MODELING, IDENTIFICATION AND CONTROL, 1993, VOL. 14, NO. 1, 27–41
doi:10.4173/mic.1993.1.2

Early fault detection and on-line diagnosis in real-time environments

ANDREAS BYE† and EYVIND NESS†

Keywords: *Operator support systems, fault detection, fault diagnosis, quantitation modeling, rule-based reasoning.*

This paper describes an approach to fault detection and diagnosis involving the simultaneous employment of quantitative and qualitative reasoning techniques. We show that early identification of process anomalies by means of a separate fault detection module paves the way for a fast and accurate follow-up diagnosis. The diagnosis task is dramatically simplified because the diagnostic inferences can be performed at the soonest possible time: when the detection module first spots deviations between its calculated reference points and the corresponding measurements from the process.

The approach taken has proved to be a successful division of work between the detection and diagnosis tasks in a large, complex process. The fault detection module, Early Fault Detection (EFD), has been implemented in our man–machine laboratory equipped with a PWR simulator and installed at the Loviisa nuclear power plant in Finland. The diagnosis modules, comprising the Detailed Diagnosis (DD) system, are currently running in our laboratory, where both DD and EFD are integrated within the cooperating supervision and control environment concept called ISACS.

1. Introduction

The main goal for the Early Fault Detection and Detailed Diagnosis systems, EFD and DD, is to supervise a process, in the sense of detecting anomalies and diagnose them on-line. The real time aspect here is not a time critical control problem, but a surveillance problem, how to get secure information of process faults as fast as possible out to the operator who is responsible for running the process. EFD/DD split the task in two separate units, one detection part and one diagnosis part. The timing aspects are then the internal timing problems in the communication between the quantitative and the qualitative part, the detection part and the diagnosis part of the system.

When diagnosing a process a major problem in knowledge based systems is handling a huge amount of knowledge, reflecting the complexity of the processes which are going to be supervised and the intricate coupling between subprocesses. The basis for the reasoning then becomes very wide and difficult to manage in an efficient way, see Holmstrøm (1989). With our system the searchspace is significantly pruned by reducing the physical area in the process where a fault may possibly be, or where the cause of the anomaly originates.

A conventional alarm system in an industrial process surveys pressures, temperatures and similar physical quantitities and triggers if they get too high or too low. There

Received 20 October 1992.

† Institutt for energiteknikk, OECD Halden Reactor Project, P.O. Box 173, N-1751 Halden, Norway.

This paper was presented at the IFAC Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control, May 29–30, 1991, Bergen, Norway, and is reprinted with permission from Pergamon Press.

are several problems with this approach to fault detection. To avoid false alarms for a dynamic process, the alarm limits should be rather wide. This means that a disturbance may develop quite a bit before it is detected and diagnosed. In order to get warnings sufficiently early to avoid taking drastic countermeasures in restoring normal plant conditions, the alarm limits should be put close to the desired operating points. However, that would cause too many error messages during dynamic plant operation. Another problem is the fault propagation in a complex process, where initiation of a small transient in one part of the process may lead to a big transient in other parts of the process. The fault may then originate from a point far from where the first alarm triggers. This makes it difficult for the operator to assess the cause of the alarm, since several sections of the process might already have been disturbed. Similarly, building diagnostic operator support systems which start their analysis only when an alarm is triggered by a conventional alarm system, also turns out to be a hard task. It proves difficult to write diagnostic rules which can be used in different plant states and can distinguish between fault symptoms and normal consequence alarms.

As an extension of several years' activity on alarm reduction methods, the OECD Halden Reactor Project started in 1985 to develop a fault detection system based on the application of reference models for process sections. Small, decoupled, mathematical models which calculate the physical behaviour of process parts in situations without faults, are running in parallel with the process. Certain key variables from the model, so-called reference variables, are then compared with corresponding measurements from the real process, and if there is a big enough deviation, an alarm is triggered. Therefore faults are detected earlier than by conventional alarm systems, even in dynamic situations. By a proper process decomposition scheme, the problem area can be quickly confined to the faulty process part. In addition, the deviations between plant measurements and the expected behaviour as calculated by the reference models, have established an excellent starting point for a diagnosis system. The problem area is already confined to a particular process section, thus reducing the search space of potential fault hypotheses.

