Computer Science 773

Robotics and Real-time Control

INTELLIGENT FAILURE MANAGEMENT

It has always been clear that failure management is essentially difficult, because, while it is straightforward (though not necessarily quick) to describe how a machine works, it is essentially impossible to describe how it doesn't work – there are far too many possibilities, many of which involve events in the environment which you cannot easily take into account. There is no guaranteed systematic way to derive a list of all possible faults. Recall that in Britton's specification scheme the procedure for identifying possible exceptions was reduced to interviews rather than being carefully codified – and consider how you would predict the Ariane 5 failure.

Conventional failure management is based on identifying possible faults, or classes of fault, and providing explicit instructions for detecting them and dealing with them when they occur. Many simple and obvious faults can be dealt with in this way, but there are problems associated with this approach because of the combinatorial nature of the problem. Followed exhaustively, the policy would result in a failure procedure for every conceivable fault, and every combination of faults, under all possible condition. Such a collection of failure procedures would be extremely expensive to write, occupy a lot of space (not as serious a problem as it used to be, but still significant for a large system), and hardly any of it would ever be used. This is poor economics. As well as that, the sheer volume and the nature of faults themselves would make the failure code exceedingly difficult to test – so when a fault was detected, the response would quite probably be to execute some inadequately tested code !

Therefore, while it is recognised as sensible to provide explicit failure management procedures for foreseeable and potentially dangerous faults, there has always been much interest in providing more "intelligent" means for dealing with others which might occur. It is accepted that even in a highly automated plant of any complexity you need people around in case of emergency, because they are good at solving unforeseen problems; intelligent fault handling can be seen as an attempt to find programmable ways of automating these people.

SORTS OF INTELLIGENCE.

Two main levels of intelligence are recognised in discussions of intelligent failure management. These are *shallow* intelligence, usually associated with the use of rule-based systems, and *deep* intelligence, associated with the use of system models. Both levels are useful, but they serve different functions.

Shallow intelligence is what we use when we learn simple "quick fix" responses to indications of failure. If a box drops off a conveyor belt, we might know that it's reasonable to put it back on; if the photocopier stops for some non-obvious reason, you switch it off and on again. Both of these are learnt responses which don't require much thought (except perhaps the first time, but that might be deep intelligence – see below), neither is necessarily the right thing to do, but neither is likely to be programmed into the ordinary control system. Perhaps more to the point, both are responses which you might tell someone whom you noticed to be puzzling over a dropped box or recalcitrant photocopier : "Just put it back on the conveyor"; "turn it off and on again".

Deep intelligence operates by analysis of a problem in terms of knowledge of the structure of the system concerned. The first time you come across a box on the floor by a conveyor belt, you might use your knowledge of the downstream process to work out that the spacing and orientation of the boxes on the belt isn't particularly significant, so it doesn't matter if you just put it back in an arbitrary position. The first time your photocopier gets stuck in an incomprehensible way, you might use your general knowledge that photocopier-like machines are unlikely to be damaged by being switched off, and commonly automatically set themselves up in a sensible state when started, to determine your course of action.

SHALLOW INTELLIGENCE.

There are, obviously enough, two steps to the shallow response to a fault : first the fault must be detected, then the appropriate action must be executed. In principle, this can always be written in the form of a simple rule :

```
when <condition observed>
execute <response>.
```

This makes it look very like an ordinary response to an event, such as might be found in any but the simplest control system. The difference is that, first, an ordinary programmed event response is expected to be executed under well defined conditions in a known context, and, second, that with an ordinary programmed event you know that the event itself can be detected.

The context is important because you know the conditions under which the required code will be run; if the initiating condition is not well defined, it might be much less obvious what you can do in `<response>`. That's why a lot of responses amount to "ring a bell loudly"; though the failure has been detected, there's no way for the system to determine just what's wrong in sufficient detail to correct itself.

The context is also important because you know where to put the instruction. What do you do with an arbitrary set of rules like the example when you know that they might be relevant pretty well anywhere ? You can collect them together in some sort of fault block – that's a common device in languages which provide exception management – but you still have to find some way to execute the test.

That brings us to the second qualification : can we detect the event ? The number of manufacturing system controllers which can detect a box lying on the floor is likely to be exceedingly small. (I guess zero, but have no evidence in support.) More realistically, given a condition like `temperature > 500 and flow valve open`, where do you put the test ? Presumably something, somewhere, can read the temperature and has a record of the state of the flow valve, but how do you ensure that the test is executed sufficiently frequently ? And, even if the temperature is polled often enough, how do you insert the required code in the control software once you've determined that it's useful ?

(One of the desired features of PDL was the ability to add such rules easily and quickly. That's one of the reasons for its rather simple structure.)

DEEP INTELLIGENCE.

Simple rule-based responses to detected fault conditions are very useful when you know what you want to do, but they are limited in their scope. You still need a rule to cover every possible failure mode, and the previous criticisms still apply. The idea of telling your programme about the structure of the system and getting it to work out for itself how to respond to faults is therefore very attractive.

It is also very difficult. One of the reasons why people are so good at this sort of problem solving is that we carry around with us a vast quantity of information, and we can retrieve relevant parts of it exceedingly efficiently. If we want computers to perform as we do, we have to find ways of giving them the information and of using it efficiently.

The first of these requirements, so far as it applies to the plant and process, is comparatively easy to handle; we do know what the plant is, and how it is used, and a great deal of this information is written down explicitly somewhere or other. Getting this into a machine-readable form might not be particularly easy, but the task is imaginable and bounded. (Recall the PDL databases, and the rule-based format of its programmes.) The task is a lot harder if we also have to provide the commonsense knowledge which people us in their reasoning, but it seems to be true that one can manage to a useful extent without going to such lengths. The stored information can be in terms of straightforward descriptions of the system, or it can be to some extent predigested into more immediately useful forms; for example, fault trees and event trees might be used to organise possible causes of faults and consequences of events.

Using the information is harder. Diagnosing the fault is a non-trivial problem in artificial intelligence, even given all the available information. To make it tractable, it is common to make assumptions such as that of "single point failure", which is the assumption that only one fault is the cause of the observable symptoms. While that sounds reasonable, in practice it is common for a real very deep cause to lead to several fairly deep consequences perceived as causes from the viewpoint of the software. Not a lot can be done about that.

Alan Creak,
April, 1997.