.

Class model of the Parallel Input Unit shown in Figure 6.7 consists of the following signal-lines:

- The Resistor Pack (RP) is a data component with respect to input signal D0-D7. This signal is stored by D Flip-Flop2 (D FF2) and consumed by the CPU. The resulting signal-line *RP-D Flip-Flop2-CPU* is a data signal-line with respect to the CPU.

- Strobe input is controlled by the Schmitt Trigger Inverter which eliminates noise. It is consumed by D Flip-Flop1 and D Flip-Flop2. It is a clock signal-line with respect to the D Flip-Flops.

- $\overline{RD}$ signal is produced by the CPU and consumed by the OR gate. It is a control signal-line with respect to the OR gate.

- Signal-lines necessary to generate $\overline{I/O4}$ signal have been described as part of the Configuration Switches Unit. The Decoder is an address component with respect to this signal. The OR gate controls the signal and D Flip-Flop2 consumes it. The resulting signal-line *CPU-Decoder-OR-D Flip-Flop2* is a control signal-line with respect to D Flip-Flop2. It enables D Flip-Flop2 to output the data it has latched, on to the data bus D0-D7 (but does not affect the latching itself).

  The output of the OR gate is also consumed by the Monostable Multivibrator. The resulting signal-line *CPU-Decoder-OR-Monostable* is a control signal-line with respect to the Monostable Multivibrator.

- $\overline{ACK}$ signal is produced/controlled by the Monostable Multivibrator and output by the Parallel Input Unit. The Multivibrator controls production of the signal based on its inputs B, $\overline{A}$ and $\overline{CLR}$. The $\overline{ACK}$ signal is also consumed by D Flip-Flop1 and is a control input with respect to it.

- $\overline{A}$ and $\overline{CLR}$ inputs are consumed by the Monostable Multivibrator. They are control signal-lines with respect to it. Note that, the resistor and capacitor which fix the duration of $\overline{ACK}$ pulse have been encapsulated [Hamscher, 1988] into the Monostable Multivibrator in this model.

- BUSY signal is produced as Supply Voltage (5V) connected to D input of D Flip-Flop1. It is stored/controlled by D Flip-Flop1 and is output by the Parallel Input Unit. D Flip-Flop1 uses $\overline{CLR}$ and $\overline{PR}$ inputs to control BUSY signal and is a consumer with respect to them.
- In addition, the Parallel Input Unit has three Jumpers connected to 'Paper Empty', 'Printer Selected' and 'Error' output signals. The Jumpers are manually switched address components which connect either 5V or Ground to these output signals. They have not been shown in the figure.

6.2.6 The Parallel Output Unit

The Parallel Output Unit sends files to the printer in parallel fashion. It consists of CPU, Monostable Multivibrator, Schmitt Trigger Inverter, D Flip-Flops, OR gates and Decoder. Class model of the Parallel Output Unit is shown in Figure 6.8.

The Parallel Output Unit sends a new byte of data to the printer when $\overline{DREQ1}$ (Data Request) input of the CPU is low. $\overline{DREQ1}$ signal is low only when BUSY signal is low and $\overline{ACK}$ signal has been received from the printer for the last byte of data sent. However, since $\overline{ACK}$ signal is of limited duration, D Flip-Flop2 (D FF2) is used to register its arrival as follows: the positive-edge triggered D Flip-Flop2 sets its output to 0 (based on its D input connected to Ground) when $\overline{ACK}$ goes high. Its output remains at 0 unless and until it is set to 1 by a low $\overline{PR}$ input.

When the CPU receives a low $\overline{DREQ1}$ signal, the Parallel Output Unit sends a new byte of data to the printer by loading it into D Flip-Flop1 (D FF1) and issuing a low $\overline{Strobe}$ signal. In order to load the data into D Flip-Flop1, the CPU selects it by issuing a low $\overline{WR}$ signal in addition to the signals used by the Decoder to generate $\overline{I/O5}$ signal (as explained in the Configuration Switches Unit). These signals, viz., $\overline{WR}$ and $\overline{I/O5}$ are combined to generate the clock signal necessary for D Flip-Flop1 to store the byte present on data bus D0-D7. When the data has been written into D Flip-Flop1, the CPU unselects it. As a result, B input of the Monostable Multivibrator rises to 1 and triggers it to issue a low $\overline{Strobe}$ output of fixed time duration. This low $\overline{Strobe}$ output in turn sets the output of D Flip-Flop2, and thereby, $\overline{DREQ1}$ input of the CPU to 1. The 1-byte data stored in D Flip-Flop1 is now available to be read by the printer through output D0-D7.

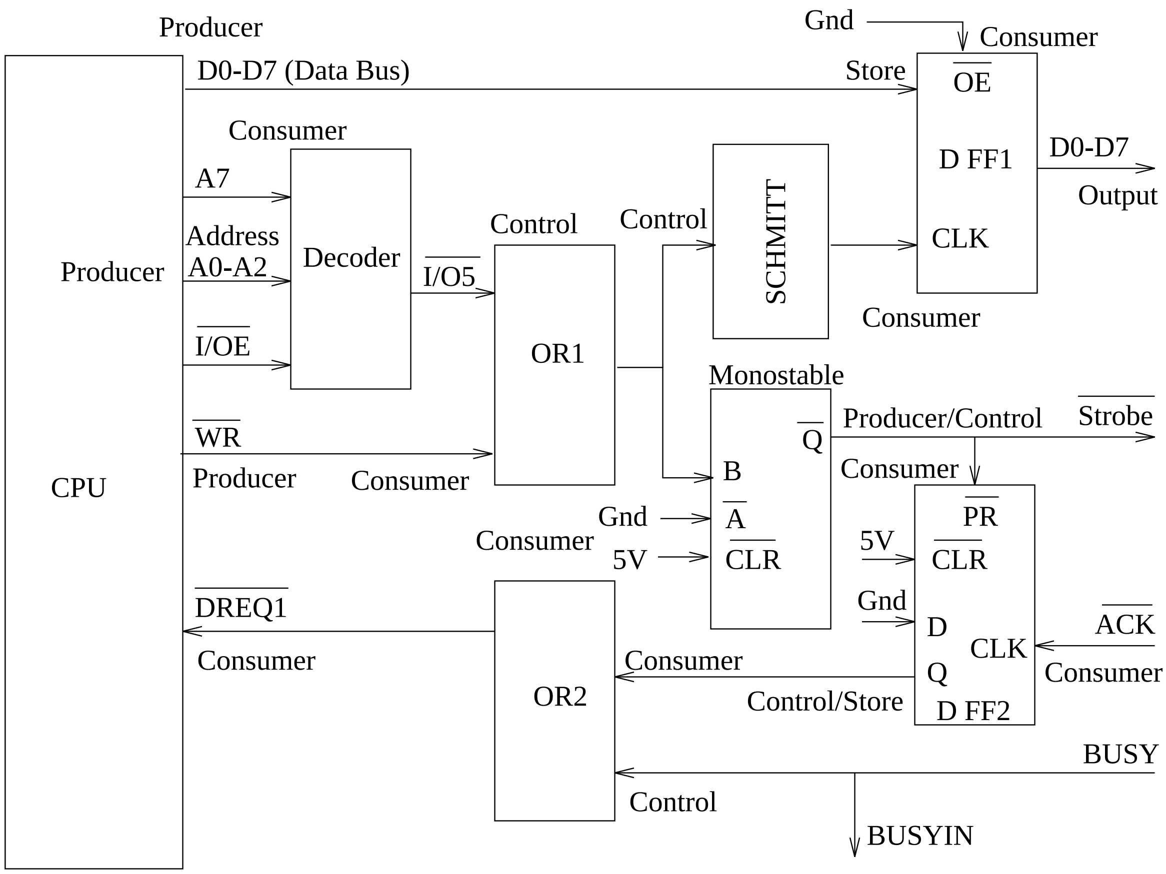


Figure 6.8: Class Model of the Parallel Output Unit

Class model of the Parallel Output Unit (See Figure 6.8) consists of the following signal-lines:

- BUSY input is controlled by OR2 gate and consumed by the CPU at its $\overline{DREQ1}$ input. This is a control signal-line with respect to the CPU, since it prompts the CPU to begin transfer of the next byte of data.

- $\overline{ACK}$ input is consumed by D Flip-Flop2. It is a clock signal-line with respect to D Flip-Flop2.

- D Flip-Flop2 is a control/store component with respect to Ground signal connected to its D input. This signal is consumed by OR2 gate. The resulting signal-line is a control signal-line with respect to OR2.

  Supply Voltage (5V) connected to $\overline{CLR}$ input of D Flip-Flop2 forms a control signal-line with respect to D Flip-Flop2 and is consumed by it.

- The CPU is the producer of $\overline{WR}$ signal consumed by OR1. $\overline{WR}$ is a control signal-line with respect to OR1.

- Signal-lines necessary to generate $\overline{I/O5}$ signal have been described as part of the Configuration Switches Unit. The Decoder is an address component with respect to this signal. OR1 and the Schmitt Inverter control the signal and D Flip-Flop1 consumes it. The resulting signal-line *CPU-Decoder-OR1-Schmitt-D Flip-Flop1* is a clock signal-line with respect to D Flip-Flop1.

  Output of OR1 is also consumed by the Monostable Multivibrator. The resulting signal-line *CPU-Decoder-OR1-Monostable* is a control signal-line with respect to the Monostable Multivibrator.

- The Monostable Multivibrator is the producer/control of $\overline{Strobe}$ signal which is output by the Parallel Output Unit. The Multivibrator controls production of the signal based on its inputs B, $\overline{A}$ and $\overline{CLR}$. The $\overline{Strobe}$ signal is also consumed by D Flip-Flop2 and is a control signal-line with respect to it.

- $\overline{A}$ and $\overline{CLR}$ inputs are consumed by the Monostable Multivibrator. They are control signal-lines with respect to it. Again, note that, the resistor and capacitor which fix the duration of $\overline{Strobe}$ pulse have been encapsulated [Hamscher, 1988] into the Monostable Multivibrator in this model.

- The CPU produces data bus signal D0-D7 which is stored by D Flip-Flop1 and is connected to output D0-D7 of the Parallel Output Unit.

- Control input $\overline{OE}$ is consumed by D Flip-Flop1. It is set to constant 0, thereby permanently enabling the output of the Flip-Flop.

- In addition, the Parallel Output Unit also has three Jumpers connected to 'Auto Linefeed', 'Reset Printer' and 'Select' output signals. The Jumpers are manually switched address components which connect either 5V or Ground to these output signals. They have not been shown in the figure.

6.2.7 The RAM Unit

The Random Access Memory (RAM) Unit stores the files sent by the computer(s) till they can be printed on the printer(s).

The printer buffer comes with two options: either 8 RAM chips are interleaved to provide 64 Kbytes or 16 chips are interleaved to provide 512 Kbytes of memory. Multiplexers are used to properly address all the chips, whether 64 Kbytes or 512 Kbytes of RAM is installed. In the following discussion, the printer buffer is assumed to have 64 Kbytes of RAM.

For simplicity, the RAM Unit is considered in two parts: the Address Decode Unit and the Dynamic Refresh Unit. The Address Decode Unit accesses individual locations in the RAM. The Dynamic Refresh Unit periodically rewrites the contents of the RAM chips. Rewriting is necessary to prevent the dynamic RAM chips from gradually losing their contents due to leakage.

RAM Address Decode Unit

The Address Decode Unit consists of Multiplexers (MUX), RAM chips and CPU. Its class model is shown in Figure 6.9. Each of the 8 interleaved RAM chips holds 64 Kbits of information, and hence, requires 16 bits of address input. However, each RAM chip has only 8 pins dedicated to address input. The 16 bits of address are time-multiplexed through these 8 pins, i.e., each two-byte address is sent into the chip as a sequence of two one-byte addresses. The Multiplexers are used to time-multiplex the two halves of the address.

