Knowledge Based Expert Systems for Engineering: Classification, Education and Control

Editors:
D. Sriram
R.A. Adey

Computational Mechanics Publications

D. SRIRAM

Department of Civil Engineering
M.I.T.
Cambridge, MA 02139
U.S.A.

R.A. ADEY

Computational Mechanics Institute
Ashurst Lodge
Ashurst
Southampton
SO4 2AA
U.K.

Associate Editors. J. Gero, M. Tenenbaum and R. Milne

Sub-editors: C.M. Mellors and J. Knudsen

British Library Cataloguing in Publication Data

Knowledge based expert systems for engineering:
classification, education and control
1. Computer-aided design 2. Expert systems
(Computer-science)
I. Sriram, D. II. Adey, R.A.
620'.00425'0285633 TA174

ISBN 0-905451-92-9

ISBN 0-905451-92-9 Computational Mechanics Publications, Southampton
ISBN 0-931215-81-1 Computational Mechanics Publications, Boston, USA

Library of Congress Catalog Card Number 87-71289

Printed in Great Britain by Unwin Brothers Ltd., Old Woking, Surrey

Device Representation and Graphics Interfaces of VMES

J. Geller, M.R. Taie, S.C. Shapiro, S.N. Srihari
Department of Computer Science, State University of New York at Buffalo, Buffalo, NY 14260, U.S.A.

ABSTRACT

The VMES project aims to create a device-model-based versatile maintenance expert system which assists the user in isolating specific faulty components or connections in a malfunctioning digital circuit. We describe a device representation formalism that supports the diagnostic reasoning of VMES and eases its adaptation to new devices. The salient feature of this scheme is the inclusion of both logical and physical structural descriptions of the target device. The two representations enable VMES to make efficient diagnostic judgements and to interact effectively with the user in performing repair and test. The user interface of VMES is treated as a separate area of scientific investigation. We describe the design and implementation of three interfaces, *viz.*, a graphics display interface, a graphics input interface, and a natural language input interface. The use of knowledge based interface technology has proven a rewarding area of research from the theoretical as well as the applied perspective.

INTRODUCTION

VMES is a device-model-based versatile maintenance expert system for the domain of digital circuits [7]. The objective of VMES is to interact with a maintenance technician (the user) to identify the specific faulty component or connection of a malfunctioning circuit. The versatility of VMES is multifold: across a wide range of devices, covering most possible faults, suitable for different maintenance levels, and providing an intelligent user interface. VMES uses a device-model-based approach since it is more general than the traditional empirical-rule-based approach [1, 2, 7].

VMES consists of five modules: the knowledge-base; the inference engine; the active database; the end-user interface; and the intermediate-user interface (Fig. 1). The knowledge-base is implemented as an expandable component library which contains component descriptions. The inference engine has the generic diagnosis knowledge of the domain, and uses the SNIP semantic network inference package of SNePS, the semantic network processing system, as its basis [5, 3]. An active database is created and updated throughout each diagnostic session to keep the instantiated objects and their associated diagnostic states and values. The end-user interface interfaces the maintenance technicians when carrying out a diagnostic session. The intermediate-user interface interfaces the engineers or senior technicians to update the knowledge-base for new devices. All these five modules are implemented on top of SNePS.

As knowledge engineering is to empirical-rule-based systems, device modeling/representation is the key to the success of a device-model-based fault diagnosis system, since knowledge about the structure and function of a device is the major knowledge source of reasoning in such a system. Consequently, our efforts are focused on the development of a device representation formalism for versatile maintenance. All knowledge, whether of structural, functional, or graphical, is in a unified knowledge base (Fig. 2), which is easily expandable and is referred as a "component library".

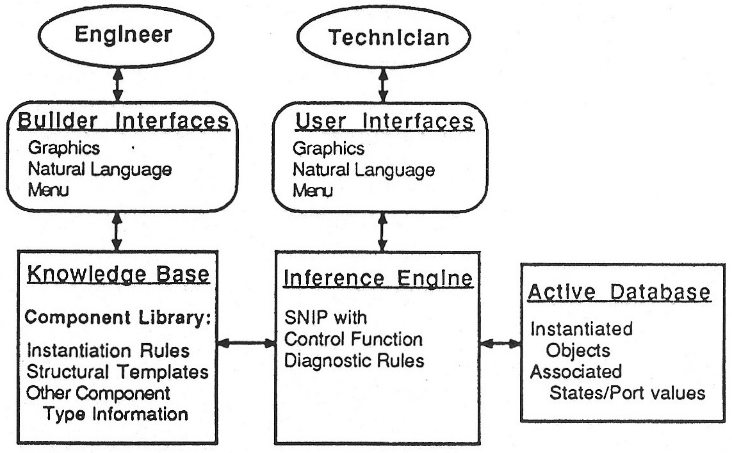

Figure 1 Architecture of VMES.

