

9. Holonic Diagnosis for an Automotive Assembly Line

D.H. Jarvis and J.H. Jarvis

Agent Oriented Software Pty. Ltd.

Australia

Email: {dennis.jarvis | jacquie.jarvis}@agent-software.com

Abstract. Diagnosis is an important function of a holonic manufacturing system if the desired levels of stability, adaptability and flexibility are to be achieved. Our research agenda is to study holonic behaviours (such as diagnosis and control) through the incorporation of these behaviours into operational industrial systems. Given the lack of fielded holonic solutions in industry, we are currently constrained to use conventional systems in our work. In this paper we describe the development of a holonic diagnostic capability for a PLC-controlled vehicle assembly line. A novel model-based strategy is used for diagnosis. Because of the constraints imposed on model formation in this environment, a two-phase approach consisting of off-line fault space generation and online fault space analysis is used. The fault space analysis utilises heuristics to achieve the desired performance levels (diagnosis in less than 60 seconds and success rates of greater than 90%). Areas for further research in holonic diagnosis are identified.

9.1 Introduction

In [9.1], the Hungarian author and philosopher Arthur Koestler proposed the word “holon” to describe a basic unit of organisation in biological and social systems. “Holon” is a combination of the Greek word *holos*, meaning whole, and the suffix *on*, meaning particle or part. Koestler observed that in living organisms and in social organisations, entirely self-supporting non-interacting entities did not exist. Every identifiable unit of organisation, such as a single cell in an animal or a family unit in a society, consists of more basic units (plasma and nucleus, parents and siblings) while at the same time forming a part of a larger unit of organisation (a muscle tissue or community). A holon, as Koestler devised the term, is an identifiable part of a system that has a unique identity yet is made up of subordinate parts and in turn is part of a larger whole. The strength of holonic organisation, or holarchy, is that it enables the construction of very complex systems that are nonetheless efficient in the use of resources, highly resilient to disturbances (both internal and external) and adaptable to changes in the environment in which they exist. All these characteristics can be observed in biological and social systems. The task of the Holonic Manufacturing Systems consortium is to

translate the concepts that Koestler developed for social organisations and organisms into a set of appropriate concepts for manufacturing industries. The goal of this work is to attain in manufacturing the benefits that holonic organisation provides to living organisms and societies, e.g. stability in the face of disturbances, adaptability and flexibility in the face of change and efficient use of available resources.

Viewed in the above context, diagnosis is clearly an important function of a holonic manufacturing system if the desired levels of stability, adaptability and flexibility are to be achieved. Our research agenda is to study holonic behaviours (such as diagnosis) through the incorporation of those behaviours into operational industrial systems. Given the lack of fielded holonic solutions in industry, we are currently constrained to use conventional systems in our work. In this paper we describe the development of a holonic diagnostic capability for a PLC-controlled vehicle assembly line.

9.2 The Testbed

The testbed was a low-volume vehicle assembly line for an Australian automotive manufacturer. It consisted of three assembly stations (Stations 10, 20 and 30) linked by a transfer line. It had a well-defined hierarchical structure, making it ideal for an exploration of holonic diagnosis. Two types of bodies were assembled on the line (Style A and Style B). The line was controlled by a single PLC with approximately 700 I/O points. The control logic was similar to that of the larger systems within the company.

The operations performed at each station were an alternating sequence of automatic steps (known as stages) and manual steps. Automatic operation at a station was initiated by an operator (or operators) depressing one or more sets of palm buttons. Those buttons remained depressed for the duration of that step. The activities performed during automatic operation typically involved the opening and closing of clamps. On completion of an automatic step (indicated by a lamp being illuminated on the station control panel), the operator removed his or her hands from the palm buttons and initiated a sequence of manual activities. These typically involved the loading of panels and/or welding. When the assembly operations had been completed at a station, the operator needed to wait until assembly has been completed in the other stations. At that point, the transfer line was activated (by the operators at each station simultaneously depressing their palm buttons) and the station cycle began again with a new body. Note that the number of stages in a cycle depended on the model that was being built.