The CPU outputs the appropriate address on the address bus A0-A18. E signal generated by the CPU enables the Multiplexers to choose between A0-A7 and A8-

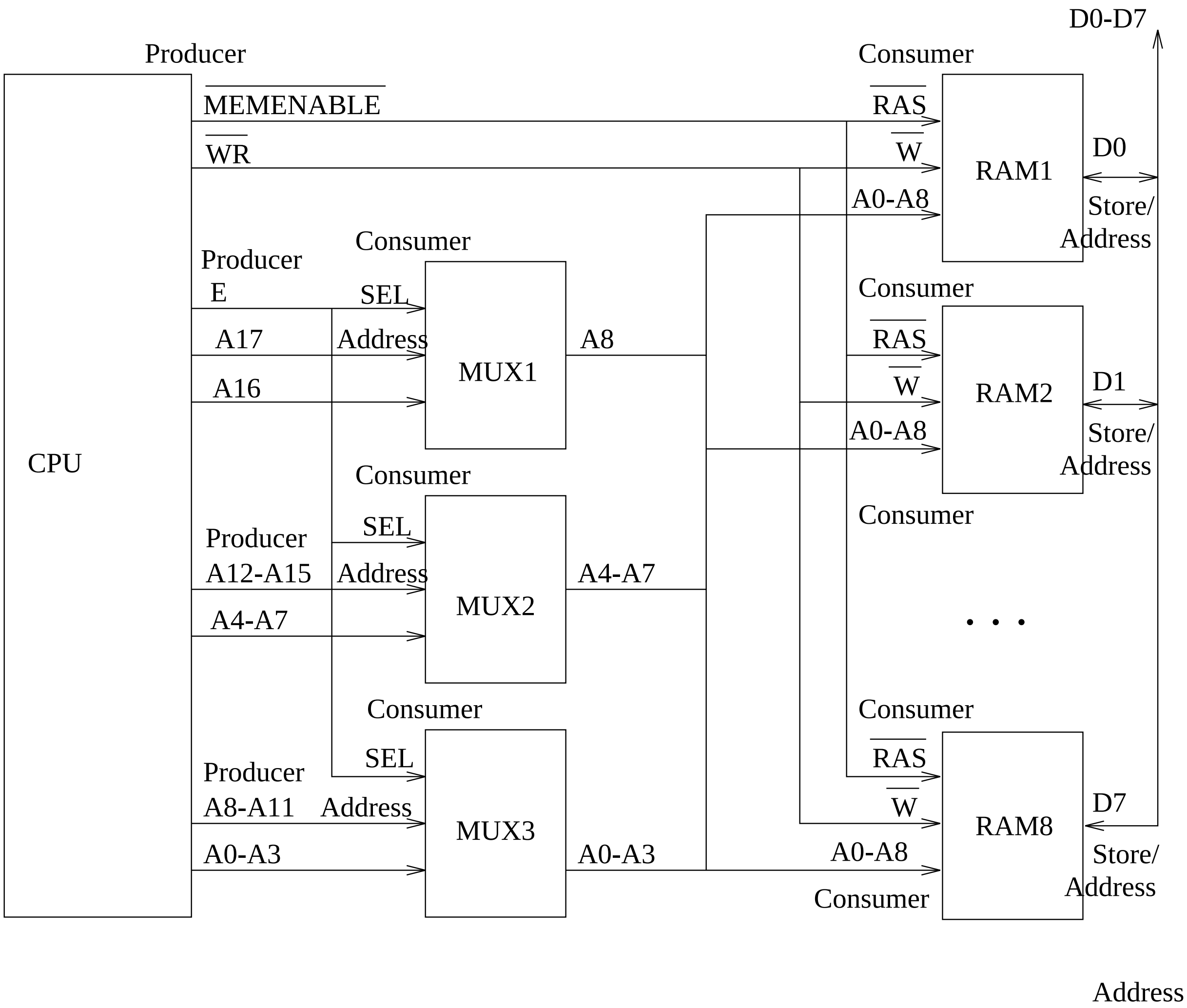


Figure 6.9: Class Model of the RAM Address Decode Unit

A15. The chosen 8-bit address is fed to address inputs of the RAM chips. When the RAM chips have assembled a 16-bit address from two time-multiplexed 8-bit addresses, they are ready to read and write. If their $\overline{W}$ input is low, the data on the data bus D0-D7 is written into the location selected by the address. On the other hand, if $\overline{RAS}$ input is low, contents of the RAM location selected by the address are copied to the data bus.

Class model of the Address Decode Unit (See Figure 6.9) consists of the following signal-lines:

- The CPU is the producer of E signal which is consumed by the Multiplexers at their SELECT inputs. This is an address signal-line with respect to the Multiplexers.

- The CPU produces the signal A0-A17. The Multiplexers are address components with respect to this signal. Outputs of the Multiplexers are consumed by the RAM chips [7]. The signal-line *CPU-Multiplexer-RAM* is an address signal-line with respect to the RAM chips.

- The CPU is the producer of two other signals, i.e., $\overline{MEMENABLE}$ and $\overline{WR}$, both of which are consumed by the RAM chips. Both are control signal-lines with respect to the RAM chips. They enable the RAM chips to release (read) and save (write) data respectively.

- The RAM chips are store/address components with respect to the data bus D0-D7.

RAM Dynamic Refresh Unit

The Dynamic Refresh Unit consists of RAM chips, D Flip-flops, Schmitt Trigger Inverters, 3-input NAND gates and CPU. Schmitt1, Schmitt2, D Flip-Flop1 (D FF1) and NAND1 refresh the lower 8 RAM chips $RAM1, \ldots, RAM8$, and are described here. Class model of the Dynamic Refresh Unit is shown in Figure 6.10.

The lower 8 RAM chips are refreshed when their $\overline{CAS}$ input is low. Whenever E signal generated by the CPU is low, outputs of D Flip-flops, and hence, $\overline{CAS}$ inputs of the RAM chips are set to 1, which disables RAM refresh. $\overline{CAS}$ signal is set to 0 only when all of the following occur:

[7] The RAM chips used in the 64 Kbyte configuration of printer buffer use A0-A7 bits. A8 bit generated by MUX1 in the figure is used only in the 512 Kbyte configuration.

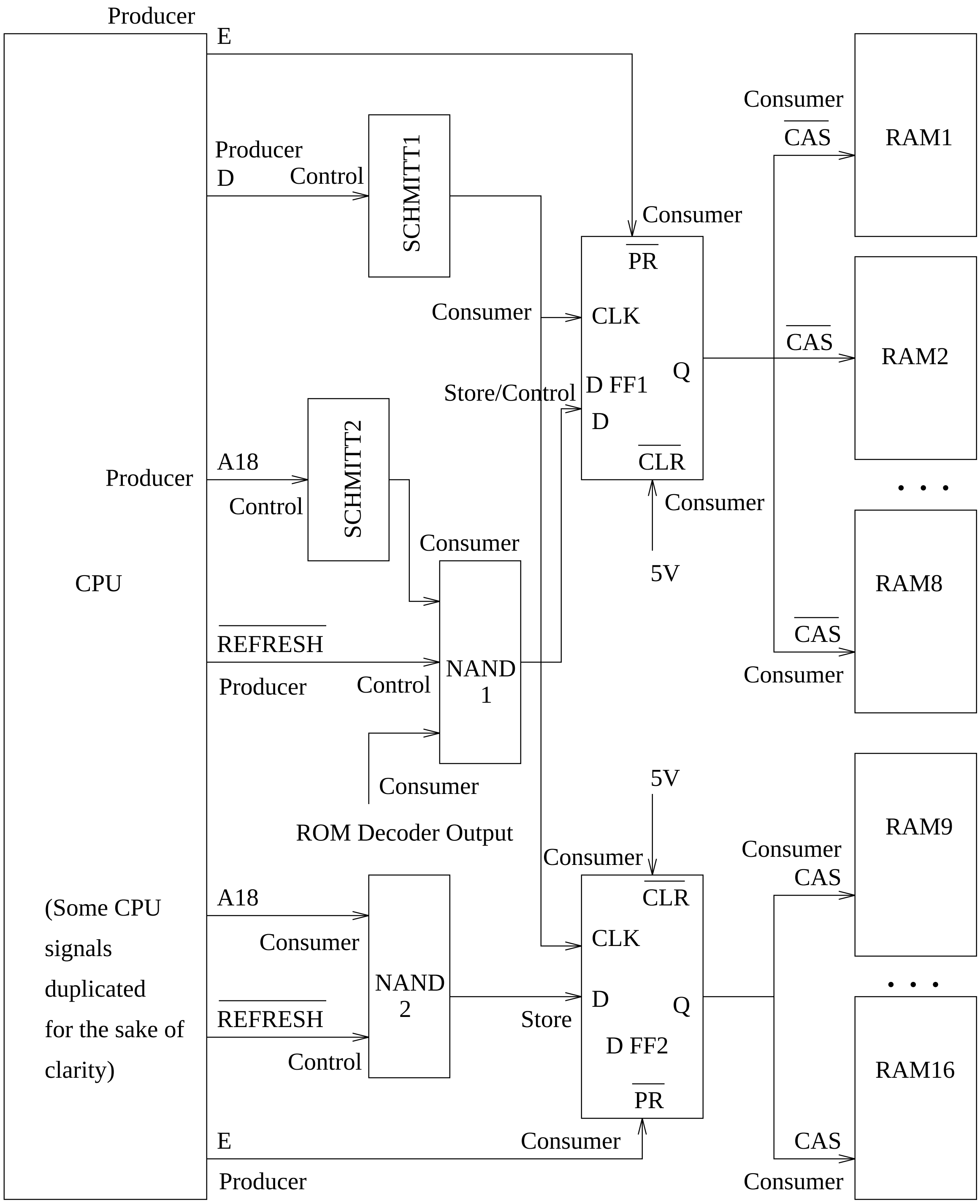


Figure 6.10: Class Model of the RAM Dynamic Refresh Unit

- ROM Decoder output is 1, i.e., the ROM chip is not selected in Figure 6.3;
- $\overline{REFRESH}$ output of the CPU is 1;
- Address bit A18 produced by the CPU is 1;
- E signal produced by the CPU is 1;
- and D signal produced by the CPU makes a transition from 1 to 0.

When all the above signals are present, NAND1 outputs 0, which is input to D Flip-Flop1. D Flip-Flop1 sets its output connected to $\overline{CAS}$ inputs of the RAM chips to 0, thereby enabling RAM refresh.

Following are the signal-lines in the class model of the Dynamic Refresh Unit (See Figure 6.10):

- Signal E produced by the CPU is consumed by the D Flip Flops. This is a control signal-line with respect to the D Flip Flops.
- The CPU produces signal D which is controlled by Schmitt1 Inverter. The D Flip-Flops are consumers of this signal. The resulting signal-line *CPU-Schmitt1-D Flip Flop* is a clock signal-line with respect to the D Flip Flops.
- The CPU produces signal A18 which is controlled by Schmitt2 Inverter and consumed by NAND1. The signal-line *CPU-Schmitt2-NAND1* is a control input with respect to NAND1.
- Output of the ROM Decoder is consumed by NAND1 gate. The resulting signal-line *CPU-ROM Decoder-NAND1* shown in Figure 6.3 is a control signal-line with respect to NAND1.
- The CPU produces signal $\overline{REFRESH}$, which is controlled by NAND1. NAND1 controls the signal based on its inputs from the ROM Decoder and Schmitt2 Inverter. D Flip-Flop1 is a store/control with respect to this signal. The lower 8 RAM chips are consumers of the signal in the form of $\overline{CAS}$ input. The resulting signal-line *CPU-NAND1-D Flip-Flop1-RAM* is a control signal-line with respect to the RAM chips because it refreshes their contents. Without it, the RAM chips would lose all their contents.
- $\overline{CLR}$ inputs of the D Flip-Flops are control signals consumed by those components.

Additional signal-lines necessary to refresh the higher eight RAM chips $RAM9, \ldots, RAM16$ are:

- The CPU produces signal A18 which is consumed by NAND2. This input is a control signal-line with respect to NAND2.
- *CPU-NAND2-D Flip-Flop2-RAM* signal-line is a control signal-line with respect to the RAM chips.

6.2.8 The Panel Button Unit

The Panel Button Unit enables the user to touch select various printing options offered by the printer buffer. It consists of PushButtons, Buffer, (Quad) D Flip-Flop, Decoder and CPU.

PushButton functions provided by the printer buffer are:

- **Clear** or delete all the files received from the port.
- **Copy** or print multiple copies of the next file received from the port.
- Priority **Print** the next file received from the port.
- **Restart** printing the current file.
- **Offline**, i.e., stop sending data to the output port. The input port however, continues to accept data.
- **Swap** or change the data path between the input and output ports. In all, four datapaths can be configured between the two input and two output ports.

All the above functions except Swap are provided for both serial and parallel ports. In Figure 6.11 as well as the ensuing discussion, the functions are prefixed with P (for parallel) and S (for serial) to indicate the specific port they affect.

The CPU scans the PushButtons once every 25 milliseconds. For instance, in order to scan the Swap button, it sets output DP8 of the D Flip-Flop to 1 by:

- selecting it through $\overline{I/O8}$ signal generated by the Decoder (as explained in the Configuration Switches Unit);
- setting bit D0 to 1;

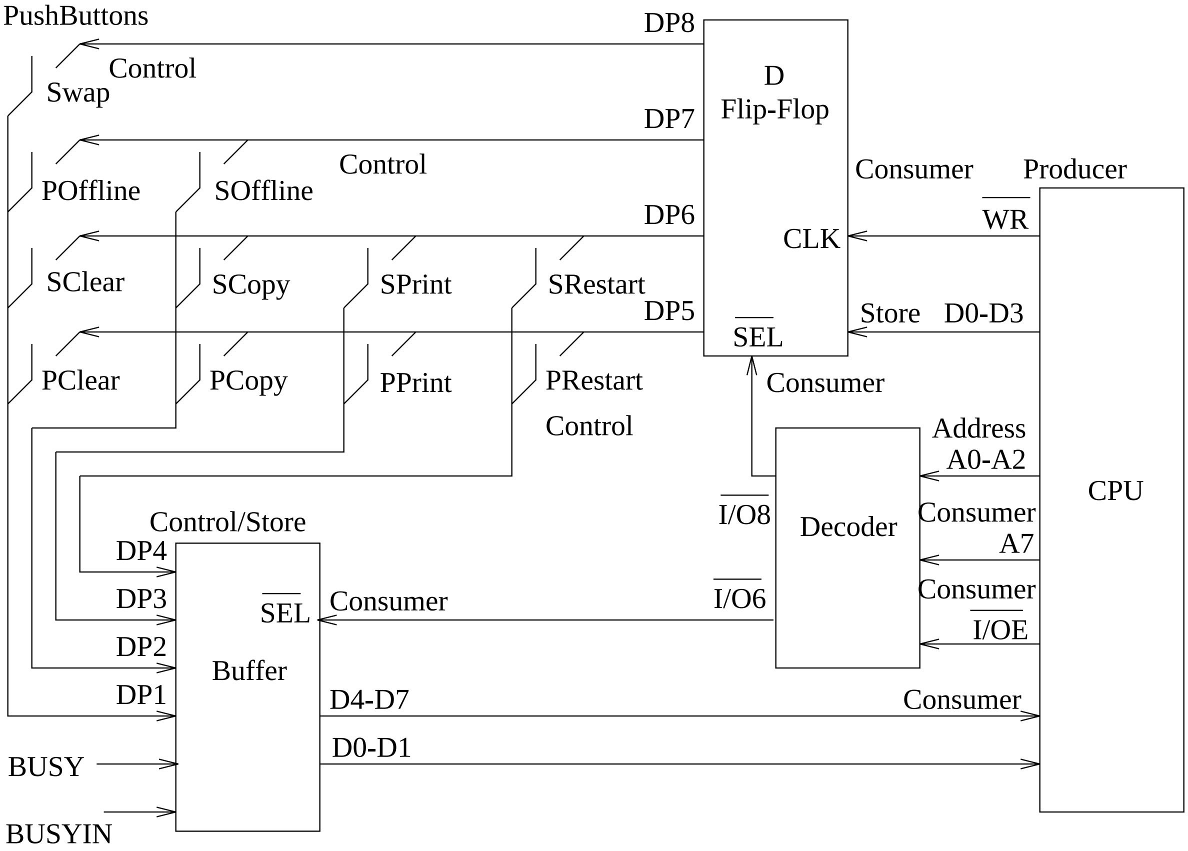


Figure 6.11: Class Model of the Panel Button Unit

- and issuing a low $\overline{WR}$ signal. The D Flip-Flop sets its output DP8 to 1 on the rising edge of $\overline{WR}$ signal.