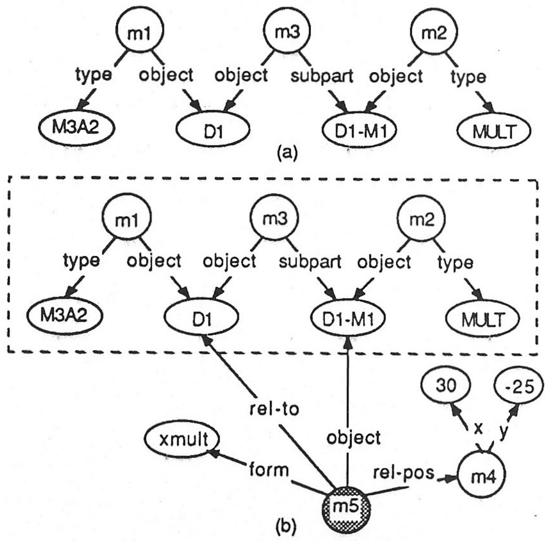

Figure 2 A unified knowledge representation using SNePS. (a) The SNePS network representing "D1 is an M3A2, D1-M1 is a MULTiplier, and D1-M1 is a subpart of D1". (b) Adding graphical knowledge without changing the original representation in (a).

User interaction is an important issue of the VMES project in two aspects: VMES has to communicate with the maintenance technician for test and repair, and it has to provide an engineer or a senior technician facilities for adapting it to other devices by adding their descriptions to the component library. In the next section, the VMES device representation scheme is described with the emphasis on device representation which facilitates a better user interaction. Section 3 discusses the knowledge-based graphics package of VMES, which maintains and reasons on "graphical deep knowledge" when interacting with the user. Section 4 is the conclusion.

DEVICE REPRESENTATION

In VMES, a device is modeled as hierarchically arranged modules. While this is hardly a new idea, the innovative part of our work includes: the use of an expandable component library; a clear distinction between two levels of abstraction of an object; representing a component "type" as an instantiation rule and a structural template; an explicit representation of wires and points of contact (POCONs); and the incorporation of logical and physical structure of devices for both diagnostic reasoning and user interaction.

General Representation Scheme

Devices in the digital circuit domain share many common component types. Representing every detail of a device causes much representation overhead, which in turn leads to system inefficiency. Instead of coding each device, we only describe the component types used by the device to the component library. Parts of a device are instantiated as needed. Adapting VMES to a new device is an easy task — just adding to the component library those component types used by the new device and not already in the component library. Since the representation scheme is still being experimented with, we currently have only about twenty different component types in the component library.

A component type is abstracted at two levels. At level-1, it is a module (black box) with I/O ports and a functional description. At level-2, its subparts and connections are described. In a previous implementation, these two levels were represented as two instantiation rules [7]. Since, usually, only a few subparts of an object are relevant to further diagnostic investigation, instantiating all subparts is inefficient. As an improvement, the two levels are now represented as an instantiation rule and a structural template [10]. An instantiation rule instantiates an object with its ports and functional associations. A structural template is a piece of passive knowledge, which allows VMES to search the suspicious subparts of an object. Unlike other procedural representations of the level-2 abstraction [1,7], the structural template itself is never fired or copied. Since the investigation of a suspect is often terminated without checking its subparts, the clear distinction of the two levels of abstraction along with their separate representation has advantages on diagnosis and representation efficiencies.

Explicit representation of wires and POCONs is necessary for diagnosing faults of circuit connections [11]. The traditional model of a wire as a uni-directional module is inappropriate, because it ignores its bi-directional nature, and it does not include POCONs. In VMES, a wire is modeled as a bi-directional module to preserve its physical property, and its uni-directional design intention is retained by the connection mechanism. Components are connected either by forming a POCON from two different ports or by superimposing two ports, which are a same port abstracted at two different hierarchical levels, together. With this new model, VMES is able to locate interrupted wires and bad contact points.