When a fault occurs because of a malfunction of a PLC-controlled I/O point, (for example, a limit switch not making contact), it is often very difficult to determine the cause of the problem. The reason for this is that the complexity of the control pathways means that the manufacturing process might not stop until

some time after the actual fault has occurred (and further along in the manufacturing sequence). To compound this problem, it is also difficult to establish valid conditions for resuming production on such systems – the downtime between fault diagnosis and production restart is often significant and costly.

One of the major obstacles that has prevented the development of effective diagnostic systems in such environments is that conventional rule-based approaches are unable to produce comprehensive systems in reasonable timeframes [9.2]. The underlying reason for this is that rule-based diagnostic systems are constructed by associating a set of states with a set of observed faults. This association is normally performed by people who have had extensive experience with the underlying manufacturing system. Unfortunately, when dealing with complex, one-off manufacturing systems, it becomes impossible for a single person (or even a group of people) to experience a sufficiently wide range of fault states to generate a comprehensive diagnostic system. Furthermore, the knowledge acquisition task involved in the development of such a system is extremely time-consuming (and therefore expensive), thus providing more disincentive.

It is our belief that a model-based approach offers a much more attractive path for the development of comprehensive and timely diagnostic systems. In a model-based approach, one would first construct a model of the manufacturing system. Reasoning strategies would then be developed that enabled an association to be made dynamically between an observed fault state and an underlying cause. This approach is particularly attractive for holonic diagnosis, as each holon in the system would have its own model. For example, in the case of our testbed, if one formed a holarchy consisting of the assembly line and the three assembly stations, then we would develop models for each of the four holons.

In the case of PLC-controlled manufacturing systems, the bulk of the knowledge required to construct the model is already available in the form of the PLC program. However, in manufacturing systems, it is common for a PLC output to indirectly affect a PLC input. For example, a PLC output may cause a clamp to close. The closure of that clamp might then be detected by a limit switch, which then drives a PLC input. Such associations are not specified in a PLC program, and in order to construct an operational model of the system, the control logic embodied in the PLC program needs to be augmented with information about the state behaviours of the devices that are connected to the PLC and about how those devices affect each other. This additional information can normally be extracted from wiring diagrams and the pneumatic and hydraulic circuit diagrams. Note that our reliance on experts is greatly reduced, and that if efficient reasoning strategies can be devised, our fault coverage will be comprehensive.

9.3 Model Based Diagnosis

In model-based diagnosis, one reasons directly about the system, using an appropriate model. In the case of PLC controlled discrete manufacturing systems, such a model needs to be able to simulate the behaviour of the system at a causal level (i.e. to generate the sequence of manufacturing events that occur). In the previous section, we have shown that such models can be constructed given the control logic and the finite state behaviours of the components of the manufacturing system.

Discrete manufacturing systems are most naturally represented by component-oriented models. Such models contain objects which correspond to the system's components on a one-to-one basis. These components will normally correspond to physical entities (e.g. adders and multipliers in a digital circuit, or limit switches or clamps in a manufacturing system). Component-oriented models explicitly state how a system's components are connected together and how their outputs are derived from their inputs. They directly represent the structure and function of the system and can simulate its operation. Model-based reasoning has been successfully applied to models other than component-oriented; the interested reader is referred to [9.3] for further details. A fundamental issue in the design of diagnostic systems that use a component-oriented approach is whether to build faulty behaviours into component descriptions or to dispense with explicit fault models and dynamically infer which components are faulty. The latter approach was pioneered by Davis [9.4] (constraint suspension) and de Kleer and Williams [9.5] (GDE); the former approach was pioneered by Bratko et al. [9.6] (Kardio). These three approaches are discussed below.