If the Swap button is open, the default low of DP1 is stored in the Buffer. If the Swap button is closed, the high of DP8 is stored in it.

The CPU now selects the Buffer through $\overline{I/O6}$ signal generated by the Decoder, so that the Buffer sets its outputs to the values at its inputs. Therefore, D7 is set to the value of DP1, i.e., 1, which is then read by the CPU as the status of the Swap button.

Similarly, if the CPU wants to scan the serial PushButtons, it sets D2, and thereby, output DP6 of the D Flip-Flop to 1. Next, it selects the Buffer in order to read the status of SClear, SCopy, SPrint and SRestart buttons through D7 (DP1), D6 (DP2), D5 (DP3) and D4 (DP4) bits respectively.

Class model of the Panel Button Unit consists of the following signal-lines (See Figure 6.11):

- $\overline{WR}$ signal is produced by the CPU and consumed by the D Flip-Flop. It is a clock signal-line with respect to the D Flip-Flop.

- Signal-lines necessary to generate $\overline{I/O6}$ and $\overline{I/O8}$ signals have been described as part of the Configuration Switches Unit. The Decoder is an address component with respect to these signals. $\overline{I/O8}$ is consumed by the D Flip-Flop and the signal-line *CPU-Decoder-D Flip-Flop* is a control signal-line with respect to it. $\overline{I/O6}$ is consumed by the Buffer and the signal-line *CPU-Decoder-Buffer* is a control signal-line with respect to it.

- Signal D0-D3 is produced by the CPU and stored by the D Flip-Flop. It fans out into a separate signal-line for each PushButton. PushButton is a control component with respect to its signal-line. The Buffer is a control/store with respect to these signal-lines which are consumed by the CPU through D4-D7. For instance, the signal-line *CPU-D Flip-Flop-Swap-Buffer-CPU* is a data signal-line with respect to the CPU.

- BUSY signal generated by the Parallel Input Unit (See Figure 6.7) and BUSYIN signal obtained by the Parallel Output Unit (See Figure 6.8) are both controlled/stored by the Buffer and consumed by the CPU through D0-D1. They are data signal-lines with respect to the CPU.

6.3 The Display Unit

In this section, representation and reasoning in Classes is illustrated using the Display Unit of the printer buffer (See Figure 6.12).

The Display Unit consists of three LEDs and a Swap diode. Each LED in turn consists of a seven-segment display and a decimal point (DP). The amount of available memory is displayed on the LEDs. The Swap diode lights up when serial and parallel ports are crossed by pressing the Swap panel button. The CPU controls the LEDs and the Swap diode through 8-bit Buffers, and refreshes their displays only on change.

6.3.1 Modeling the Display Unit

The Display Unit (See Figure 6.12) is modeled using the deterministic representation algorithms described in Section 4.1.

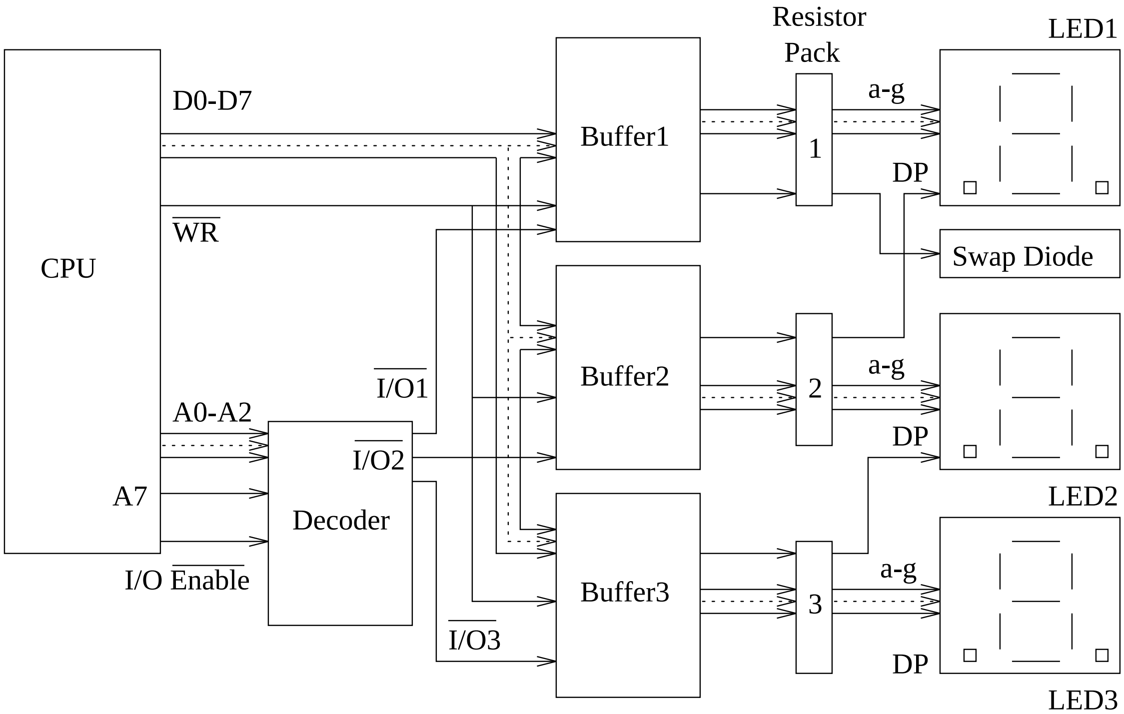


Figure 6.12: Display Unit of the Printer Buffer

Class models of components of the Display Unit are shown in Figure 6.13. The ports of each component in the figure are labeled as follows: the class of the component with respect to the port is specified, followed by the class of the port/signal-line with respect to the component[8]. For instance, the *Swap Diode* is a consumer with respect to its input and the input is a data signal-line with respect to the *Swap Diode*. The *LED*s and *Swap Diode* are consumers of electrical signal, their output being in light element. The *Resistor Pack* is a non-ideal data component. The *CPU* is the producer of signals on the data bus *D0-D7*, address bus *A0-A2,A7* and outputs $\overline{WR}$ and $\overline{I/OEnable}$ ($\overline{I/OE}$). The *Decoder* is an address component because it redirects its input signal to one of its several outputs. The *Buffer* is both a store and a control component. Note that in the Display Unit, the input for all three buffers is from the data bus *D0-D7*. A particular buffer is selected to store data from *D0-D7*, through $\overline{I/O}$ signals generated by the *Decoder*. Therefore, the *ensemble* of three buffers and the *Decoder* acts as an address component in the Display Unit. Although $\overline{I/O}$ is a control input with respect to the *Buffer*, it serves as an address signal-line in the

[8] For *Buffers* and *Decoders* in particular, names of the signals connected to them in the Display Unit are specified in parenthesis in order to disambiguate their ports.

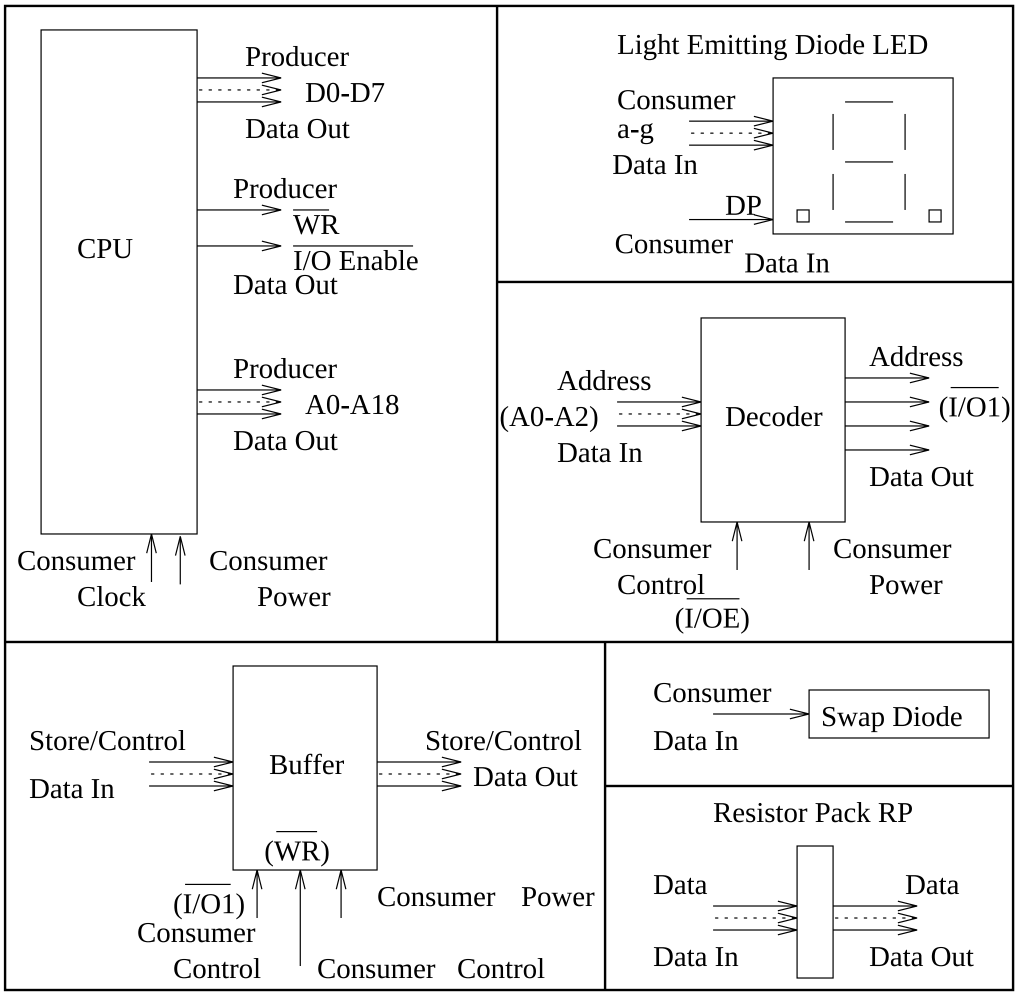


Figure 6.13: Class Models of Components in the Display Unit

Display Unit.

The *DR Algorithm* (See Section 4.1.2) starts from the outputs of the printer buffer given by the set { LED1, LED2, LED3, Swap Diode } (See Figure 6.12). It identifies signal-lines with respect to every link incident on each of these outputs, using the *DS Algorithm* (See Section 4.1.1). For example, the links incident on $LED1$ are set $a - g$ corresponding to the seven-segment display, and input *DP* of the decimal point. Consider the input set $a - g$. It is connected to data outputs of the resistor pack *RP1*, which is a non-ideal data component. Corresponding data inputs of *RP1* are connected to data outputs of *Buffer1*, which is a combination of store and control. Corresponding data inputs of *Buffer1* are connected to data outputs of the *CPU*, which is the producer of the signal. Therefore, the signal-line generated with respect

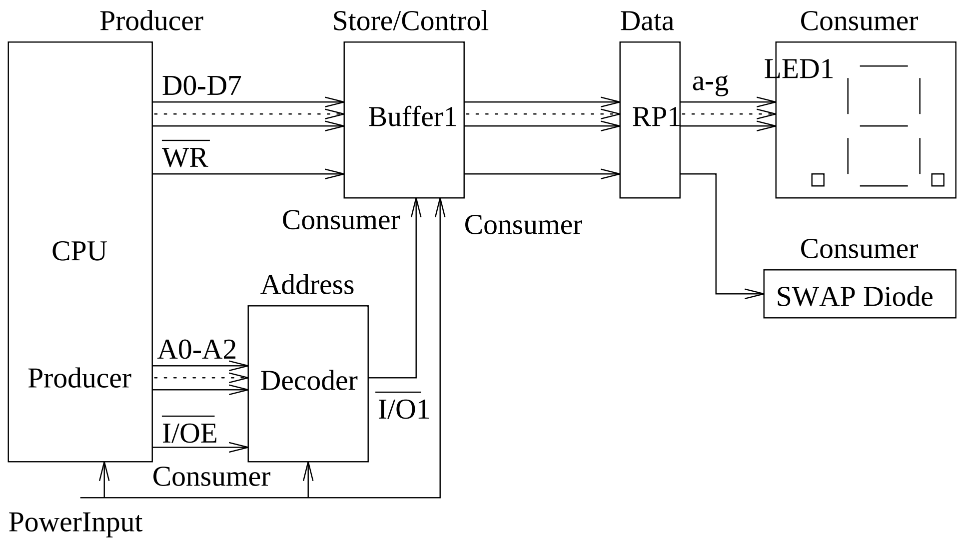


Figure 6.14: Partial Class Model of the Display Unit

to the input set $a - g$ of *LED1* is *CPU-Buffer1-RP1-LED1*. Similarly, the other signal-lines generated are *CPU-Buffer2-RP2-LED1* with respect to input *DP* of *LED1*, *CPU-Buffer1-RP1-Swap Diode*, *CPU-Buffer2-RP2-LED2*, *CPU-Buffer3-RP3-LED2* and *CPU-Buffer3-RP3-LED3*.