Adding Physical Representation

Human diagnosticians of electronic devices seem to simultaneously maintain models of the logical and physical structures of the target device. They carry out most of the diagnostic reasoning over the logical structure of the device due to its functional association. While carrying out the reasoning, the logical structure is apparently mapped to the physical structure from time to time. Tests and measurements are first initialized using the logical structure, and then are realized and executed on the physical structure. Repair, which is usually done by replacing a physical unit or by fixing a physical connection, is planned and done on the physical structure. In other words, maintenance technicians use a model of physical structure of the target device, which is a hierarchically arranged set of replaceable physical components at various maintenance levels such as field-level and depot-level. By mapping the logical structure of the device to its physical equivalent, maintenance technicians are able to terminate the diagnostic process at the right moment and to form an adequate repair plan.

Given that the mapping between the logical structure of the device and its physical equivalent happens throughout the diagnostic process at all hierarchical levels, the speed in carrying out the mapping is critical to the time needed to locate faults. This implies that objects on both the logical structure and the physical structure of the device should be closely linked to each other so that the mapping is done efficiently. Even experienced technicians may have difficulty in locating a point of a schematic diagram on the real device, where the schematic diagram represents the logical structure of the device, and the form of the real device is the physical structure; which is attributable to the large difference between the logical and the physical structures and a lack of cross-links at all hierarchical levels of the device in human memory. On the other hand, when modeling and representing a device in an automatic fault diagnosis system, the cross-links between its logical structure and physical structure can be modeled and represented to an appropriate level of detail. This is indeed possible to do in a computer with reasonably sized memory.

In VMES, the physical structure of a device is represented distinctly from but in a similar way as its logical structure. In a structural template for a logical component type, every subpart of the component type is specified with a subpart "id" and a subpart "type", which are used to instantiate the subpart if it is found to be a suspect and further investigation of it is necessary. In addition to the subpart "id" and "type", an "mntn-lv" indicator is also associated with every subpart of a physical component type. The "mntn-lv" indicator shows the intended maintenance level of the subpart, i.e., the maintenance level where the subpart, if found faulty, is replaced without further diagnosis. The "mntn-lv" indicator is associated with the physical structure rather than the logical structure of a device to reflect the fact that human experts form and carry out a repair plan based on a physical model rather than a logical model of the device.

In order to abstract a device into a model, which can be efficiently represented and interpreted, some abstraction restrictions have to be made. First, the hierarchical trees abstracted from the two perspectives should have the same number of hierarchical levels. Second, the cross-links can only be made at the same hierarchical level. Third, several logical objects on the logical structure can correspond to the same physical object on the physical tree, but a logical object can not spread over several physical objects. This restriction seems unreasonable at first, but a closer investigation of the electronic domain shows the contrary — physical objects in the domain usually have larger grain size than logical objects. This is especially true with modern technology as more and more logical functioning units are being packed into a physical unit, e.g., a simple HexInverter chip (a physical object) contains six independent inverters (logical objects).

Cross-links between Representations

The two representations of the logical and the physical structures of a device are cross-linked at every hierarchical levels. There are two kinds of cross-links between the logical structure and the physical structure of a device. The first kind are cross-links for components. The second kind are cross-links for ports. Cross-links for components are implemented by the "object<*logical.obj*>/inside<*physical.obj*>" semantic network case-frame (Fig. 3(a)). No distinction is necessary as to whether the physical object contains a single logical object or several logical objects. This is because we just care about whether the corresponding physical object of a faulty logical object is at the intended maintenance level and should be replaced, or it is not and the diagnostic process should continue; this is independent of whether the physical objects contains anything else. (Actually, a physical object in the electronic domain is often replaced with most of its parts being intact.)

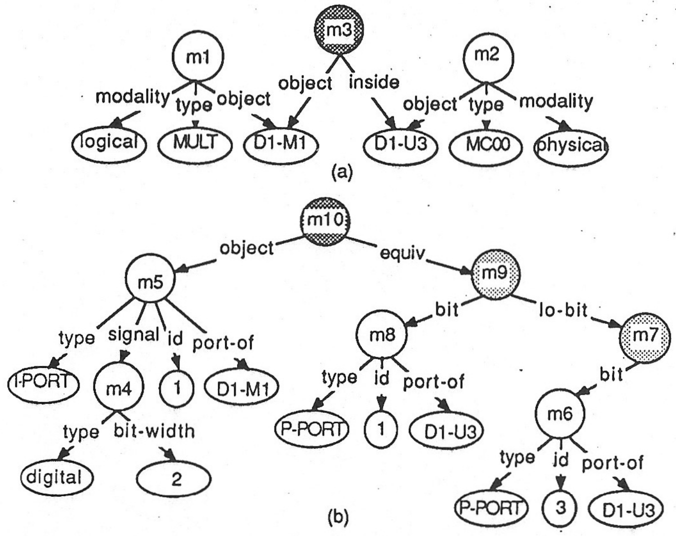

Figure 3 Representation of cross-links between the logical and the physical structures of a device. (a) Component cross-links. (b) Port cross-links.