1. *Constraint suspension*: Davis modelled the intended behaviour of a system as a network of interconnected constraints, where each constraint captures the behaviour of one of the components (or connections) [9.4]. If the device is exhibiting abnormal behaviour, then the observed outputs will not correspond with those predicted from a consideration of the constraint network. Normally, contradictions in constraint networks are resolved by the retraction of one or more input values. Davis, however, retracted constraints (i.e. component behaviours) in order to generate a consistent network. He called this process constraint suspension. Diagnosis then becomes a process of finding a set of constraints which, when suspended, accounts for the observed behaviour.
2. *GDE*: De Kleer and Williams used an assumption-based truth maintenance system (ATMS) as the cornerstone of the diagnostic process in their system, called GDE [9.5]. An ATMS [9.7, 9.8] is a mechanism for maintaining alternative states of a world which are consistent with some theory, but in which different assumptions hold. In a diagnostic application, this theory is the device description, and the alternative states of the world are characterised by all possible combinations of failed components. These combinations are known as candidate sets. Diagnosis in GDE can be viewed as the search of a candidate

set space for the best candidate set. In performing this task, GDE will propose further measurements to help distinguish between candidate sets. The input to GDE consists of the structure of the system, the component behaviour descriptions, the a priori probabilities of component failure and a set of observations.

3. *Kardio*: Dynamically inferring the presence of faults in a system from a knowledge of the correct behaviour of that system is a technique which has been little used outside the realm of digital circuits. This is perhaps no coincidence, as the technique presupposes that the domain in question is governed by a finite (and preferably small) set of laws that are capable of being precisely stated. The alternative approach is to embed fault behaviours into component descriptions. An example of this method can be found in the *Kardio* project [9.6]. *Kardio* is a qualitative model of the electrical control system of the heart. The model was constructed as a network of components such as impulse generators, conduction pathways, impulse summators and ECG generators. Normal and abnormal behaviours were associated with components and a dictionary of simple arrhythmias was incorporated into the model. A simulation algorithm was then implemented which enabled an ECG to be generated for a given combination of arrhythmias. Fault diagnosis with such a system could be achieved by running the simulation backwards, but the number of alternatives that would need to be considered made such an approach unattractive. Instead, what was done was to generate all legal arrhythmia/ECG pairs by exhaustive simulation. An inductive learning program was then used to compact this rule set. These rules could then be used for either diagnosis or prediction. A similar approach was used by Pearce in the development of a diagnostic system for the electrical power sub-system of a satellite [9.9].

Recent work in model-based diagnosis has focused on the use of richer behaviour models which incorporate both normal and faulty behaviours, the development of more sophisticated reasoning strategies to dynamically determine faulty components, and deployment in practical applications. This work is exemplified by projects undertaken at NASA [9.10] and Xerox PARC [9.11]. As part of its New Millennium Program, NASA is exploring the issues of monitoring, diagnosis and repair in the context of autonomous spacecraft. An embedded real-time execution kernel (known as Livingstone) has been developed that uses a compositional, component-based model of the underlying system to perform appropriate actions automatically [9.10]. Each component is modelled as a transition system that specifies the behaviours of operating and failure modes of the component, nominal and failure transitions between modes, and the costs and likelihoods of transitions. The transition system model is a composition of its component transition systems. Livingstone uses its transition system model both to identify the current configuration of the system and to move the system into a new configuration that achieves the desired configuration goals. To achieve the stringent demands of real-time performance, Livingstone reduces each function

to a deductive search problem on a propositional database. This search must be completed before the system moves to the next state, with required response times of the order of hundreds of milliseconds. Hence the success of Livingstone's model-based execution paradigm hinges critically upon the efficiency of the propositional deductive database. Livingstone uses a logic-based truth maintenance system (LTMS) [9.12] to achieve the desired performance.

The Model Based Computing project at Xerox PARC [9.11] arose from the observation that large amounts of software are developed for electromechanical machines, such as software for design, control, scheduling, simulation, diagnosis and productivity analysis. Software like this has to contain a lot of information about the machines, and about their components, particular configurations, normal and faulty behaviours, interaction and timing issues, etc. There is an emerging view that not only should this machine information not be hard-wired into software, but also that it should be prepared separately in the form of declarative, constraint-based models. These models need to capture the structure and behaviour of the machine and they should also be compositional.