In the next step, the *DR* algorithm identifies power, clock, address and control signal-lines connected to each component in the above signal-lines. *LED1*, *RP1*, *Buffer1* and *CPU* are the components in the first signal-line. *LED1* and *RP1* have no other signal-lines connected to them. *Buffer1* has address ($\overline{I/O1}$), control ($\overline{WR}$) and power signal-lines connected to it. *Buffer1* is a consumer of $\overline{I/O1}$ signal received from *Decoder*. *Decoder* is an address component with respect to $\overline{I/O1}$ signal, and its data inputs corresponding to $\overline{I/O1}$ output are $A0 \ldots A2$ produced by the *CPU*. Therefore, the address signal-line identified with respect to *Buffer1* is *CPU-Decoder-Buffer1*. Similarly, all the other signal-lines in the Display Unit are generated.

A partial class model of the Display Unit is shown in Figure 6.14. It consists of the power signal-line and the signal-lines terminating in *LED1*, *Swap Diode* and *Buffer1*.

6.3.2 Diagnosing the Display Unit

In this section, function based diagnosis discussed in Chapter 5 is applied to the class model of the Display Unit shown in Figure 6.14. First, two hypothetical diagnostic scenarios are considered to illustrate the use of teleological technique for candidate discrimination. Next, a sample session from our implemented system diagnosing the Display Unit is explained.

Scenario 1: Suppose the symptom at *LED1* is that its display changes at random times and to random values. This symptom translates to invalid transition. *CPU-Buffer1-RP1-LED1* is the only signal-line connected to the seven segment output of *LED1*. (The other signal-line *CPU-Buffer2-RP2-LED1* is connected to the decimal point *DP* of *LED1* which is not part of the observed symptom.) Based on the feature-class associations listed in Section 5.1.1, store component in this signal-line, i.e., *Buffer1* is suspected first.

Since *Buffer1* is also a control component, peripheral diagnosis is performed first: other signal-lines connected to it are checked. By constraint suspension, the symptom at *LED1* translates to incorrect value written to *Buffer1* or invalid transition in its contents or both. Therefore, based on the feature-class associations, address signal-line *CPU-Decoder-Buffer1* and control signal-line *CPU-Buffer1* are suspected first.

Incorrect value written to *Buffer1* translates to incorrect value at the output of the address signal-line. Therefore, based on the feature-class associations, address component, i.e., *Decoder* is suspected before the producer (*CPU*).

Scenario 2: Suppose the Swap diode in Figure 6.14 does not light up when the Swap button is pressed. The only signal-line connected to the symptom is *CPU-Buffer1-RP1-Swap Diode*, and it is considered. Since the symptom is that the output value does not *exist*, based on the feature-class associations, producer, i.e., the *CPU* is suspected first. A test of the *CPU* is initiated. However, if the *CPU* is found to be fault-free, peripheral diagnosis is carried out on it. Based on the feature-class associations, its power signal-line is tested first.

Note that, in both the above scenarios, the diagnosis algorithm backtracks if necessary, until the fault is found or all the components in the unit have been examined.

Sample Session

The sample session given in Appendix A shows the application of our implemented system to diagnose the class model of the Display Unit of the printer buffer. The symptom considered is that *LED*s display an available memory of 338 Kbytes instead of 333 Kbytes. Default order technique is used for candidate discrimination. Greedy order is used for all the candidates, i.e., component diagnosis is performed before peripheral diagnosis.

LEDs are seven-segment displays. For convenience, their output has been translated to a number. For example, the digit "6" displayed by an *LED* is numerically represented as 125, whose binary representation specifies the segments of the *LED* that are lit. The decimal point of an *LED* is considered separate from its seven-segment display.

Initially, the diagnosis system obtains measured values at the inputs and outputs of the Display Unit. Note that *Out1*, *Out4* and *Out6* are the seven-segment displays of *LEDs*. *Out2*, *Out5* and *Out7* are the decimal points of *LEDs*. *Out3* is the *Swap-Diode*. *In1* is the power input, *In2* is the clock input and *In3* is ground. Since the printer buffer is a sequential device, the system also prompts for the output values of the *CPU*, in order to obtain a snap-shot of the state of the device.

After obtaining all the above values, the diagnosis system loads models of all the components in the Display Unit. It performs simulation and constraint suspension at the level of the loaded components. As a result, it identifies output *Out1* of the Display Unit to be faulty. It considers the only signal-line connected to the symptom, viz., *CPU-Buffer1-RP1-LED1*.

The system orders components in the signal-line according to the default order of their classes. Therefore, the ordered candidate-list is { *LED1*, *CPU*, *Buffer-1*, *RP1* }. Since *LED1* is a consumer, it is at the top of the list. The system prompts for measurements across *LED1*.

Measurements indicate that *LED1* is not faulty. Therefore, the system backtracks to consider the producer, i.e., the *CPU*. When the *CPU* is found to be fault-free, it performs *peripheral diagnosis*, i.e., attempts to diagnose signal-lines connected to the other inputs of the *CPU*. It suggests that clock input to the *CPU* may be faulty because it cannot be eliminated on the basis that it is also connected to other working components. On the other hand, it exonerates power signal-line because it is connected to the buffers and *Decoder*[9].

The next component to be considered in the signal-line is the store, i.e., *Buffer1*. The system initiates measurements across *Buffer1*. Since measurements suggest that *Buffer1* is not faulty, it performs *peripheral diagnosis* on it. Three signal-lines are connected to *Buffer1*, viz., control signal-line *CPU-Buffer1*, address signal-line *CPU-Decoder-Buffer1* and the power signal-line. A candidate-list is generated, consisting of these three signal-lines ordered by their class.

Since the *CPU* has already been eliminated from the candidate-list, the system

[9]This is a result of candidate validation by constraint suspension.

trivially checks the control signal-line. In the address signal-line, it suspects the *Decoder* first. Measurements indicate that the *Decoder* is indeed faulty. The *Decoder* failed to select *Buffer1*. Therefore, *Buffer1* did not register its input (the unit 3 of 333K), and *LED1* displayed the random contents of *Buffer1*.

6.4 The Carburetor

The automobile carburetor has been chosen to illustrate the domain-independence of Classes. In this section, its class model is discussed and a hypothetical diagnostic scenario is presented.

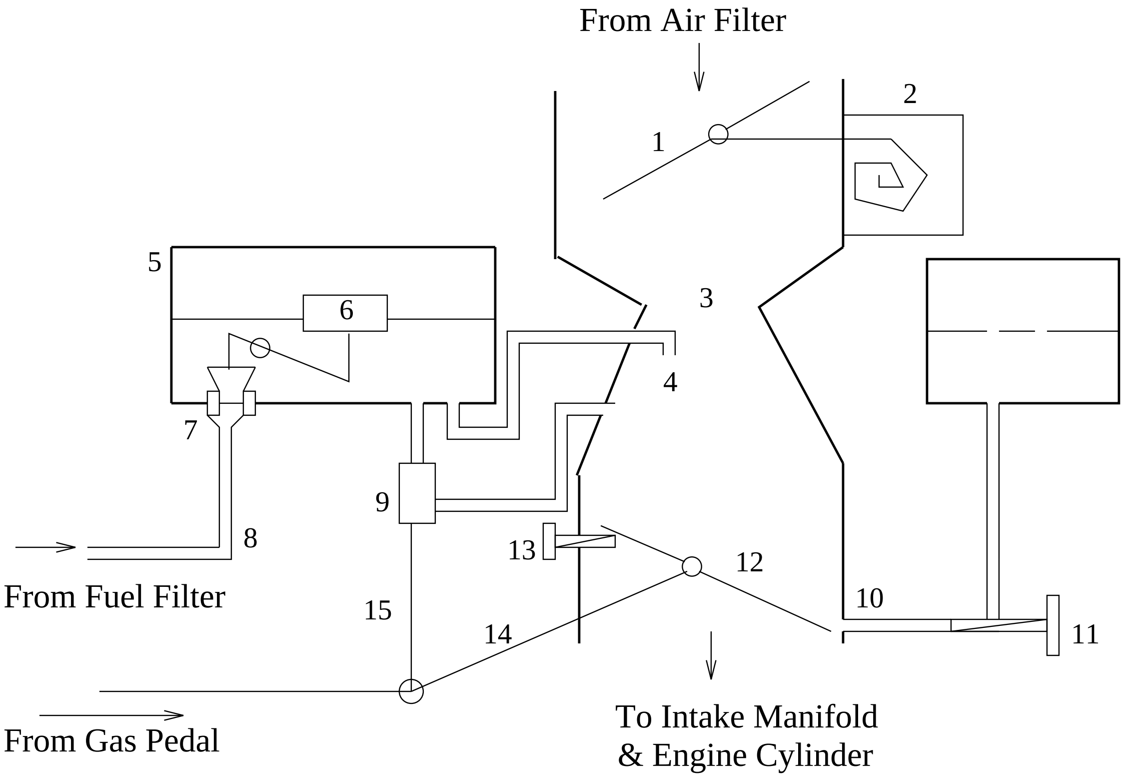
The carburetor in a car manages the production and composition of fuel-air mixture, which is later ignited in the engine to produce power by internal combustion. One barrel of the carburetor is shown in Figure 6.15. It is a simplified schematic, with none of the linkages shown.

In Figure 6.15, the float bowl(5) holds a small amount of fuel ready for use. The amount of fuel in it is regulated by the float (6). The float unplugs the needle/seat assembly (7) to let in more fuel from the fuel pump whenever the level of fuel in the bowl is low.

As air passes the main metering orifice (4), it picks up some fuel due to Venturi effect. The resulting fuel-air mixture passes past the throttle (12) into the engine cylinder, to be ignited. When the car is cold, fuel condenses on the cold metal parts, leaving the fuel-air mixture too lean (i.e., starved for fuel). In order to counter this, the choke thermostat (2) closes the choke butterfly (1) in a cold engine. Now, less air travels past the choke butterfly, more vacuum is created in the barrel (3) and hence, more fuel is drawn out to create a richer fuel-air mixture. When the car warms up, the thermostat opens the butterfly and admits more air into the barrel. The fuel-air mixture now returns to normal.

When a driver steps on the gas pedal to accelerate, the accelerator pump (9) squirts extra fuel into the barrel, and creates a richer fuel-air mixture which explodes better in the engine cylinder. Also connected to the gas pedal is the throttle (12), which controls the speed of the car by controlling the amount of fuel and air sucked into the carburetor barrel. The more open the throttle, the greater the speed of the car.

The idle speed screw (13) keeps the throttle partially open even when the driver's foot is off the gas pedal. This enables the car to idle and not stall for lack of fuel-air



1. Choke Butterfly
2. Choke Thermostat
3. Barrel
4. Main Metering Orifice
5. Float Bowl
6. Float
7. Needle/Seat Assembly
8. Fuel Line
9. Accelerator Pump
10. Idle Fuel Inlet
11. Idle Mixture Screw
12. Throttle
13. Idle Speed Screw
14. Throttle Arm
15. Accelerator Pump Arm

Figure 6.15: The Carburetor Barrel

mixture. The idle mixture screw (11) controls the richness of fuel-air mixture when the car is idling.

6.4.1 Modeling the Carburetor

The carburetor has several elements including fuel, air and fuel-air mixture. It has several signals in each of these elements. Class models of components of the carburetor are shown in Figure 6.16. The ports of each component in Figure 6.16 are labeled according to the following convention: Class of the component with respect to the port is specified, followed by the element of flow in parenthesis, followed by class of the signal-line/port with respect to the component. Class model of the carburetor is shown in Figure 6.17.

Class model of the carburetor consists of the following signal-lines:

- Air signal is controlled by the choke butterfly and consumed by the carburetor barrel. This is a data signal-line with respect to the carburetor barrel.

- The choke thermostat is the producer of motion which is consumed by the choke butterfly. This is a control signal-line with respect to the choke butterfly. The choke is itself a consumer of ambient heat.

- Fuel signal is controlled by the needle/seat and float assembly, stored by the float bowl and consumed by the carburetor barrel. The resulting signal-line is a data signal-line with respect to the carburetor barrel.

- The gas pedal is a data component with respect to motion produced by the driver's foot. The motion is consumed by the accelerator pump, and is a control signal-line with respect to it.

- Fuel signal is stored in the float bowl, controlled by the accelerator pump, and consumed by the barrel. It is a data signal-line with respect to the barrel.

- Again, fuel signal is stored in the float bowl, controlled by the idle mixture screw, and consumed by the barrel. It is a data input with respect to the barrel.

  The idle mixture screw in its assembly (See Figure 6.16) is a consumer of motion. Motion is its control input when it is being manually adjusted. However, even after the fact, the screw continues to be a control component with respect to the fuel signal by virtue of its position.