1.1. *Typical transient scenario*

The Early Fault Detection module, EFD for short, is continuously monitoring state variables, calculating possible deviations between their measured and expected values. Typically, one employs a set of mass and heat balance equations together with information on plant topology and component characteristics. Whenever a deviation is detected, EFD submits a message to the on-line diagnosis module, the Detailed Diagnosis (DD) Runner. The message contains a reference to the plant subsystem in which the error appeared together with the values of the associated deviations between measured and calculated variable values in that subsystem. The DD Runner module processes this message, possibly requesting additional information from EFD (typically values and time-derivatives of process variables), and finally comes up with a conclusion about what might be the source of the disturbance.

1.2. *Brief outline of the paper*

This paper is divided into two main sections. § 2, Methodological Foundations, on the underlying methods and concepts applied, and § 3, Technological Foundation, on the technological issues related to our current system implementations.

2. Methodological foundations

In this part of the paper we discuss general aspects of the theory and methods employed in our research. Technical details and operational experience of the EFD and DD systems are given in Ness (1989) and Sørenssen (1990a, b).

2.1. *Quantitative aspects—early fault detection*

The quantitative part, EFD, is handling the detection aspect of the combined EFD/DD system. In reasoning about process failures it is important to be able to see the difference between the cause and the consequences of an anomaly in the process: What is the process failure, and what are propagating consequences? In order to answer these important questions one needs deep knowledge of the process, and in this system simulation models provide this basis.

2.1.1. *Conventional fault detection* As basis for the reasoning many expert systems use the same input as the operators in conventional control rooms do, directly measured state quantities in the process. These are typical quantities as pressures, temperatures, flows etc. To detect anything abnormal, many of these variables have constant alarm limits, where alarms are triggered when something gets too high or too low. Some expert systems are then triggered by such static alarms, to diagnose the process to find the anomaly.

But there are certain weaknesses with conventional alarms.

To avoid false alarms when the process is in normal dynamic transients, for example when running a plant up or down, static alarm limits have to be rather wide. But because of these wide alarm limits, alarms will trigger rather late when there is something wrong. In certain cases malfunctions are not detected at all, when for example a control valve stabilizes the level in a tank even if there is a leakage in the tank.

Another serious problem is false alarms. Worst case behaviour occurs when the first alarm is false, and maybe far away from the cause of the problem, thus misleading both the operator and the reasoning system which is trying to diagnose the failure. Another problem is that after the first alarm has triggered, many other alarms trigger too, because the failure transient propagates to other parts of the process. Shortly the problem may be formulated: Which are the cause alarms, and which are the consequence alarms? With conventional alarm systems one cannot distinguish between them, and if a diagnosis system is triggered by such alarms, the search space of possible fault hypotheses has to be rather large.

2.1.2. *Model-based foundation for qualitative analysis* The goal with this basis is to detect failures at the earliest possible time, reduce the number of alarms in case of an anomaly, limit the physical area where the fault may be, and thereby reduce the search space for the reasoning module. In other words: We want to detect where the fault is at an early stage, with no false alarms.

In EFD we use simulation models in parallel with the real process, running in real-time. These are so-called reference models, which are small, decoupled, mathematical models which calculate the physical behaviour of a process part assuming no faults. Certain key variables from the model, so-called reference variables, are then compared with corresponding measurements from the real process, and if there is a big enough deviation, an alarm is triggered. Figure 1 illustrates the approach taken. In this way the sub-model will follow the behaviour of the real process as long as there is nothing