Concurrent constraint-based programming provides a natural framework for developing such models [9.11]. A particularly hard part in developing machine software is analysing how machine components or operations interact. This analysis should be done automatically and is facilitated if we can start with models of simple machine elements and then connect them to form larger models. Overall behaviour is then derived automatically from the behaviour of the simpler elements. Besides making modelling easier and less error-prone, this also allows one to automatically and dynamically put together and customise configurations. If such models can be developed, the intuition is that they can be combined with software systems corresponding to specific tasks (e.g. diagnosis) via customised reasoning engines. These engines will take as input the specification of a system configuration, task-specific but machine-independent architectures and algorithms, and the model of the components, to generate the target application. Initial efforts in this field have focused on the development of appropriate modelling and constraint languages [9.13] and kernels for supporting model-based autonomous activity [9.10].

We are unaware of any model-based diagnostic strategies that have been developed specifically for PLC-controlled manufacturing systems. As we shall see, the separation that exists in these systems between the control logic for the total system (as embodied in the PLC program) and the finite state behaviour of the physical components imposes significant constraints on the determination of a suitable strategy.

9.4 The Modelling Process

A model of a PLC-controlled manufacturing system, if it is to be used for fault diagnosis, will need to simulate the following activities:

- *PLC operation*: A PLC typically executes the following algorithm:

```
for (ever)
{
  Read all inputs;
  Evaluate the PLC program;
  Set all outputs;
  Service remote i/o requests;
}
```

One loop is known as a scan. Our objective was not to emulate the detailed operation of the PLC so that accurate scan times could be determined, but rather to simulate its input/output behaviour. That is, having read all the PLC inputs, we need to determine what PLC outputs should be set by evaluating a representation of the PLC program.

- *Manufacturing system operation*: The PLC controls a manufacturing system. We can view the manufacturing system as consisting of two different entity types:
 1. *Actuated devices*: An actuated device is a collection of electromechanical devices that can exist in one of several states. State selection is controlled by one or more PLC outputs. One or more events in the manufacturing process are associated with each state. As an example, consider the actuated device SAV23 in the testbed. It consists of two solenoids, an air valve, a piston and three clamps (12K2-K2, 12S-S and 12O-O). It has two states, designated SAV23A and SAV23B. In the first state, the clamps are open; in the second state, the clamps are closed.
 2. *Sensors*: Sensors enable us to determine whether a particular manufacturing event has occurred. Examples of sensors include limit switches, proximity switches and palm buttons.

These entities typically interact in the following manner:

```

Set a PLC output;
The device changes state;
A manufacturing event occurs (e.g. clamp closure);
if (there is a sensor for this event / state)
{
    The new state is detected (e.g. by a limit switch);
    The associated PLC input is set;
}

```

Ideally we would like to model each actuated device as a holon, but in general the control logic for a single device is difficult to extract from the PLC code because its state transitions are invariably dependent on the state of other devices. The control logic can be extracted in terms of finite-state machines [9.14] but one then has a representation that is different from that of the underlying system (the PLC code) and diagnosis becomes problematic. Consequently, we decided to model each station (and the overall line) as holons. This was a pragmatic decision based on the fact that the control logic for these entities was easily identifiable in the PLC code. The holon models were thus partitioned into two sub-models, one dealing with the control logic (represented as PLC code) and the other dealing with the state behaviour of the holon's actuated devices and sensors. We refer to these as the control model and physical model, respectively. Note that the interaction between the models is via PLC inputs and outputs.