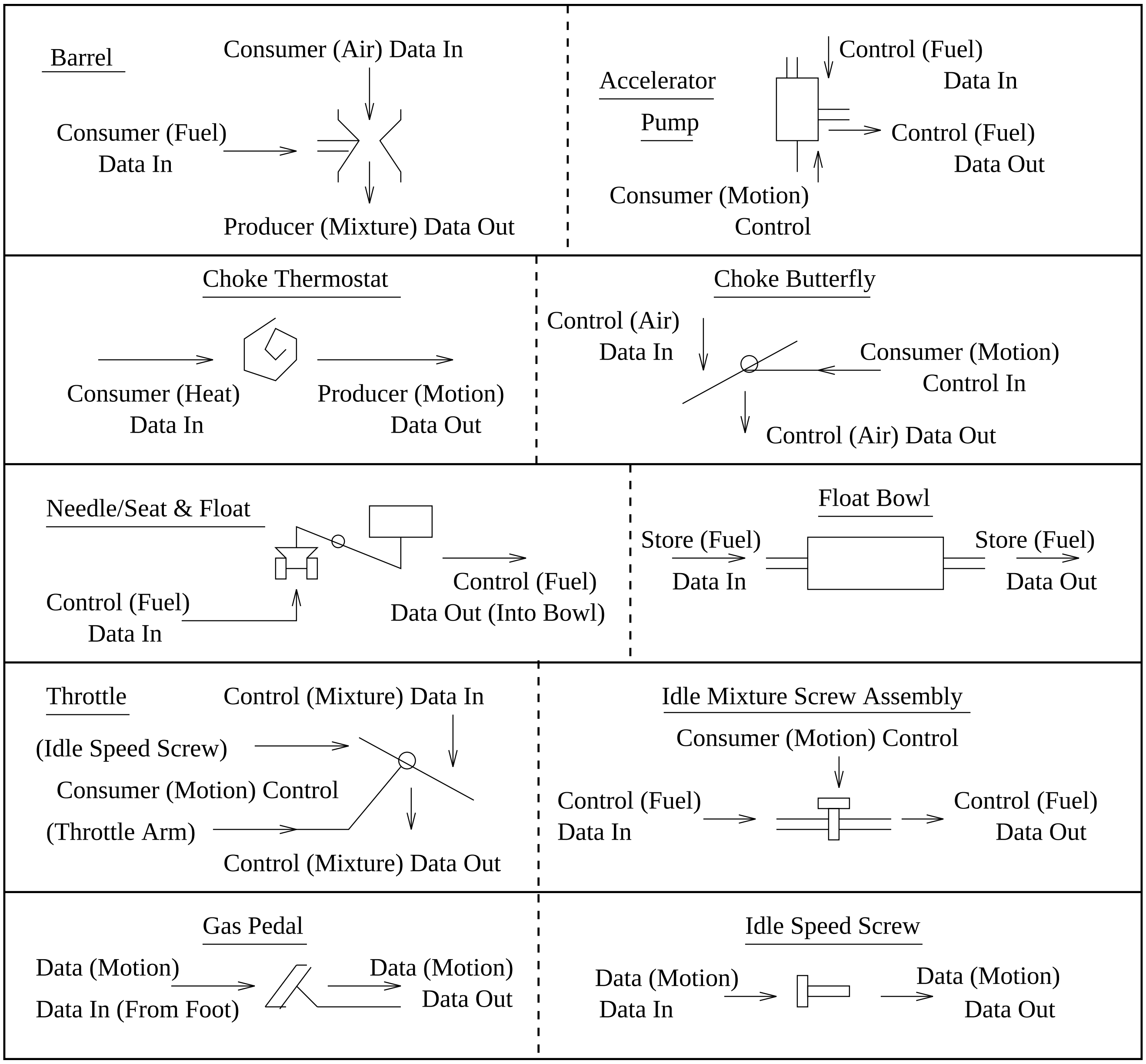


Figure 6.16: Class Models of Components in the Carburetor

- The barrel is the producer of fuel-air mixture, which is controlled by the throttle and consumed by the engine cylinder through the intake manifold. This is a data signal-line with respect to the engine cylinder.

- The gas pedal and the throttle arm are data components with respect to motion generated by the driver's foot. This signal is consumed by the throttle and is a control input with respect to it.

- The idle speed screw is a data component with respect to motion that screws it into place. The throttle is a consumer of this motion when the idle speed screw is being manually adjusted. Even after the activity of adjusting the screw stops, the idle speed screw behaves as a control signal-line with respect to the throttle by virtue of its position.

6.4.2 Diagnosing the Carburetor

Suppose the car does not start. Two data signals are consumed by the engine, viz., fuel-air mixture produced by the carburetor and sparks produced by the electrical unit. These are considered in order. The carburetor barrel is the producer of fuel-air mixture and is suspected first, according to default order. The barrel may be flooded, which explains the symptom. If the barrel is found to be fault-free, by peripheral diagnosis, other signal-lines connected to it are suspected, viz., the air signal-line and the fuel signal-lines connected through the main metering orifice, the idle mixture screw and the accelerator pump.

In the air signal-line, the choke butterfly is a control component and is suspected first, according to default order. By peripheral diagnosis, the control signal-line connected to it is considered. In this signal-line, the choke thermostat is a producer and may be found to be malfunctioning. If not, upon backtracking, the fuel signal-line connected through the main metering orifice is considered. The control component in it, i.e., the needle/seat and float mechanism is suspected first, according to default order. The float may be stuck, thereby preventing the car from starting.

In this scenario, note that the diagnosis moved from "car-does-not-start" symptom to the choke thermostat and the float mechanism in as few as four steps. Further, the diagnosis algorithm did not consider every one of the hundreds of components of the car at each step: it considered only the causally relevant components. This was possible because of the organization of the device model in terms of signal-lines, and

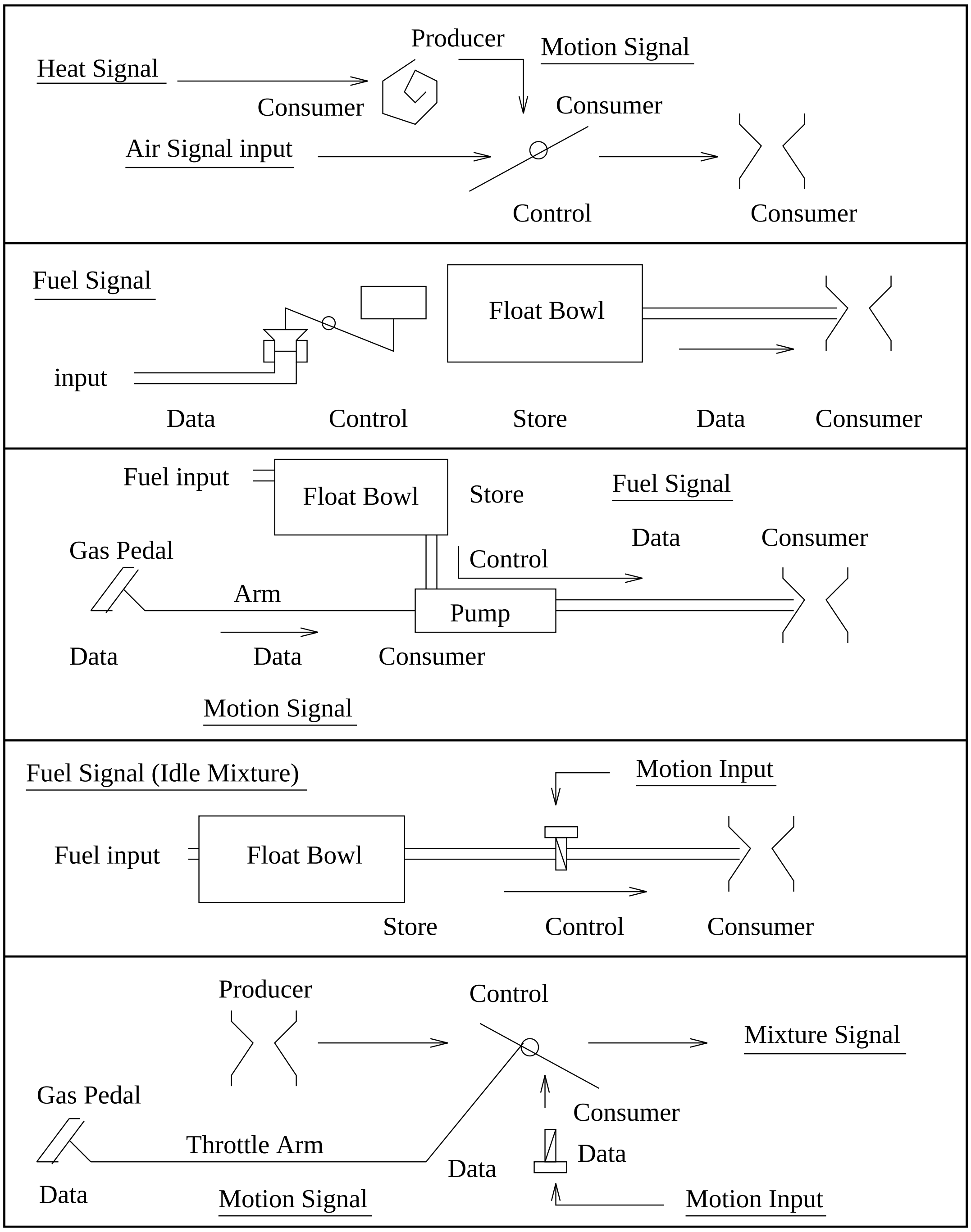


Figure 6.17: Class Model of the Carburetor

highlights a significant advantage of class representation for diagnosis: signal-lines improve the focus of diagnosis.

6.5 Modeling Observations

Following are our observations from modeling the printer buffer and the automobile carburetor using Classes:

- Most of the signal-lines were found to be short, comprising of not more than 4-5 components excluding data components. Several signal-lines were trivial, involving only a producer and a consumer, such as the $\overline{I/OE}$ signal in the Configuration Switches Unit (Figure 6.6).

  Many control components were found to have more than one control input. E.g., Monostable Multivibrator in the Parallel Input Unit (Figure 6.7), NAND1 gate in the RAM Dynamic Refresh Unit (Figure 6.10) and throttle in the carburetor (Figure 6.17). Several store components were found to have inputs which controlled read and write operations. E.g., RAM chips in the RAM Address Decode Unit (Figure 6.9) and D Flip-Flop2 in the Parallel Input Unit (Figure 6.7). Some components belonged to two different classes with respect to the same ports. E.g., In the Parallel Input Unit, the Monostable Multivibrator is a producer/control with respect to its $\overline{ACK}$ output and D Flip-Flop1 is a control/store with respect to its BUSY output.

- It was found too restrictive to have control as the only class of components which modified its output. For instance, $\overline{PR}$ and $\overline{CLR}$ inputs of a D Flip-Flop have been modeled as control signal-lines (E.g., Parallel Input Unit, Figure 6.7). However, it is desirable to be able to discriminate between these two inputs: $\overline{PR}$ sets the output of the D Flip-Flop to 1, whereas $\overline{CLR}$ resets it to 0.

  A few specializations of control class have been listed at the end of Section 3.2.1. These and other classes of components may be added to Classes by defining them with respect to device ports, specifying their position in the default order, and listing feature-class associations applicable to them.

- An advantage of component ontological representation is that function model of a device can be built from compiled models of its components without considering the particular functions that those components carry out in the device. On top of such a representation, it was found helpful to use *overlays* of component function, based on the behavior of an aggregate of components in a particular device.

For instance, in the Display Unit of the printer buffer (Figure 6.12), the ensemble of three Buffers and the Decoder was modeled as an address component overlay, although Buffers themselves are control/store components. This aided in diagnosing the Display Unit. Similarly, the aggregate of three Buffers and the Decoder in the Configuration Switches Unit (Figure 6.6) may be modeled as an address component overlay. These overlays are dependent on specific functions carried out by the components in their device. They must be hand-coded, and cannot be built using the representation algorithms of Chapter 4.

Even otherwise, domain expertise was found necessary to build the function model of a device when class models of its components were ambiguous. For instance, a three-input NAND gate can be a control component with respect to any one of its inputs, the other two being its control inputs. Therefore, in the RAM Dynamic Refresh Unit (Figure 6.10), domain expertise is required to determine that NAND1 is a control component with respect to $\overline{REFRESH}$ signal.

6.6 Summary

The details of implementation of our diagnosis algorithm were presented. A complete class model of an electronic printer buffer was built to demonstrate the scalability of class representation. The Display Unit of the printer buffer was used to illustrate the deterministic algorithms of representation discussed in Chapter 4 and the diagnosis algorithm described in Chapter 5. The class model of an automobile carburetor was built to demonstrate the domain-independence of class representation.

Conclusions of the dissertation follow in the next chapter. Classes is analyzed from several perspectives and limitations of Classes are noted. The contributions of the dissertation are summarized. Pointers are provided on how Classes can be adapted and used by the reader.

Chapter 7

Conclusions

In this chapter, limitations of Classes are listed. Classes is analyzed from several perspectives and its implications to model based reasoning are noted. Contributions of the dissertation are summarized. Pointers are provided on how Classes can be adapted and used by the interested reader.

7.1 Limitations

Classes is a representation of function designed for diagnosis. It is at a coarser level of granularity than state based representations. Therefore, it does not support simulation as well as they do.

The success of applying Classes to reason about a device depends on how varied and dissimilar the classes of its components are. Classes may be inadequate for devices where most of the components have similar functions. E.g., combinational circuits such as a ripple carry adder, where every component belongs to either data or store class. During diagnosis, little if any discrimination is provided among the various components, all of which belong to the same class.

In comparison, Classes is very useful in devices such as the printer buffer, where a great variety of classes occurs. For example, almost every class of components occurs in the function model of the Display Unit discussed in Section 6.3. Therefore, discrimination techniques can be successfully applied to order its components during diagnosis.

The above problem observed in some combinational devices may be solved by appropriately defining subtler distinctions of class primitives. However, no matter what class primitives are chosen, one can always come up with a pathological example of a device, such that all of its components belong to one or only a few of the chosen primitives. Moreover, this implies that in exceptional cases, class primitives may have to be selected depending on not only the domain but also the device at hand. This compromises the utility of Classes as a device-independent computational model for function.