While the cross-links of components helps in determining if the diagnostic process should go on or terminate, and in forming a repair plan, the cross-links of ports makes user interaction much easier — when ordering a test or a measurement, it can be used to clearly direct the user to the right location on the real device. It is implemented by the "object<*logical.port*>/equiv<*physical.port*>"

semantic network case-frame (Fig. 3(b)). The advantage of a logical abstraction of the device is that it provides a high level view of the device which facilitates the diagnostic reasoning. For instance, a n-bit wire is abstracted as a single logical wire, thus freeing the technician (or a fault diagnosis system) from thinking about bit slices. However, when a measurement is required, it is necessary to locate all the bit-ports on the real device, and this is often a difficult task since these bit-ports may spread out randomly. In our representation (Fig. 3(b)), these bit-ports (physical ports) are linked together, from high-order bit to low-order bit, by the recurrent case-frame of "bit<*a.physical.port*>/lo.bit<*the.remaining.lower.bits*>".

Physical Representation at Work

In the rest of this section, we describe how VMES uses this device representation to facilitate fault diagnosis and user interaction. When bad outputs are found in the suspect currently being investigated, the system has to determine if the diagnosis should terminate or not. Most fault diagnosis systems use the simple idea of SRU (smallest replaceable unit) which says that the diagnostic process stops when the current suspect is a SRU, i.e., a terminal node (a leaf) of the structural hierarchical tree of the device [1,2]. VMES takes a more flexible approach by incorporating the idea of "intended maintenance level" into the system. A system parameter, VMES.IML, is set to the "intended maintenance level" the system is working on. If a part shows some bad outputs and it is at the intended maintenance level, it is declared faulty and the diagnosis on it is terminated. For example, a board is replaced at *field* and then sent back to a *depot* where the fault is further isolated to a chip. The checking for the maintenance level of a part is done on the corresponding physical object of the part (a logical object), and a repair plan is formed based on the component type of the physical object. VMES also provides an opportunity for the user to short-cut the diagnosis by noticing that all remaining (logical) suspects are in a single replaceable physical unit at VMES.IML. Since the same physical object gets replaced no matter which logical suspect is faulty, further discrimination among the suspects are unnecessary provided that connections are assumed to be intact.

The major interaction between VMES and the user is the input of port values. Since diagnostic reasoning is carried out on the logical model of the device, VMES always wants the value of a logical port. Through the cross-links between logical and the physical structures, VMES is able to inform the user which "physical ports" should be measured for a logical port. Note that in digital circuits, a logical port may corresponding to several randomly spread-out physical ports (or pins of chips). Two examples of how the physical representation of a device helps the user in executing a port value measurement are shown in Fig. 4. The port to be measured in Fig. 4(a) is a port of a common component (non-wire component); and the one in Fig. 4(b) is a bi-directional port (a wire-end) of a wire. For representation and display efficiencies, wires are excluded from the physical representation of a device; this does not hurt the user interaction since the wire-end of a wire can always be identified as the wire-end connected to a port of a common component in the physical representation as shown in Fig. 4(b). Note that two kinds of values a user can type in: decimal and binary, where the binary numbers are prefixed by the letters "B" or "b". The user interaction shown in Fig. 4 is through pure text in SNePSUL (SNePS User Language) format, it can be improved by implementing it in natural language and graphics [8].

```
what's the value of port
(m316 (signal (m152 (bit-width (4)) (type (D))))
        (id (inp2)) (port-of (D1-A2)) (type (I-PORT)))
Equivalent Physical Port from Hi-bit to Lo-bit:
(m398 (id (2)) (type (P-PORT)) (port-of (D1-U2)))
(m399 (id (4)) (type (P-PORT)) (port-of (D1-U2)))
(m400 (id (6)) (type (P-PORT)) (port-of (D1-U2)))
(m401 (id (8)) (type (P-PORT)) (port-of (D1-U2)))

* [value]/nil?  B0110
```

(a)

```
what's the value of port
(m291 (signal (m35 (bit-width (2)) (type (D))))
        (id (1)) (port-of (D1-W2)) (type (B-PORT)))
Equivalent Physical Port from Hi-bit to Lo-bit:
The WIRE-ENDs connected to
(m427 (id (6)) (type (P-PORT)) (port-of (D1)))
(m428 (id (7)) (type (P-PORT)) (port-of (D1)))

* [value]/nil?  2
```

(b)