The physical model was implemented as a collection of finite-state machines; devices change states when the appropriate PLC outputs are fired. When a device undergoes a state transition, sensors associated with the previous state are deactivated, and those for the current state are activated. The PLC code was represented as a sequence of AND/OR graphs (one per rung); rung evaluation was initially viewed as a single-source shortest-path problem for a graph with unweighted edges.¹ Some sample output from a simulation trace is shown below:

Stage 1 Ready. Hands On - Clamp Body

```

Y527 = 1 -> <2,1>
Previous State
ST20 7A-A,7B-B LOCATOR UNCLAMP BOTH SIDES
X346 X345 X348 X347 X438 X437 -> OFF

```

¹ Our objective is to determine whether a suitable path between the start node and the finish node exists, not to calculate the shortest such path. Consequently, we can use a shortest-path algorithm, but stop as soon as a suitable path is found.

Current State
ST20 7A-A,7B-B LOCATOR CLAMP BOTH SIDES
X403 X436 X435 -> ON

Y529 = 1 -> <2,1>
Previous State
ST20 7D-D,7F-F,7J-J LOCATOR UNCLAMP BOTH SIDES
X351 X350 X360 X359 X353 X352 -> OFF

Current State
ST20 7D-D,7F-F,7J-J LOCATOR CLAMP BOTH SIDES
X404 -> ON

Y548 = 1 -> <1,2>
Previous State
ST20 OVERHEAD FRAME CLAMP
X482 X481 X486 X485 -> OFF

Current State
ST20 OVERHEAD FRAME UNCLAMP
X484 X483 X488 X487 -> ON

The simulation approach is described in detail elsewhere [9.15, 9.16].

9.5 The Diagnosis Process

9.5.1 Assumptions

The assumptions that we made in developing the diagnostic system are the following:

1. There will be a single point of failure.
2. The point of failure will be a PLC input (typically a limit switch).

The justification for the first assumption is that independent multiple faults which all contribute to a line stoppage are unlikely to occur. Certainly, multiple independent faults can occur in an assembly system, but our observation has been that these faults are benign – that is, limit switches had failed in such a way that the station was still completing its cycle (and consequently maintenance personnel were unaware of the problems). This was possible because of the way in which the system was designed. If one inspects the pneumatic circuit diagrams, one will see that in many cases, only one device state (e.g. clamp open) is explicitly checked for with a limit switch. If the PLC program then only checks to see whether that event has happened, and if the limit switch associated with that event fails in the on position, PLC execution will continue as normal and the station will cycle normally. (However, note that if the clamp fails to operate correctly for some reason, we have the potential for major damage to occur.) This poses a challenge to our single-fault assumption; even if we were to regularly check for benign faults (e.g. at the end of each shift) we would have no guarantee that, when a line stoppage occurred, a benign fault had not occurred since the last check.

The second assumption can be relaxed if required; if it proves necessary, we can induce faults in PLC outputs. It was not relaxed in this project, because maintenance personnel experienced no difficulties in determining the cause of the stoppage in these situations – their major difficulty was in finding faults associated with a single PLC input. Note that this would allow us to diagnose multiple dependent faults; e.g. if a cylinder failed to advance, then all limit switches associated with all clamp actions associated with cylinder advancement would be in error.

9.5.2 Strategy

The conceptual strategy that we used for diagnosis within a holon is presented below:

```
capture the PLC state when an automatic step fails to
complete
determine the stage during which failure occurred
for (all possible faults in the holon)
{
    simulate the behaviour of the holon with that fault
    induced
```

```

if (the simulation fails in the same stage as the
    physical system)
{
    compare observed state and simulated state
    if (observed state == simulated state)
    {
        report cause of fault
        exit
    }
}
}
report failure to identify cause of fault

```

In implementing such a strategy, there are two major issues that need to be considered (assuming that we have a working simulation of each holon, that we have the necessary infrastructure to capture PLC states and that efficiency is not a concern):

1. How do we determine the fault space?
2. How do we compare observed and simulated states?

The fault space for a particular station is the set of all possible faults that can occur. As we are dealing with a system which changes with time, a faulty limit switch will be characterised not only by its state (on or off) but also by the point in the cycle when the failure occurred. Our initial belief was that limit switches would fail only when they underwent a change of state from on to off, or vice versa. However, reality proved otherwise: switches get knocked by welding guns, they can get covered in glue, they can accumulate grime or they can just gradually become misaligned or fall apart. In short, a fault can occur at any point in the cycle.