The diagnosis algorithm assumes the existence of a single non-intermittent fault in the device. However, the single fault assumption is justified in Section 5.2.1. The algorithm also assumes that the device being diagnosed has only one element in it. In order to extend it to devices with multiple elements (such as the automobile carburetor), element-level discrimination techniques must be incorporated into it.

7.2 Perspectives

Classes can be viewed from several perspectives:

1. It is a computational model for function based on component ontology. It is used for efficient representation of function and application of function for discrimination during model based diagnosis. It is systematic. Hence, its scope and competence can be easily evaluated.

2. Representation:

   (a) It has been noted that using device models with multiple levels of abstraction improves the efficiency of diagnosis [Davis, 1989]. Function is an abstraction of behavior. Function model provides a more abstract "causal topology" [deKleer, 1979] of a device than its behavior model. Therefore, Classes enables model based systems to exploit multiple levels of abstraction to improve diagnosis. More to the point, it provides answers to two methodological questions related to using multiple levels of abstraction, viz., the representation question of how to build a model more abstract than the behavior model, and the reasoning question of how to use such an abstract model for diagnosis.

   (b) Model based systems are device and domain-independent. But they are inefficient. Rule based systems are efficient, but device-specific and brittle. In order to get the best of both the diagnostic paradigms, several researchers have attempted to incorporate rules into model based systems [Simmons and Davis, 1987, Fink and Lusth, 1987, David and Krivine, 1989]. Instead, Classes attempts to improve the efficiency of model based diagnosis by *biasing* the models used in it, with function. The models are biased in favor of diagnosing the components functionally responsible for a symptom, first. Biasing the model is a better approach than incorporating rules into a model-based system, because:

i. It does not assume a specific set of failure modes, as do the rule-based segments of systems which incorporate rules. Further, it is not device- or domain-specific.
ii. It is a unified approach, which simultaneously deals with the issues of coverage and efficiency of diagnosis without contributing to the complexity of the system. It is primarily a model-based system, which exploits function to achieve the efficiency of rule-based systems.

3. Reasoning:

   (a) It is a computational model for function based diagnosis, which is the third of the four levels of diagnostic reasoning enumerated by Milne [Milne, 1987, Kumar and Upadhyaya, 1992].

   (b) It orders candidates based on function. Earlier candidate ordering proposals for model based diagnosis have been based on structure [Chen and Srihari, 1989], fault probabilities [deKleer and Williams, 1987] and ease of repair [Fink and Lusth, 1987].

   (c) Function based diagnosis is heuristic. It is based on abduction. Therefore, Classes is an attempt to incorporate heuristics into model based diagnosis.

   (d) Classes helps exploit the 80-20 rule applicable to most devices. The 80-20 rule is an empirical rule which states that 80% of the faults in a device occur in only 20% of its components, and the other 20% of the faults occur in the rest 80% of its components. Ideally, a diagnosis system should be able to exploit this rule: it should be able to diagnose 80% of the faults efficiently, while also being able to diagnose the other 20% correctly. Such a system is more practicable than a standard model based system which is inefficient in every case. Classes enables model based diagnosis systems to exploit the 80-20 rule by using function models.

7.3 Goals

The primary goal of this research was to develop a computational model for function, which could be used for model based diagnosis. The requirements of such a computational model were that it should be systematic, domain-independent, scalable, model based, and enable candidate ordering based on function.

Classes is systematic: classes of components are defined with respect to signal-lines, and vice versa. The dualism of these definitions enables systematic traversal of the structure of a device during both representation (Chapter 4) and reasoning (Chapter 5).

Classes is domain-independent. It is based on the concepts of flow and effort [Paynter, 1961], and can be applied to any domain. This was demonstrated by building the class models of both an electronic printer buffer and an automobile carburetor.

Two discrimination techniques have been proposed in Classes, which order candidates based on function. They are device-independent. Hence, the diagnostic knowledge used by them can be acquired and stored efficiently.

Class representation is adequate for model based reasoning because of composability of its representation, as argued in Section 3.1. The diagnosis algorithm in Classes is also model based, and uses structure, behavior and function models.

Representation in Classes is efficient. Although function of a component is defined with respect to its ports instead of its environment, the representation is proved to ensure correctness of reasoning (Section 3.1).

A complete model of the printer buffer, which has the complexity of a microcomputer, is described in Section 6.2. The defined classes of components and signal-lines were found to be adequate to represent this device. This demonstrates the scalability of Classes.

Classes has been implemented in LISP. A sample run of diagnosing the Display Unit of the printer buffer is listed in Appendix A.

7.4 Contributions

The contributions of this research are as follows:

- Component ontology is proposed as the basis for an efficient representation of function, for use in model based diagnosis. Principles behind component ontological representation are enunciated.

- Function is proposed as the basis of effective candidate discrimination during model based diagnosis. The discrimination is carried out by abduction.

  One advantage of using function instead of the popularly used fault probabilities is that it is easily obtainable for any device, from its design. In contrast, fault probabilities are hard to obtain, and are often unavailable (novel devices) or unknown (classified data).

- Classes is proposed as a computational model for function, based on the above two proposals. It provides function primitives for components as well as subsystems (signal-lines). By defining primitives for components in terms of signal-lines and vice versa, Classes enables a systematic traversal of the device during both representation and reasoning. Therefore, the scope and competence of using function in model based reasoning can be easily evaluated in Classes.

- Component ontological representation in Classes has several merits:

  - Function of a device can be modeled independent of its environment. Hence, it can be compiled into libraries, and is re-usable.
  - The representation is composable, i.e., function model of a complex device can be constructed by composing the function models of its components. Hence, it is adequate for model based reasoning where composability is required of the representation.
  - The process of building the function model of a device from those of its components can be automated. This ensures the fidelity of representation.
  - The representation is linear in complexity, and hence, efficient compared to state and process ontological representations:
    - Space complexity of representation of an atomic component is linear in the number of its ports. Space complexity of representation of a composite device is linear in the number of its components.
    - Function model of a complex device can be composed from those of its components in linear time.
  - Availability of a library of function models of atomic components, and their re-usability and composability to yield function models of complex devices results in a considerable saving of time and effort during device representation.

  - Since Classes is based on the concepts of flow and effort [Paynter, 1961], it is domain-independent. It can be used to represent transdomain devices (E.g., automobile carburetor) and side-effects in devices without any switch in the notation between domains. Therefore, it is scalable to the representation of complex devices.
  - Finally, by fragmenting the structure model of a device into modular causal subsystems called signal-lines, class representation helps focus diagnosis.

- Two function based candidate discrimination techniques are proposed in Classes, viz., teleological and default order techniques. These techniques are as effective as the popularly used minimum entropy technique: they prompt for similar sequences of validating measurements.

  Function based diagnosis using Classes has the following merits:

  - The discrimination techniques, together with the diagnosis algorithm facilitate generation of explanation during diagnosis.
  - The discrimination techniques are device-independent. Therefore, the diagnostic knowledge used by them can be acquired and stored efficiently. The techniques are not brittle unlike rule based systems.
  - Default order technique can be employed to generate partitions of components used in the practicable version of minimum entropy technique [deKleer, 1990], although based on function rather than fault probabilities. Therefore, the practicable version of minimum entropy technique can now be used even when fault probabilities are not known.

- Classes uses hierarchical organization: both representation and diagnosis in Classes account for multiple hierarchical levels. Hence, it is scalable to large devices.

7.5 Adapting and Using Classes

Following are the guidelines for adapting Classes to a domain of the reader's choice:

- **Choosing classes:** Classes of components and signal-lines must be chosen such that they cover all the components/signal-lines in the domain, and are dissimilar enough to be distinguished apart. The classes defined in this dissertation are those that are applicable to most of the domains. For additional function primitives, the reader is referred to Miller's work [Miller, 1978].

- **Defining classes:** The chosen classes must be defined for a component with respect to its ports, and for a port with respect to its component. These definitions are abstractions of behavior, derived as in [Giunchiglia and Walsh, 1992].

- **Using classes:** In order to use the chosen classes for diagnosis:

  - a default order must be derived among them. Heuristic rules which provide the basis for such an order have been proposed in Section 5.1.2.

  - feature-class associations applicable to them must be generated. The most widely applicable features have been provided in Section 5.1.1. Constraint suspension [Davis, 1984] may be used to derive the feature-class associations.

Guidelines for using Classes may be found in the following sections of the dissertation:

- **Representation**

  - Class models of atomic components may be constructed as described in Section 4.1. A list of class primitives have been provided in Section 3.2.

  - Class models of complex devices may be built from those of their components using the algorithms provided in Section 4.1. For additional guidance, the reader is referred to similar work on behavior modeling, reported in [Kumar and Upadhyaya, 1990].

  - A complete class model of an electronic printer buffer and an automobile carburetor are provided in Chapter 6. It is hoped that they will aid the reader in better understanding and constructing class models.

- **Diagnosis**

  - Diagnosis in Classes may be carried out according to the diagnosis algorithm provided in Section 5.2. This algorithm uses candidate ordering techniques described in Sections 5.1.1 and 5.1.2.

- Implementation details are reported in Section 6.1. A sample session of our implemented system diagnosing the printer buffer is discussed in Section 6.3.2. It is hoped that these sections will aid the reader in better understanding and implementing class based diagnosis.
- Classes may be integrated into an existing model based diagnosis system according to the guidelines provided in Section 5.1.3.

7.6 Future Work

Classes has been proposed to illustrate component ontological representation of function, and not to promote any particular choice of class primitives. Although, the classes of components and signal-lines proposed in the dissertation in Section 3.2 are widely applicable, additional classes may be found necessary and/or useful in specific domains, as discussed in Section 6.5. It is interesting to investigate the relation between class primitives useful in a domain and the domain principles: whether and how the domain principles can influence the choice of classes for a domain. A related problem worth investigating is: how to define metrics to measure the coverage and resolution of representation of a given set of class primitives.

The class model of an atomic component can be constructed from an abstraction of its behavior model, as described in Section 4.1. Abstractions of behavior expressions can be systematically generated by several techniques [Giunchiglia and Walsh, 1992]. A problem for future work is to automate the construction of class models of components from their behavior specifications. This can improve the efficiency and better ensure the fidelity of class representation.

The function of a component may change in an environment, depending on the signals at its ports. In Classes, such changes have been accounted for by assigning the most specialized function to the component's model during representation. It has been noted in Section 3.1 that this entails a loss of efficiency during reasoning. It is worthwhile to investigate whether this approach can be improved upon, without significant overheads in computation or storage space.

Classes can be used to represent transdomain devices such as an automobile carburetor. However, the diagnosis algorithm developed in Section 5.2 assumes that the device being diagnosed has only one element in it. Element-level discrimination techniques must be incorporated into the algorithm in order to extend it to diagnosing transdomain devices. For instance, if a car does not start, the algorithm should be able to decide which signal-line to examine first: fuel-air mixture or electric sparks. A problem for future work is to devise function based techniques to discriminate among elements.

In teleological technique, outputs and symptoms are expressed in terms of features. Note that functions, i.e., classes may themselves be expressed in terms of conjunctive predicates. It is advantageous to devise a set of terms which can be used to express symptoms as well as functions. When features, functions and symptoms all share the same vocabulary of terms, feature-class associations need not be compiled and stored any more. Instead, during diagnosis, symptoms can be translated to missing features, and these features can in turn be translated to classes. Devising such a set of terms, primitive and canonical enough to serve as descriptive building blocks of both functions and symptoms, is an interesting extension of the work in this dissertation.