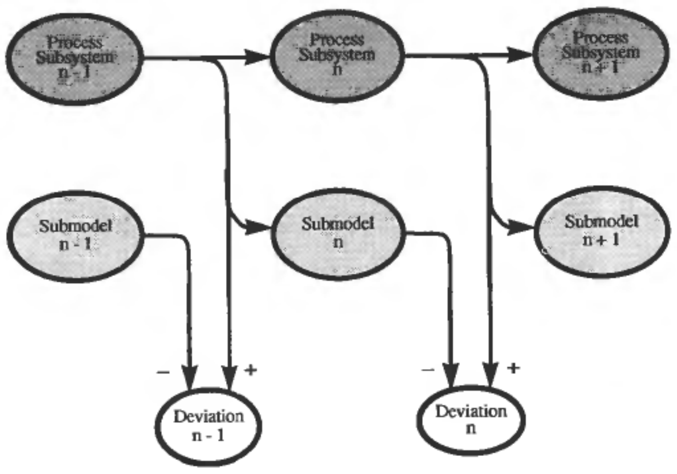


Figure 1. Schematic picture of EFD principles.

wrong in the process. Then we can set narrower alarm limits than the conventional alarm systems, and in this way detect anomalies in the process far earlier, and also detect small anomalies not discovered by traditional alarm systems.

The process models are small, covering as small a part of the process as possible, depending on the instrumentation available. If more measurements are available, it is possible to make smaller submodels. Each process submodel is decoupled from all the others in the sense that it has only input from the real process. In our application the models samples the process typically each third second, but the sampling interval must be adjusted to time constants for the process considered. Each submodel will follow and describe its corresponding process part in normal dynamic situations, even if the state variables inside the model may deviate a lot from what is normal in steady state. In case of a failure inside one sub-system, the calculated state variables will not match the corresponding measurements. This decoupling makes EFD pinpoint the failure, where in the process it is, in both steady state and in dynamic transients, without coming up with false alarms, i.e., for each fault there is one and only one alarm.

One should have in mind that EFD is merely a detection system. If the behaviour of a submodel deviates from the behaviour of the corresponding process subsystem, we know that something is wrong, but not exactly what it is. It may be a process fault, such that the model fails to mirror the corresponding process part, or it may be something wrong with one of the inputs to the model, i.e. a measurement fault. But one thing is for certain, the process area for the fault is located, making the search space for the diagnosis much smaller. The diagnosis is then left to the DD module.

2.1.3. *EFD implementation status* EFD and DD are now running in our man–machine laboratory, HAMMLAB. There they are acting as components in the Integrated Surveillance And Control System, ISACS (Bologna 1991), and coupled to a full scope PWR nuclear power plant simulator.

EFD as a self-contained unit has now been running on the Loviisa nuclear power plant in Finland for over a year, see Sørenssen (1990a). It is supervising the high pressure preheaters in the feedwater trains and has detected internal leakages in the heat exchangers, caused by fouling. The decoupling of this system makes it very easy to focus on special problem parts of the process and survey these, not having to model the whole process. In Loviisa, where EFD runs stand-alone, the diagnosis is left to the operator, who uses EFD and trend curves of key variables to diagnose what the anomaly is. During one year of operation EFD has found two leakages. One of them the plant personnel did not find before they shut down the plant and used a pressure test on the tank in question. So EFD was the far most sensitive on-line tool they had to discover small, slowly developing leakages.

2.1.4. *Surveillance—not control* Since this system is used for surveillance, and not direct control of the process (unless seen in the loop with the operator), certain aspects should be discussed. Robustness is in this case more important than accuracy, so if there for example is a static deviation, this is not as dangerous as in the control case. In the surveillance case one may allow static deviation, and use time changes of the deviations to detect anomalies.

The time discretization method of the models is simple, first order explicit Euler is applied. If a process model is used in direct control, with feedback to the process itself, inaccuracies in the process model caused by the discretization may cause instabilities in the whole system containing the process itself and the model. In our case we have no feedback from the process model to the process. The models are coupled in an open loop, so we have no amplifying effect demanding extreme accuracy in the process models.

Noise on the input signals could have been a problem, because the current models do not contain filters. In the practical use of EFD in Loviisa in Finland, this is solved by using low-pass filtered input signals. This implementation concentrates on relatively slowly developing phenomena, compared with the frequency of the typical noise in signals from the process. It is then very good at detecting creeping faults at an early stage, and also rapid changes resulting in static abnormal behaviour. It could, however, be considered to include on-line filters in the models, e.g. Kalman-filters, to improve the short-time model behaviour regarding noise influence.