At first glance, the size of the fault space would suggest that the diagnostic strategy described above would be infeasible, as for Station 20 (the most complex of the holons), there are approximately 17,000 points in the full fault space. Note that this is equal to the number of input points (210 for Station 20) multiplied by the number of logical scans in the cycle (75 for Style A, 88 for Style B). While the size of the fault space is large, the size is linearly dependent on these two parameters. Thus if we double the number of inputs or double the number of

scans, we double the size of the fault space. Also, note that the number of points in the fault space that need to be examined can be significantly reduced if we realise that:

- *The fault space can be partitioned according to the stage in which failure will occur for a given fault point:* When we perform diagnosis, we shall be focusing on a subset of the full fault space, namely those faults which, when induced, would cause termination to occur in the step at which failure was observed. Thus we can ignore all fault points which would cause termination in other stages. However, one needs to be aware that induced faults can cause a stage incompleteness in the step at which they were induced, in any subsequent stage in the current cycle or even during subsequent cycles. Thus when we perform diagnosis, we need to consider all faults in preceding steps, in both the current and the previous cycle. (If we are inducing style-specific faults, then the previous cycle means the last time that the current style was processed in the station.)
- *At each point in the fault space, there is only one (of two) possible faults that needs to be induced:* If we assume that limit switches fail only in one of two modes (on and off) then we need only induce one of those failures at each point in the fault space. (At a given point of the simulation, suppose that a limit switch is on. If it fails in the on position, then it cannot impact on cycle completion until we reach the point in the cycle where it is supposed to go off.) Thus, at each point in our fault space, we induce a fault in a limit switch by determining its current state in the simulation (on or off) and then setting it to the opposite state.
- *Benign faults can be ignored:* As was mentioned earlier, if the PLC program checks only to see if a limit switch is in one state (typically on) and the switch fails in the on position, the line will continue to operate normally. Thus, if we can identify these faults, they can be eliminated (or pruned) from the fault space. We were able to identify the bulk of the faults that exhibited this behaviour through an off-line analysis of the PLC program; approximately half the limit switches in the line are checked for one state only.

Comparison of observed and fault states is not as complicated. The issue here is “What is the minimum amount of information that will enable us to identify a simulated state as being the *same* as the observed fault state?” We have chosen in the current implementation to do the comparison on the basis of selected inputs. Owing to production pressures, maintenance personnel were unable to prepare a system which was free of benign faults, so we chose to ignore those inputs which corresponded to benign faults. This has worked well in practice, although in general a PLC state will be characterised by all its inputs, latched outputs and non-volatile memory locations.

9.5.3 Optimisations

Even with the pruning of the fault space as described in the previous section, the amount of computation required by our strategy is excessive. Consequently, we implemented the following two-stage strategy:

```
//Stage 1 (Off-line)
for (each point in the pruned fault space)
{
    induce the fault
    simulate holon behaviour with the fault induced
    record the fault state
    record the stage at which failure occurred
}

//Stage 2 (On-line)
capture the PLC state when a stage completion fails to occur
determine the stage during which failure occurred
for (each point in the pruned fault space)
{
    if (recorded failure stage == observed failure stage)
    {
        if (recorded failure state == observed failure state)
        {
            report cause of fault
            exit
        }
    }
}
}
```

Note that in Stage 1, if fault points are examined in scan sequence, the computation required can be significantly reduced by judicious saving and restoring of simulation states. However, the amount of computation that is required is still excessive if the AND/OR representation of the PLC program is used and rung evaluation is treated as a shortest-path problem. We therefore transformed the PLC program into compilable C code. This was achieved by writing a program which, for each rung, generated all possible paths between the start and end of the rung. These paths then formed the logical-expression component of an `if` statement (the nodes in each path were combined with `&&` operators; the paths for each rung were combined with `||` operators); the consequents of the `if` statement become assignment statements involving the rung output node.