Figure 4 Asking a port value measurement. (a) On a common component. (b) On a wire.

```
>>>>> I GOT THE FAULTY PARTS AS >>>>>
(D1-M2)
$$ Repair Order: replace D1-U3 (type:MC00)
done
```

(a)

```
>>>>> I GOT THE FAULTY PARTS AS >>>>>
(D1-W1)
$$ Repair Order: fix the wire connecting
(m420 (id (4)) (type (P-PORT)) (port-of (D1)))
(m389 (id (8)) (type (P-PORT)) (port-of (D1-U3)))
(m429 (id (4)) (type (P-PORT)) (port-of (D1-U3)))
$     and also the wire connecting
(m419 (id (3)) (type (P-PORT)) (port-of (D1)))
(m388 (id (10)) (type (P-PORT)) (port-of (D1-U3)))
(m428 (id (2)) (type (P-PORT)) (port-of (D1-U3)))
done
```

(b)

```
>>>>> I GOT THE FAULTY PARTS AS >>>>>
(m324 (contact (m322 (signal (m35 (bit-width (2)) (type (D))))
                      (id (3)) (port-of (D1-W1)) (type (B-PORT)))
                (m323 (signal (m35 (bit-width (2)) (type (D))))
                      (id (inp1)) (port-of (D1-M2)) (type (I-PORT)))))
$$ Repair Order: fix the contact point at
(m388 (id (10)) (type (P-PORT)) (port-of (D1-U3)))
done
```

(c)

Figure 5 Repair suggestion made by VMES. (a) On a common component. (b) On a wire. (c) On a POCON.

The third use of the physical representation of a device is in repair suggestions. When a faulty object is found or at the end of the diagnosis session, VMES suggests a repair plan to the user according to the type of the faulty object (Fig. 5). If the faulty object is a common component, VMES just suggests that the user replace its corresponding physical part. If it is a wire, the corresponding physical wires are identified for repair. Note that a logical wire may correspond to several physical wires, for example, a 4-bit logical wire is realized by four wires on a printed circuit board; only the physical wires which are responsible for the fault are identified for repair. This is done by decomposing the port value of a logical wire into bit slices to determine which bit(s) are giving incorrect values. Finally, if the faulty object is a POCON (point of contact [11]), that is, it is a bad contact point, the user is directed to the location of the contact point. The physical representation is not only used to form the repair plan, it also helps direct the user to the object or the location on the real device where the repair is actually performed. In other words, it provides for better user interaction in both test and repair.

THE INTELLIGENT USER INTERFACE

General Remarks

As described in [7] VMES contains a knowledge based graphics package which is used as part of the VMES user interface. The purpose of this part of the VMES project is to investigate new designs for user interfaces, and to investigate what we have called "Graphical Deep Knowledge". We consider a knowledge representation system to be dealing with Graphical Deep Knowledge (as opposed to graphical knowledge), if the knowledge is organized in a way that makes it accessible not only to display routines, but also supports some form of graphical reasoning with this knowledge.

Naturally, a procedural knowledge paradigm is not acceptable for Graphical Deep Knowledge. While many graphics systems eliminate all information not essential to the purpose of display, our system contains *prima-facie* "redundant" information that is not immediately necessary for display purposes. However, we have found good reason to maintain this additional knowledge and have at least four reasons why additional knowledge adds to the power of a representational system.

(1) It is helpful for a system to know what is currently visible on the screen. A graphical representation looses much of its power if the user cannot refer to the objects shown by that representation. One can convince oneself easily of the importance of this notion by looking at virtually any system of graphical representation (including the diagrams in this paper). The given figures are always referred to by some text and derive their explanatory power from this interaction with the text. However, this requires that the system think in the same relations as the user and maintain the same conceptual units as he does.

(2) Declarative representations are required for any logic based reasoning. This factor has been the prime motivation for the representational tools developed here. An example of a simple reasoning operation would be a situation where the position of one object O1 is known, and it is also known that this object has an indeterminate spatial relation to another object O2, like e. g. leftness. Any person could immediately derive from these facts that the object O2 must therefore be somewhere to the right of O1, and any system that could not follow this step of reasoning would be considered by a user as not intelligent. Reasoning in the domain of graphics is especially interesting, because not only traditional forms of logic based reasoning have to be investigated, but also analog reasoning is of interest.