Appendix A

Sample Run

```
Script started on Mon Aug 24 19:54:13 1992
> lisp
KCl (Kyoto Common Lisp)  September 16, 1986

>(load "run")
Loading run
Loading fun.lsp
Finished loading fun.lsp
Loading descr.lsp
Finished loading descr.lsp
Loading syslib.lsp
Finished loading syslib.lsp
Loading libe.lsp
Finished loading libe.lsp
Loading simulate.lsp
Finished loading simulate.lsp
Loading cs.lsp
Finished loading cs.lsp
Finished loading run
T

>(classes)
What is the device you wish to be diagnosed?
pbb-led-1
```

```
Loading ../vmes2/Library/PBB-LED-1.lsp
Finished loading ../vmes2/Library/PBB-LED-1.lsp
Loading ../vmes2/Library/PBB-LED.lib
Finished loading ../vmes2/Library/PBB-LED.lib
What is the MEASURED value of the port OUT1 of the device PBB-LED-1 ?
127
What is the MEASURED value of the port OUT2 of the device PBB-LED-1 ?
0
What is the MEASURED value of the port OUT3 of the device PBB-LED-1 ?
0
What is the MEASURED value of the port OUT4 of the device PBB-LED-1 ?
79
What is the MEASURED value of the port OUT5 of the device PBB-LED-1 ?
0
What is the MEASURED value of the port OUT6 of the device PBB-LED-1 ?
79
What is the MEASURED value of the port OUT7 of the device PBB-LED-1 ?
0
What is the MEASURED value of the port IN1 of the device PBB-LED-1 ?
5
What is the MEASURED value of the port IN2 of the device PBB-LED-1 ?
1
What is the MEASURED value of the port IN3 of the device PBB-LED-1 ?
0
What is the MEASURED value of the port OUT1 of the device CPU-1 ?
158
Loading ../vmes2/Library/CPU-1.lsp
Finished loading ../vmes2/Library/CPU-1.lsp
Loading ../vmes2/Library/CPU.lib
Finished loading ../vmes2/Library/CPU.lib
Loading ../vmes2/Library/DECODER-1.lsp
Finished loading ../vmes2/Library/DECODER-1.lsp
Loading ../vmes2/Library/DECODER.lib
Finished loading ../vmes2/Library/DECODER.lib
Loading ../vmes2/Library/BUFFER-1.lsp
```

```
Finished loading ../vmes2/Library/BUFFER-1.lsp
Loading ../vmes2/Library/BUFFER.lib
Finished loading ../vmes2/Library/BUFFER.lib
Loading ../vmes2/Library/BUFFER-2.lsp
Finished loading ../vmes2/Library/BUFFER-2.lsp
Loading ../vmes2/Library/BUFFER-3.lsp
Finished loading ../vmes2/Library/BUFFER-3.lsp
Loading ../vmes2/Library/RESISTOR-PACK-1.lsp
Finished loading ../vmes2/Library/RESISTOR-PACK-1.lsp
Loading ../vmes2/Library/RESISTOR-PACK.lib
Finished loading ../vmes2/Library/RESISTOR-PACK.lib
Loading ../vmes2/Library/RESISTOR-PACK-2.lsp
Finished loading ../vmes2/Library/RESISTOR-PACK-2.lsp
Loading ../vmes2/Library/RESISTOR-PACK-3.lsp
Finished loading ../vmes2/Library/RESISTOR-PACK-3.lsp
Loading ../vmes2/Library/LED-1.lsp
Finished loading ../vmes2/Library/LED-1.lsp
Loading ../vmes2/Library/LED.lib
Finished loading ../vmes2/Library/LED.lib
Loading ../vmes2/Library/SWAP-DIODE-1.lsp
Finished loading ../vmes2/Library/SWAP-DIODE-1.lsp
Loading ../vmes2/Library/SWAP-DIODE.lib
Finished loading ../vmes2/Library/SWAP-DIODE.lib
Loading ../vmes2/Library/LED-2.lsp
Finished loading ../vmes2/Library/LED-2.lsp
Loading ../vmes2/Library/LED-3.lsp
Finished loading ../vmes2/Library/LED-3.lsp

Checking the value on the WR signal
What is the MEASURED value of the port OUT2 of the device CPU-1 ?
0

Checking the value on the address bus A0-A2
What is the MEASURED value of the port OUT3 of the device CPU-1 ?
0
```

```
Checking the value on the address bit A7
What is the MEASURED value of the port OUT4 of the device CPU-1 ?
0

Checking the value on the I/O signal
What is the MEASURED value of the port OUT5 of the device CPU-1 ?
0

****The default order of signals associated with:********

Ports : (OUT1)  of Device  PBB-LED-1  are :
---------------------------------------------------------

(((SL1 DATA (PBB-LED-1 OUT1) (LED-1 OUT1) (LED-1 IN1)
       (RESISTOR-PACK-1 OUT1) (RESISTOR-PACK-1 IN1) (BUFFER-1 OUT1)
       (BUFFER-1 IN1) (CPU-1 OUT1)))
 NIL NIL NIL NIL)
*********************************************************
****The default component order in the signal-line:****

(SL1 DATA (LED-1 OUT1) (LED-1 IN1) (RESISTOR-PACK-1 OUT1)
     (RESISTOR-PACK-1 IN1) (BUFFER-1 OUT1) (BUFFER-1 IN1) (CPU-1 OUT1))
---------------------------is---------------------------

((LED-1 CPU-1) NIL NIL NIL (BUFFER-1) (RESISTOR-PACK-1))
*******************************************************

Currently considering component:  LED-1
What is the MEASURED value of the port OUT1 of the device LED-1 ?
127
What is the MEASURED value of the port OUT2 of the device LED-1 ?
0
What is the MEASURED value of the port IN1 of the device LED-1 ?
127
```

```
What is the MEASURED value of the port IN2 of the device LED-1 ?
0

******************************************
LED-1 is not faulty.
******************************************

****The default order of signals associated with:********

Ports : (OUT1 OUT2)  of Device  LED-1  are :
---------------------------------------------------------

(NIL NIL NIL NIL NIL)
*********************************************************

Currently considering component:  CPU-1
What is the MEASURED value of the port IN1 of the device CPU-1 ?
5
What is the MEASURED value of the port IN2 of the device CPU-1 ?
1

******************************************
CPU-1 is not faulty.
******************************************

****The default order of signals associated with:********

Ports : (OUT1 OUT2)  of Device  CPU-1  are :
---------------------------------------------------------

(NIL NIL NIL ((SL16 CLOCK (CPU-1 IN2) (PBB-LED-1 IN2)))
     ((SL15 POWER (CPU-1 IN1) (PBB-LED-1 IN1))))
*********************************************************
****The default component order in the signal-line:****
```

```
(SL16 CLOCK)
---------------------------is---------------------------

(NIL NIL NIL NIL NIL NIL)
********************************************************

******************************************

The  IN2  port of superdevice  PBB-LED-1

may be getting faulty input

******************************************
****The default component order in the signal-line:****

(SL15 POWER)
---------------------------is---------------------------

(NIL NIL NIL NIL NIL NIL)
********************************************************

The superdevice-port  IN1  does not seem to be erroneous

Currently considering component:  BUFFER-1
What is the MEASURED value of the port OUT1 of the device BUFFER-1 ?
127
What is the MEASURED value of the port OUT2 of the device BUFFER-1 ?
0
What is the MEASURED value of the port IN1 of the device BUFFER-1 ?
158
What is the MEASURED value of the port IN2 of the device BUFFER-1 ?
1
What is the MEASURED value of the port IN3 of the device BUFFER-1 ?
0
What is the MEASURED value of the port IN4 of the device BUFFER-1 ?
```

```
5

******************************************
BUFFER-1 is not faulty.
******************************************

****The default order of signals associated with:********

Ports : (OUT1 OUT2)  of Device  BUFFER-1  are :
---------------------------------------------------------

(NIL ((SL12 CONTROL (BUFFER-1 IN3) (CPU-1 OUT2)))
     ((SL7 ADDRESS (BUFFER-1 IN2) (DECODER-1 OUT1) (DECODER-1 IN1)
           (CPU-1 OUT3)))
     NIL ((SL18 POWER (BUFFER-1 IN4) (PBB-LED-1 IN1))))
*********************************************************
****The default component order in the signal-line:****

(SL12 CONTROL (CPU-1 OUT2))
---------------------------is---------------------------

((CPU-1) NIL NIL NIL NIL NIL)
*******************************************************
****The default component order in the signal-line:****

(SL7 ADDRESS (DECODER-1 OUT1) (DECODER-1 IN1) (CPU-1 OUT3))
---------------------------is---------------------------

((CPU-1) (DECODER-1) NIL NIL NIL NIL)
*******************************************************

Currently considering component:  DECODER-1

****The default order of signals associated with:********
```

```
Ports : (OUT1 OUT4 OUT5 OUT6 OUT7 OUT8)  of Device  DECODER-1  are :
----------------------------------------------------------


(NIL NIL NIL NIL NIL)
**********************************************************
What is the MEASURED value of the port OUT1 of the device DECODER-1 ?
1
What is the MEASURED value of the port OUT2 of the device DECODER-1 ?
1
What is the MEASURED value of the port OUT3 of the device DECODER-1 ?
1
What is the MEASURED value of the port OUT4 of the device DECODER-1 ?
1
What is the MEASURED value of the port OUT5 of the device DECODER-1 ?
1
What is the MEASURED value of the port OUT6 of the device DECODER-1 ?
1
What is the MEASURED value of the port OUT7 of the device DECODER-1 ?
1
What is the MEASURED value of the port OUT8 of the device DECODER-1 ?
1
What is the MEASURED value of the port IN1 of the device DECODER-1 ?
0
What is the MEASURED value of the port IN2 of the device DECODER-1 ?
0
What is the MEASURED value of the port IN3 of the device DECODER-1 ?
0
What is the MEASURED value of the port IN4 of the device DECODER-1 ?
0
What is the MEASURED value of the port IN5 of the device DECODER-1 ?
5

*******************************************
DECODER-1 is faulty.
*******************************************
```

```
(DECODER-1)

>(bye)
Bye.
> ^DΦΦexit

script done on Mon Aug 24 19:57:34 1992
```

Bibliography

[Abu-Hanna *et al.*, 1991] A. Abu-Hanna, R. Benjamins, and W. Jansweijer. Device understanding and modeling for diagnosis. *IEEE Expert*, 6(2):26–32, 1991.

[Abu-Hanna *et al.*, 1994] A. Abu-Hanna, W. Jansweijer, R. Benjamins, and B. Wielinga. Functional models in perspective: Their characteristics and integration in multiple model based diagnosis. *International Journal of Applied Artificial Intelligence*, 8:219–237, 1994.

[Allemang and Chandrasekaran, 1991] D. Allemang and B. Chandrasekaran. Functional representation and program debugging. In *Proceedings of the Sixth Knowledge Based Software Engineering Conference*, pages 136–143, 1991.

[Bakker *et al.*, 1992] R.R. Bakker, M. Bourseau, and M.K. de Weger. Pragmatic reasoning in model-based diagnosis. In *Working Notes of The Third International Workshop on Principles of Diagnosis*, pages 107–116, October 1992.

[Bandekar, 1991] V.R. Bandekar. Reasoning with multiple models. In *Working Notes of AAAI-91 Workshop on Model-Based Reasoning*, pages 29–34, July 1991.

[Ben-Bassat *et al.*, 1988] M. Ben-Bassat, D. Ben-Arie, I. Beniaminy, J. Cheifetz, and M. Klinger. AI-Test: A real life expert system for electronic troubleshooting (A description and a case study). In *Proceedings of Fourth IEEE Conference on AI Applications*, pages 2–10, March 1988.

[Bobrow, 1984] D.G. Bobrow. Qualitative reasoning about physical systems - An introduction. *Artificial Intelligence*, 24(1-3):1–5, 1984.

[Bradshaw and Young, 1991] J.A. Bradshaw and R.M. Young. Evaluating design knowledge using knowledge of purpose and knowledge of structure. *IEEE Expert*, 6(2):33–40, 1991.

[Brajnik *et al.*, 1989] G. Brajnik, L. Chittaro, G. Guida, C. Tasso, and E. Toppano. The use of many diverse models of an artifact in the design of cognitive aids. In *Proceedings of Second European Meeting on Cognitive Approaches to Process Control*, pages 69–79, 1989.

[Brajnik *et al.*, 1990] G. Brajnik, L. Chittaro, C. Tasso, and E. Toppano. Epistemology, organization and use of functional knowledge for reasoning about physical systems. In *Working Notes of Tenth International Workshop on Expert Systems and Their Applications, held in conjunction with Second Generation Expert Systems Conference, Avignon, France*, pages 53–66, May 1990.

[Brajnik *et al.*, 1991] G. Brajnik, L. Chittaro, G. Guida, C. Tasso, and E. Toppano. Representation and use of teleological knowledge in the multi-modeling approach. In *Working Notes of AAAI-91 Workshop on Model-Based Reasoning*, pages 64–73, July 1991.

[Brand *et al.*, 1993] M. Brand, L. Birnbaum, and P. Cooper. Sensible scenes: Visual understanding of complex structures through causal analysis. In *Proceedings of National Conference on Artificial Intelligence (AAAI 93)*, pages 588–593, July 1993.

[Brown *et al.*, 1982] J.S. Brown, R. Burton, and J. deKleer. Pedagogical, natural language, and knowledge engineering issues in SOPHIE I, II and III. In D. Sleeman and J.S. Brown, editors, *Intelligent Tutoring Systems*, pages 227–282. Academic Press, 1982.

[Brown, 1976] A. Brown. Qualitative knowledge, causal reasoning, and the localization of failures. Technical Report 362, MIT AI Lab, 1976.

[Buchanan and Shortliffe, 1984] B.G. Buchanan and E.F. Shortliffe, editors. *Rule-based Expert Systems*. Addison-Wesley, Reading, Mass., 1984.

[Bylander, 1988] T. Bylander. A critique of qualitative simulation from a Consolidation viewpoint. *IEEE Transactions on Systems, Man and Cybernetics*, 18(2):252–263, 1988.

[Cantone *et al.*, 1983] R.R. Cantone, F. Pipitone, W. Lander, and M. Marrone. Model-based probabilistic reasoning for electronics troubleshooting. In *Proceed-*

ings of Eighth International Conference on Artificial Intelligence, pages 207–211, August 1983.

[Chandrasekaran and Milne, 1985] B. Chandrasekaran and R. Milne. Reasoning about structure, behavior and function. *SIGART Newsletter*, 93:4–59, 1985.

[Chen and Srihari, 1989] J.S. Chen and S. N. Srihari. Candidate ordering and elimination in model-based fault diagnosis. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 89)*, pages 1363–1368, August 1989.

[Chittaro *et al.*, 1989] L. Chittaro, C. Constantini, G. Guida, C. Tasso, and E. Toppano. Diagnosis based on cooperation of multiple knowledge sources. In *Working Notes of AVIGNON '89: Ninth International Workshop on Expert Systems and Their Application: Specialized Conference on Second Generation Expert Systems*, pages 19–33, 1989.

[Console and Torasso, 1991] L. Console and P. Torasso. A spectrum of logical definitions of model based diagnosis. *Computational Intelligence*, 7(3):133–141, 1991.

[Dague *et al.*, 1987] P. Dague, O. Raiman, and P. Deves. Troubleshooting: When modeling is the trouble. In *Proceedings of Sixth National Conference on Artificial Intelligence*, pages 600–605, August 1987.

[David and Krivine, 1989] Jean-Marc David and Jean-Paul Krivine. Augmenting experience-based diagnosis with causal reasoning. In Horn, editor, *Causal AI Models*. Hemisphere Publishing Corporation, 1989.

[Davis and Hamscher, 1988] R. Davis and W. Hamscher. Troubleshooting. In H. Shrobe, editor, *Exploring Artificial Intelligence*. Morgan Kaufman, 1988.

[Davis, 1984] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.

[Davis, 1989] R. Davis. Form and content in model based reasoning. In *Working Notes of IJCAI Workshop on Model Based Reasoning*, pages 11–27, August 1989.