Program evaluation is now achieved by embedding the C code that we generate in a function (called `scan()`). This routine is then compiled along with the remainder of the source code for the system. Using the compilable representation reduced our computation time by a factor of 10; stage 1 of our strategy took about 20 minutes on a SPARCstation 10.

9.5.4 Verification

The diagnostic system consisted of four sub-systems, corresponding to the four holons in the system. The diagnostic system was verified by inducing faults during normal production and then attempting to ascertain the cause of the resultant stoppage by using the diagnostic system. If a station had stopped mid-cycle then the diagnosis was initially performed using the model for that station. If that proved unsuccessful, then the assembly line model was used, and failing that, the downstream stations were used. If the stoppage occurred during transfer, then the assembly line model was used first, followed by the models for each of the stations in turn. Three trials were conducted and 55 faults were induced. On completion of the trials three faults remained unresolved, giving a success rate of ~95%.

9.6 Discussion

The system level strategy that we described in the previous section was not implemented as an automated process. At the request of the operators, the individual diagnosis holons were made available to them, and in performing diagnosis, they used the strategy described above. Implementation of this strategy for the testbed would be straightforward, but the development of general system-level strategies is an area for future research. We would expect that any general strategy would need to incorporate expert knowledge in addition to structural and behavioural models of the system level. This issue becomes even more interesting if you have more than two levels in the hierarchy. Our testbed has a well-defined hierarchical structure – the line consists of stations – stations consist of frames and hoists – frames and hoists consist of clamps, motors and sensors. Unfortunately, the structure below stations is not readily discernible in the PLC code, so we were unable to explore this issue in this particular project.

While a model-based approach to holonic diagnosis is attractive in that it can reduce the reliance on experiential knowledge and that the models are available for other uses, it is by no means a prerequisite for building a holonic diagnostic system. Each application needs to be treated on its merits, and depending on the nature of the system, a conventional rule-based approach or perhaps a neural network may be more appropriate. Having said that, in this particular instance we

would argue very strongly that a model-based approach was the most appropriate approach.

Diagnostic systems are typically evaluated against fault coverage and execution time. Execution time for our system was acceptable (20 – 40 seconds per fault); note that this time is fault-independent. Coverage is more difficult to evaluate. We would expect that if our model was correct, then all single faults would be detected. However, it would appear unrealistic to expect that we can replicate the exact behaviour of such complex systems when limit switches fail on a regular basis, clamps fail to completely close because of variations in panel quality and positioning, and the environment is conducive to intermittent faults (such as grime and glue on limit switches, and limit switches and wires being knocked by welding guns). There may well be some inaccuracies remaining in our model that can be rectified and we shall continue to strive for 100% coverage. However, we consider a 95% success rate to be quite satisfactory, particularly considering that in the trials that we have conducted, we produced no false positives (i.e. whenever the system identified a fault, it was correct), and that the effort involved in generating the diagnosis by the operator is minimal (no input is required, and the elapsed time for a diagnostic run is only 20 – 40 seconds).

We anticipate that fault states will exist for which we cannot assign a unique cause; in such cases, all potential candidates will be reported. Note that the likelihood of this happening is increased because of the limited amount of information that we use in state comparisons. As mentioned earlier, this work has assumed that we shall have a single point of failure, and that it will be a limit switch. If we were to allow devices to fail as well, we could diagnose situations where cylinders had failed to advance or retract. In the current system, these situations would appear as multiple points of failure (all the limit switches associated with a cylinder position would be in error). The system also has the interesting capability of being able to detect intermittent faults, provided that they cause a line stoppage. In this situation, the diagnostic system will indicate that a particular limit switch is faulty, but a casual inspection may indicate that the switch is behaving normally.