(3) Modern software engineering has made the concept of modularization mandatory, and very often production programmers divide a program into modules, such that the user interface is one module, and the *host program* that performs the services the user is really interested in is another module. This leads to the existence of an interface between host program and user interface which, according to methods of good program design, has to be kept well defined and small. The result of this modularization is that the important concepts of the host program are not available to the user interface, and vice versa. Therefore a user can only refer to units the system designer decided explicitly to export from the host program. Knowledge based programming permits the sharing of information between different modules without creating a bottle-neck between them, and without having dangerous global information available to several modules. The reason why a knowledge base is not anywhere near as dangerous as the sharing of global variables is that knowledge bases are dealing with facts that are considered generally true, and if a fact is true there is no reason to keep it private to a single module, and little danger of conflicting definition or access. This last statement is especially true for rule-based systems. A rule expressing that A is left of B if and only if B is right of A is a universal truth that can be made available to any module in any system.

(4) If a knowledge based system supplies tools for natural language interaction a knowledge base containing Graphical Deep Knowledge becomes in an interesting sense an interlingua, namely an interlingua between the visual and the linguistic faculties of the system. Given that most knowledge based systems have been created with some consideration of natural language processing this observation should be of general interest to KR research. Specifically the SNePS system has a number of tools for natural language processing which permit the implied interactions.

Some other comments on the use of knowledge based methodologies in user interface design can be found in [4] and [9].

The TINA Graphics Interface to VMES

Three user interfaces have been developed for the VMES system which are in different states of completion and integration with the maintenance reasoning program. The first interface is the "TINA" program for knowledge based image generation. This program maps a declarative knowledge structure into a diagrammatic representation on a visual display device. This module exists in two versions with different focus, one of which has been used in the past by the maintenance reasoner to inform the user about the current state of the diagnosis process. Details of this representation have been reported elsewhere [7], but it should be pointed out that symbol colors are an important aspect of this representational facility.

It is relevant to the discussion of TINA that an important mode of display for which the theoretical ground work has been layed in this project, called the Intelligent Machine Drafting mode, has been developed. Traditional CAD systems for circuit boards are usually concerned with the maintenance of graphical representations of wire plans that show a physical picture of the device. In contrast, technicians usually look (also) at logical wire plans. Logical wire plans are interestingly different from physical ones in that no absolute positions for components need to be maintained, only certain connectivity relations.

Nevertheless it is possible to draw the same wire plan in two different ways such that one of them brings across the idea of the representation, and the other one doesn't. It is the goal of TINA in IMD mode to create a logical diagram that is "easy to read", by laying it out and routing it not according to principles of CAD like minimization of energy consumption, but according to principles of minimal cognitive complexity. The most important such principle that has been used is the *equal distribution* of structure in the given space.

It goes without saying that a knowledge base for use with IMD mode does not contain any knowledge of coordinates, and that the major effort in the process of display is the reconstruction of this knowledge from hierarchy and connectivity information. The abilities of the IMD module in use are limited to a small device class that we refer to as A*M* and which has been modeled around the "Adder-Multiplier" (Fig. 6), a device famous in the maintenance literature. A*M* permits small variations of the Adder-Multiplier, for instance variations in the number of ports per components, in the number of components per row, in the number of processing rows, and in the number of connections per port. A formal description of the device class has to be omitted due to limitations of space.

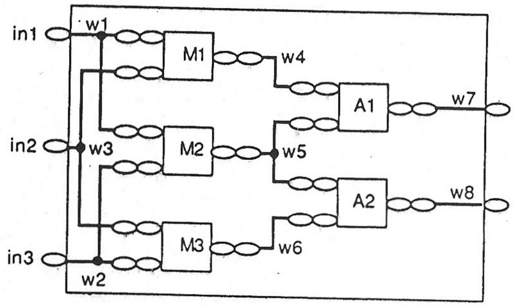

Figure 6 A 3-multiplier/2-adder board.

The Readform Interface for Object Creation

The second interface is the "Readform" program which is used for the creation of visual icons in a format that is accessible to the knowledge representation system. This avoids the necessity of hand generation of graphics code. The compilation of a larger pictorial unit is done by asserting information about objects in the network, such that in the process of drawing access is made to the icons created by Readform. A knowledge based version of Readform has been in the process of development for some time, however as of this writing only the theory of this system will be claimed.

By observing users in the process of object creation (with Vanilla flavored Readform) it has become obvious that the internal conceptual structures of the person can to a certain degree be derived from the order of his actions as well as by asking a few questions at strategic points. Readform supplies the user with a scratch buffer which is separate from the object created at the current moment.

Users have been observed to create objects by drawing a simple unit in the scratch buffer and then repeatedly yanking the buffer content into the picture.