[deKleer and Brown, 1984] J. deKleer and J.S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24(1-3):7–83, 1984.

[deKleer and Williams, 1987] J. deKleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[deKleer and Williams, 1989] J. deKleer and B.C. Williams. Diagnosis with behavioral modes. In *Proceedings of Eleventh International Joint Conference on Artificial Intelligence*, pages 1324–1330, 1989.

[deKleer, 1976] J. deKleer. Local methods for localizing faults in electronic circuits. AI Memo 394, MIT, November 1976.

[deKleer, 1979] J. deKleer. The origin and resolution of ambiguities in causal arguments. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 197–203, August 1979.

[deKleer, 1984] J. deKleer. AI approaches to troubleshooting. In *Artificial Intelligence in Maintenance: Proceedings of the Joint Services Workshop*, pages 83–94, June 1984.

[deKleer, 1990] J. deKleer. Using crude probability estimates to guide diagnosis. *Artificial Intelligence*, 45:381–392, 1990.

[Farquhar, 1989] A. Farquhar. Focusing ATMS-based diagnosis and prediction. In *Working Notes of AAAI Workshop on Model Based Reasoning*, pages 41–45, August 1989.

[Fink and Lusth, 1987] P.K. Fink and J.C. Lusth. Expert systems and diagnostic expertise in the mechanical and electrical domains. *IEEE Transactions on Systems Man and Cybernetics*, 17(3):340–349, 1987.

[Fink, 1985] P.K. Fink. Control and integration of diverse knowledge in an expert system. In *Proceedings of Ninth International Joint Conference on Artificial Intelligence*, pages 426–431, August 1985.

[First *et al.*, 1982] M.B. First, B.J. Weimer, S. McLinden, and R.A. Miller. Localize: Computer-assisted localization of peripheral nervous system lesions. *Computers and Biomedical Research*, 15(6):525–543, 1982.

[Forbus, 1984] K.D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[Franke, 1991] D.W. Franke. Deriving and using descriptions of purpose. *IEEE Expert*, 6(2):41–47, 1991.

[Gallanti *et al.*, 1989] M. Gallanti, M. Roncato, A. Stefanini, and G. Tornielli. A diagnostic algorithm based on models at different level of abstraction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1350–1355, August 1989.

[Genesereth, 1984] M. R. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436, 1984.

[Ginsberg, 1986] M. Ginsberg. Counterfactuals. *Artificial Intelligence*, 30(1):35–80, 1986.

[Giunchiglia and Walsh, 1992] F. Giunchiglia and T. Walsh. Theories of abstraction: A historical perspective. In *Working Notes of AAAI-92 Workshop on Approximation and Abstraction of Computational Theories*, pages 85–89, July 1992.

[Goel and Chandrasekaran, 1989] A. Goel and B. Chandrasekaran. Functional representation of designs and redesign problem solving. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 89)*, pages 1388–1394, August 1989.

[Hammond, 1989] K. Hammond, editor. *Working Notes of the Case-Based Reasoning Workshop*, San Mateo, CA, 1989. Morgan Kaufmann, Inc.

[Hamscher and Struss, 1990] W. Hamscher and P. Struss. *The AAAI-90 Tutorial on Model-Based Diagnosis*, 1990.

[Hamscher, 1988] W. C. Hamscher. Model-based troubleshooting of digital systems. Technical Report 1074, MIT Artificial Intelligence Laboratory, 1988.

[Hamscher, 1990] W.C. Hamscher. Modeling digital circuits for troubleshooting: An overview. In *Proceedings of Sixth IEEE Conference on AI Applications*, pages 2–8, March 1990.

[Hawkins *et al.*, 1994] R. Hawkins, J. K. McDowell, J. Sticklen, T. Hill, and R. Boyer. Function-based modeling and troubleshooting. *International Journal of Applied Artificial Intelligence*, 8:285–302, 1994.

[Hodges, 1992] J. Hodges. Naive mechanics: A computational model of device function and use in design improvisation. *IEEE Expert, Special Track on Functional Reasoning*, 7(1):14–27, 1992.

[Hunt and Price, 1992] J. Hunt and C. Price. Multiple-model diagnosis of electromechanical subsystems. *Journal of Systems Engineering*, 1992.

[Hunt *et al.*, 1995] J. E. Hunt, D. R. Pugh, and C. J. Price. Failure mode effects analysis: A practical application of functional modelling. *International Journal of Applied Artificial Intelligence*, 1995. to appear in January.

[Iwasaki *et al.*, 1995] Y. Iwasaki, M. Vescovi, R. Fikes, and B. Chandrasekaran. A causal functional representation language with behavior-based semantics. *International Journal of Applied Artificial Intelligence*, 1995. to appear in January.

[Josephson and Josephson, 1993] J. Josephson and S. Josephson. *Abductive Inference: Computation, Philosophy, Technology.* Cambridge University Press, 1993.

[Kalme, 1986] C. Kalme. AI tool applies the art of building expert systems to troubleshooting boards. *Electronic Design*, 4:159–166, April 3 1986.

[Kannapan, 1995] S. M. Kannapan. Functional metrics for engineered devices. *International Journal of Applied Artificial Intelligence*, 1995. to appear in January.

[Keller, 1989] Keller. Compiling special-purpose rules from general-purpose device models. KSL Report KSL-89-49, Stanford, June 1989.

[Keuneke, 1991] A.M. Keuneke. Device representation: The significance of functional knowledge. *IEEE Expert*, 6(2):22–25, 1991.

[Kuipers, 1986] B.J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29(3):289–338, 1986.

[Kumar and Upadhyaya, 1990] A. N. Kumar and S. J. Upadhyaya. Automating representation. In *Working Notes of the AAAI-90 Workshop on Model Based Reasoning*, pages 124–130, 1990.

[Kumar and Upadhyaya, 1992] A. N. Kumar and S. J. Upadhyaya. Abstraction and approximation in diagnostic reasoning. In *Working Notes of the Workshop on Approximation and Abstraction of Computational Theories*, pages 124–131, July 1992.

[Kumar and Upadhyaya, 1995] A.N. Kumar and S.J. Upadhyaya. Function based discrimination during model based diagnosis. *International Journal of Applied Artificial Intelligence*, 1995. to appear in January.

[Kumar, 1994a] A.N. Kumar. Function based reasoning. *The Knowledge Engineering Review*, 1994. to appear in September.

[Kumar, 1994b] A.N. Kumar. Function based reasoning: An introduction. *International Journal of Applied Artificial Intelligence*, 8:167–172, 1994.

[Larsson, 1993] J.E. Larsson. Reasoning with explicit means-end models. In *Working Notes of the AAAI-93 Workshop on Reasoning About Function*, pages 73–77, July 1993.

[Lind *et al.*, 1992] M. Lind, M.N. Larsen, and A. Osman. Applications of Multilevel Flow Modeling. In *Proceedings of ANP International Conference on Design and Safety of Advanced Nuclear Power Plants*, 1992.

[Lind, 1994] M. Lind. Modeling goals and functions of complex industrial plants. *International Journal of Applied Artificial Intelligence*, 8:259–283, 1994.

[Liu and Farley, 1990] Z. Liu and A.M. Farley. Shifting ontological perspectives in reasoning about physical systems. In *Proceedings of Ninth National Conference on Artificial Intelligence*, pages 395–400, July 1990.

[Lusth, 1985] J.C. Lusth. Diagnosis using a functional model. In *Proceedings of International Conference on Systems, Man and Cybernetics*, 1985.

[Marr, 1982] D. Marr. *Vision.* W.H. Freeman, San Francisco, 1982.

[Miller, 1978] J.G. Miller. *Living Systems.* McGraw-Hill Book Company, 1978.

[Milne, 1985] R. Milne. Diagnosing faults through responsibility. In *Proceedings of the Annual Conference of the ACM*, pages 88–91, October 1985.

[Milne, 1987] R. Milne. Strategies for diagnosis. *IEEE Transactions on Systems, Man and Cybernetics*, 17:333–339, 1987.

[Minsky, 1981] M. Minsky. A framework for representing knowledge. In R.J. Brachman and H.J. Levesque, editors, *Readings in Knowledge Representation.* Morgan Kaufmann, San Mateo, California, 1981.

[Pan, 1984] J. Pan. Qualitative reasoning with deep-level mechanism models for diagnosis of mechanism failures. In *Proceedings of First Conference on AI Applications*, pages 295–301, 1984.

[Patil, 1981] R.S. Patil. Causal representation of patient illness for electrolyte and acid-base diagnosis. Technical Report 267, MIT Lab for Computer Science, October 1981.

[Paynter, 1961] H.M. Paynter. *Analysis and Design of Engineering Systems.* MIT Press, Cambridge, 1961.

[Pittges *et al.*, 1993] J. Pittges, K. Eiselt, A.K. Goel, A.G.S. Garza, K. Mahesh, and J. Peterson. Representation and use of function in natural language understanding. In *Working Notes of the AAAI-93 Workshop on Reasoning About Function*, pages 114–120, July 1993.

[Poole, 1989] D. Poole. Normality and faults in logic-based diagnosis. In *Proceedings of Eleventh International Joint Conference on Artificial Intelligence*, pages 1304–1310, 1989.

[Poole, 1992] D. Poole. Efficient model-based diagnosis: Searching in Bayesian networks. In *Working Notes of The Third International Workshop on Principles of Diagnosis*, pages 159–168, October 1992.

[Pople, 1977] H. Pople. The formation of composite hypotheses in diagnostic problem solving: An exercise in synthetic reasoning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 77)*, pages 1030–1037, August 1977.

[Reggia *et al.*, 1985] J.A. Reggia, D.S. Nau, and P.Y. Wang. A formal model of diagnostic inference II: Algorithmic solution and application. *Information Sciences*, 37:257–285, 1985.

[Reiter, 1987] R. Reiter. A theory of diagnosing from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[Rich and Knight, 1991] E. Rich and K. Knight. *Artificial Intelligence.* McGraw Hill Inc., 1991.

[Rodenacker, 1971] W. Rodenacker. *Methodisches Konstruieren.* Springer-Verlag, Berlin, 1971.

[Rosenman and Gero, 1995] M. A. Rosenman and J. S. Gero. The what, the how and the why in design. *International Journal of Applied Artificial Intelligence*, 1995. to appear in January.

[Scarl *et al.*, 1985] E. Scarl, J.R. Jamieson, and C.I. Delaune. A fault detection and isolation method applied to liquid oxygen loading for the space shuttle. In *Proceedings of Ninth International Joint Conference on Artificial Intelligence*, pages 414–416, 1985.

[Sembugamoorthy and Chandrasekaran, 1986] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner and C. Riesbeck, editors, *Experience, Memory and Reasoning*. Erlbaum, 1986.

[Simmons and Davis, 1987] R. Simmons and R. Davis. Generate, test and debug: Combining associational rules and causal models. In *Proceedings of Tenth International Joint Conference on Artificial Intelligence*, pages 1071–1078, August 1987.

[Steels, 1989] L. Steels. Diagnosis with a function-fault model. In Horn, editor, *Causal AI Models*. Hemisphere Publishing Corporation, 1989.

[Stephanopoulos, 1983] G. Stephanopoulos. Synthesis of control systems for chemical plants: A challenge for creativity. *Computers and Chemical Engineering*, 7(4):331–365, 1983.

[Sticklen and Chandrasekaran, 1989] J. Sticklen and B. Chandrasekaran. Integrating classification-based compiled level reasoning with function-based deep level reasoning. In Horn, editor, *Causal AI Models*. Hemisphere Publishing Corporation, 1989.

[Sticklen *et al.*, 1993] J. Sticklen, J.K. McDowell, R. Hawkins, T. Hill, and R. Boyer. Troubleshooting based on a functional device representation: Diagnosing faults in the external active thermal control system of space station FREEDOM. In U.M. Fayyad and R. Uthurusamy, editors, *SPIE Proceedings Series Volume 1993*. The International Society for Optical Engineering, 1993.

[Struss and Dressler, 1989] P. Struss and O. Dressler. Physical negation: Integrating fault models into the general diagnostic engine. In *Proceedings of Eleventh International Joint Conference on Artificial Intelligence*, pages 1318–1323, August 1989.

[Struss, 1988] P. Struss. Extensions to ATMS-based diagnosis. In J.S. Gero, editor, *Artificial Intelligence in Engineering: Diagnosis and Learning*. Elsevier and Computational Mechanics Publications, 1988.

[Sutton *et al.*, 1993] M. Sutton, L. Stark, and K.W. Bowyer. Function-based generic recognition for multiple object categories. In A.K. Jain and P.J. Flynn, editors, *Three-dimensional Object Recognition Systems*. Elsevier Science Publishers, 1993.

[Tomiyama and Umeda, 1993] T. Tomiyama and Y. Umeda. A CAD for functional design. In *Working Notes of the AAAI-93 Workshop on Reasoning About Function*, pages 172–179, July 1993.

[Umeda *et al.*, 1992] Y. Umeda, T. Tomiyama, and H. Yoshikawa. A design methodology for a self-maintenance machine based on functional redundancy. In D.L. Taylor and L.A. Stauffer, editors, *Design Theory and Methodology - DTM '92*. ASME, 1992.

[Vescovi *et al.*, 1993] M. Vescovi, Y. Iwasaki, R. Fikes, and B. Chandrasekaran. Cfrl: A language for specifying the causal functionality of engineered devices. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 626–634, July 1993.

[Weld, 1990] D.S. Weld. Approximate reformulations. In *Proceedings of Eighth National Conference on Artificial Intelligence*, pages 407–416, July 1990.

[Wirth and O'Rorke, 1993] R. Wirth and P O'Rorke. Representing and reasoning about functions for failure modes and effects. In *Working Notes of the AAAI-93 Workshop on Reasoning About Function*, pages 188–194, July 1993.