Our approach to diagnosis has similarities to that used by Bratko et al. [9.6] in their diagnosis of cardiac arrhythmias. They first developed a qualitative model of the electrical system of the heart. Faults were then induced in this model and the fault states (heart arrhythmias) were recorded. An inductive learning program was then used to generate compressed diagnostic rules from knowledge of the induced faults and their associated fault states. Unlike us, Bratko et al. did not initially prune the search space; the removal of degenerate fault states occurs during the inductive learning phase. Note that we could have used inductive learning to generate compressed diagnostic rules on our pruned fault space, but diagnostic performance was satisfactory using uncompressed rules. This situation may of course change if we apply our approach either to larger systems or to systems where significant pruning of the fault space cannot be achieved.

9.7 Conclusion

A model-based diagnostic system for a PLC-controlled vehicle assembly station has been constructed and tested online. Fault coverage on the test data was 95%, and execution time was 20 – 40 seconds/fault. Experience with the system has identified areas for further research in the area of holonic diagnosis.

References

- [9.1] A. Koestler: *The Ghost in the Machine*, Arkana, London (1967).
- [9.2] W.B. Day and M.J. Rostovsky: "Diagnostic Expert Systems for PLC Controlled Manufacturing Equipment", *International Journal of Computer Integrated Manufacturing*, **7**, 116–122 (1994).
- [9.3] A. Cohn: "Qualitative Reasoning", in *Lecture Notes in Artificial Intelligence*, No. 345, Springer, Berlin, Heidelberg (1988), pp. 60–95.
- [9.4] R. Davis: "Diagnostic Reasoning Based on Structure and Behaviour", *Artificial Intelligence*, **24**, 347–410 (1984).
- [9.5] J. de Kleer and B. Williams: "Diagnosing Multiple Faults", *Artificial Intelligence*, **32**, 97–130 (1987).
- [9.6] I. Bratko, I. Mozetic and N. Lavrac: *Kardio: A Study in Deep and Qualitative Knowledge for Expert Systems*, MIT Press, Cambridge MA (1989).
- [9.7] J. de Kleer: "An Assumption-based TMS", *Artificial Intelligence* **28**, 127–162 (1986).
- [9.8] J. de Kleer: "Problem Solving with the ATMS", *Artificial Intelligence*, **28**, 197–224 (1986).
- [9.9] D. Pearce: "The Induction of Fault Diagnosis Systems from Qualitative Models", in *Proceedings of AAAI-88*, AAAI Press, Menlo Park CA (1988), pp. 353–357.
- [9.10] B. Williams and P. Nayak: "Immobile Robots: AI in the New Millennium", *AI Magazine*, **17**, 16–35 (1996).
- [9.11] V. Saraswat, D. Bobrow and J. de Kleer: *Infrastructure for Model-Based Computing*, Xerox PARC (1993).
- [9.12] K.D. Forbus and J. de Kleer: *Building Problem Solvers*, MIT Press, Cambridge MA (1993).
- [9.13] H. Wong, M. Fromherz, V. Gupta and V. Saraswat: "Control-Based Programming of Electro-Mechanical Controllers", in *Proceedings of IJCAI Workshop on Executable Temporal Logics*, Montreal (1995).
- [9.14] J.H. Jarvis and D.H. Jarvis: "Design Recovery for PLC Controlled Manufacturing Systems", in *Manufacturing Systems: Modelling, Management and Control*, ed. P. Kopacek, Elsevier (1998).
- [9.15] J.H. Jarvis and D.H. Jarvis: "Life Cycle Support for PLC Controlled Manufacturing Systems", in *Software Engineering for Manufacturing Systems*, eds. A. Storr and D. Jarvis, Chapman & Hall, London (1996).
- [9.16] J.H. Jarvis and D.H. Jarvis: "Simulation of a PLC Controlled Assembly Line", in *Simulation in Industry: 9th European Simulation Symposium*, eds. W. Hahn and A. Lehmann, SCS (1997).