From this chain of actions one can derive that all the yanked objects are presumably members of a certain class, and the system can verify this by asking the user whether there is in fact such a class, and if so, how to name it. This information can be used to create exactly the Graphical Deep Knowledge Structures that have been mentioned before as being used for picture generation.

The other thing that can be derived from the above chain of user interactions is that all the yanked icons are presumably parts of a larger object which consists of all the iconic primitives (lines, arcs, etc.) which were not created by using the scratch buffer. This permits a system to ask the user whether he wishes to name this larger object separately, and if he desires so, a part relation between the yanked parts and the main structure can be formulated and stored in the knowledge base as a proposition. This proposition becomes part of the part hierarchy in the knowledge base. Part hierarchies are a backbone of many representational systems and are used in the process of maintenance reasoning as well as having major importance in the derivation of pictures from Graphical Deep Knowledge and in controling complexity of displayable pictures.

The third user interface that we will treat in this paper is the natural language interface.

The Natural Language Interface

A versatile maintenance system is in need of a user interface in two different situations. In the first situation a maintenance technician uses the system to get help in troubleshooting a currently faulty device. The second situation is as important, namely the initial creation of the device representation. In order to deserve the title "versatile" it must be possible to create device representations with ease and flexibility. The natural language interface that will be described here belongs to the second class of interfaces. It is the goal of this interface to create an internal device representation to the point where it is possible to display the whole device. However, as much of this creation as possible should be done with natural language.

As has been pointed out in the section on Intelligent Machine Drafting, there is no necessity to actually enter coordinate information, so the natural language descriptions become quite natural. Natural language processing is done by way of an ATN interpreter/compiler that is part of the SNePS environment [6]. The class of objects that can be built by natural language is limited, even in comparison to the already limited class A*M* of displayable devices. The major additional limitation that is imposed by the language interface is the branching factor of electrical connections. It is possible to create wires impinging on at most three port.

Below, the original set of sentences that is understood by the NL interface and that describes the Adder-Multiplier will be presented. Running this set of sentences through the ATN interpreter will create all the structures necessary to describe the Adder-Multiplier *completely* for display purposes.

(nl)
D1 is a board
D1M1 is a multiplier
D1M2 is a multiplier
D1M3 is a multiplier

```
D1A1 is an adder
D1A2 is an adder
D1 has 3 inports
D1 has 2 outports
D1M1 has 2 inports
D1M1 has 1 outport
D1M2 has 2 inports
D1M2 has 1 outport
D1M3 has 2 inports
D1M3 has 1 outport
D1A1 has 2 inports
D1A1 has 1 outport
D1A2 has 2 inports
D1A2 has 1 outport
connect input 1 of D1 with input 1 of D1M1 and input 1 of D1M2
connect input 2 of D1 with input 2 of D1M1 and input 1 of D1M3
connect input 3 of D1 with input 2 of D1M2 and input 2 of D1M3
connect output 1 of D1M1 with input 1 of D1A1
connect output 1 of D1M2 with input 2 of D1A1 and input 1 of D1A2
connect output 1 of D1M3 with input 2 of D1A2
connect output 1 of D1A1 with output 1 of D1
connect output 1 of D1A2 with output 2 of D1
D1M1, D1M2, D1M3, D1A1, and D1A2 are parts of D1
wires are parts of D1
the form of a board is xboard2
the form of a multiplier is xmult2
the form of an adder is xadd2
the form of a PORT is xport
^end
```

The first (nl) above calls the natural language processor from the SNePS environment, while the ^end at the end returns to the SNePS environment. Although the vocabulary of this interface is quite limited there are variations of the sentences shown above possible.

Of special interest are the final sentences that start with "the form" because these sentences call, if necessary, the before mentioned Readform interface from inside the ATN interpreter and not only assert the relations between object class and form, but also create any unknown form-icons by having the user draw this icon. If the form is already known to the system, then only the assertional component of this operation will be executed.

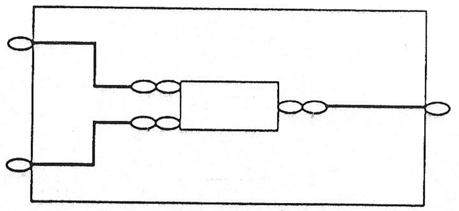

Figure 7 A low-end member of M*A*.

This last operation involves the call of Readform from the ATN interpreter which itself runs embedded in SNePS which itself runs on top of Franz LISP. While display operations are extremely slow (hours on a VAX 11/750 for non-trivial layout tasks), the natural language interface works reasonably fast (response times under a minute) considering especially the multiple layering of the used systems.

A device consisting of a single multiplier with two inputs and one output and the three wires needed to connect the multiplier to the three device ports, plus the six biports of the wires themselves are laid out in about 20 minutes (Fig. 7). This device has also been created completely by natural language interactions and it represents a low end member of the A*M* class.

CONCLUSION

In diagnostic problem solving, human experts seem to use both the logical structure and the physical structure of the target device throughout the diagnostic process at every hierarchical level. Knowledge of the logical structure of the target device together with the associating functional knowledge is used for diagnostic reasoning, and knowledge of its physical structure is used to carry out a test, to determine when the diagnostic process be terminated, and to form a repair plan. It is important to incorporate the physical representation and the logical representation of a device in maintenance. We find that a physical representation of the target device, together with the representation of the cross-links between the logical and the physical structures of the device, contributes to fault diagnosis in several aspects — such a system does not merely mimic the behavior of human experts, it may outperform human experts in certain situations. It helps determine when a diagnostic process should be terminated, thus it provides versatility across maintenance levels. It can provide a short-cut to diagnosis by noticing that all logical suspects are in a physical object at the intended maintenance level. It helps to form a repair plan based on the physical nature of the target device. Finally, (probably the most important point,) physical representation eases user interaction: it helps direct the user to the exact location in the real device for test and repair.

It has been argued in this paper that knowledge based methodologies are of increasing importance for intelligent systems. They permit intelligent behavior of the interface and help to offset problems in information privacy that are enforced by modern software engineering technology. A powerful set of user interfaces is also a precondition for a versatile system, because most expert systems have to talk with people of different requirements during their life cycle. Specifically three user interfaces have been introduced in this paper. The first one caters to the end user, which for VMES is the maintenance technician. It creates graphical representations of circuit boards. The specific research contribution of this part of VMES is the creation of logical wire plans without any prior knowledge about coordinate values of the system icons. The other two interfaces are mainly of use for the device designer who wants to enter information about a newly created device into the maintenance system, without having to learn some obscure graphics or KR language. The first of these two interfaces permits the creation of graphical icons of new components. This interface is called from inside the natural language interface, if a user attempts to use a primitive form which was not previously declared. The natural language interface is based on the SNePS ATN interpreter, and the complete necessary natural language input for the creation of an artificial device called the Adder-Multiplier has been presented.

ACKNOWLEDGEMENT

This work was supported in part by the 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, which supports the Northeast Artificial Intelligence Consortium (NAIC).

REFERENCES

1. R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence 24* (1984), 347-410.
2. M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence 24* (1984), 411-436.
3. D. P. McKay and S. C. Shapiro, "Using Active Connection Graphs for Reasoning with Recursive Rules," in *Proc. of IJCAI-81*, William Kaufman, Los Altos, CA, 1981, 368-374.
4. B. Neches and T. Kaczmarek, "Knowledge Based Interfaces," in *AAAI-86 Workshop on Intelligence in Interfaces*, August 14, 1986.
5. S. C. Shapiro, "The SNePS Semantic Network Processing System," in *Associative Networks: The Representation and Use of Knowledge by Computers*, N. V. Findler (editor), Academic Press, New York, 1979, 179-203.
6. S. C. Shapiro, "Generalized Augmented Transition Network Grammars For Generation From Semantic Networks," *The American Journal of Computational Linguistics 8*, 1 (1982), 12-25.
7. S. C. Shapiro, S. N. Srihari, M. R. Taie and J. Geller, "VMES: A Network-Based Versatile Maintenance Expert System," in *Proc. of 1st International Conference on Applications of AI to Engineering Problems*, Springer-Verlag, New York, April 1986, 925-936.
8. S. C. Shapiro and J. Geller, "Artificial Intelligence and Automated Design," in *Proc. of the SUNY Buffalo Symposium on CAD: The Computability of Design*, SUNY at Buffalo, NY, 1986.
9. S. C. Shapiro and J. Geller, "Knowledge Based Interfaces," in *AAAI-86 Workshop on Intelligence in Interfaces*, B. Neches and T. Kaczmarek (editor), August 14, 1986, 31-36.
10. M. R. Taie, S. N. Srihari, J. Geller and S. C. Shapiro, "Device Representation Using Instantiation Rules and Structural Templates," in *Proc. of Canadian AI Conference - 86*, Presses de l'Université du Québec, Montréal, Canada, May 1986, 124-128.
11. M. R. Taie and S. N. Srihari, "Modeling Connections for Circuit Diagnosis," in *Proc. of The 3rd IEEE Conference on AI Applications*, IEEE Computer Society Press, Orlando, FL, Feb. 1987, 81-86.