2.1.5. *Model validity* For large disturbances in the process one may question whether the models are valid or not. For certain systems it may be difficult to make models which are valid in the whole operating range of the process. In this case a solution would be to make many models, perhaps gathering them in a library, which are valid under different operating conditions. Experience from our work shows that in many cases the models are valid as long as the process topology is known. When the automatics in the process intervene and actually change the process topology one may then switch from one model to another.

2.2. *Qualitative aspects—DD*

In this section the inner workings of the diagnosis modules are put under the magnifying glass.

2.2.1. *Course of events in a typical diagnosis* When a fault has been detected by the EFD module, an alarm is sent† to the diagnosis module. The alarm contains two items: a reference to a particular plant subsystem in which the alarm appeared and an error-pattern, i.e. a list of deviations, each of which in principle is the sign of the difference

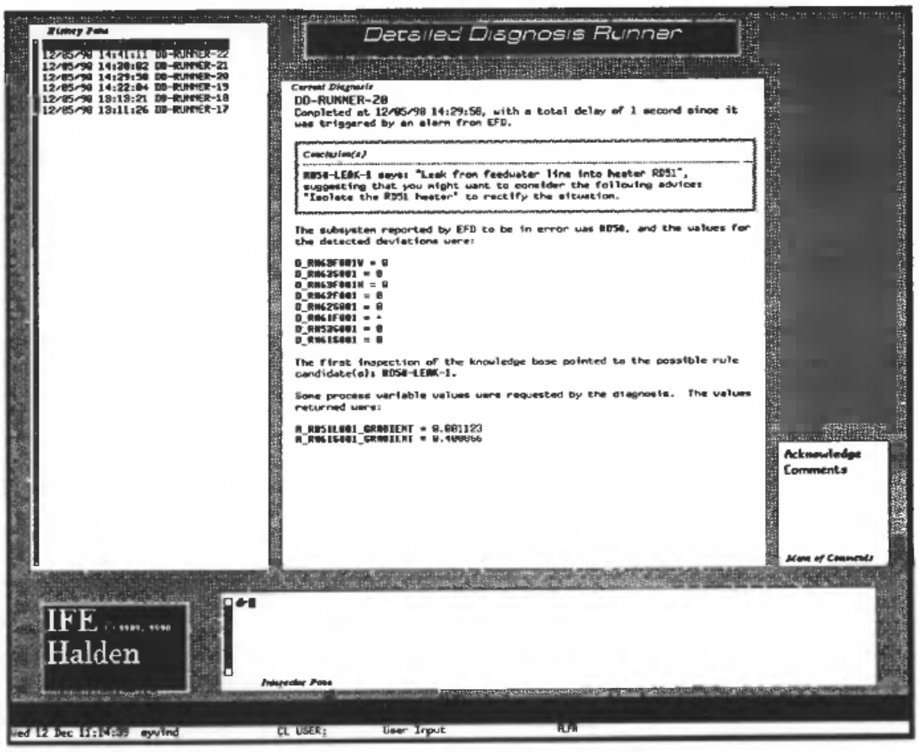
† Possibly via a controlling agent like the Intelligent Coordinator in ISACS—see Bologna 1991.

between a measured and an expected process variable value as calculated by EFD, pertaining to this subsystem. Where a direct measurement is unavailable, we can let a calculation act as one, using one equation, and let the expected value be the value as obtained from another, unrelated equation (calculating the same quantity using two different physical models, if possible).

When the fault-detection message from EFD arrives, a separate diagnosis (computer) process is created to handle this particular event. Based on the preliminary information from EFD the diagnosis process starts by inspecting the rules associated with the given subsystem, collecting those with a matching error-pattern. These rules become the hypothesis candidates for the remaining diagnosis.

Then the candidates are examined further by looking at their left-hand side, which is a Boolean expression composed of general conditions for the rule to become a selected diagnosis hypothesis. These conditions may involve references to any process variables or plant parameters, but typically they are expressions involving tests on the sign of gradients. The diagnosis continues with an examination of the left-hand side of all candidates, collecting all such references. Then a request for their current values is submitted to the EFD data base module, which is supposed to be responsible for looking up each process variable value and transferring the whole list of pairs ⟨process variable name, current value⟩ back to the diagnosis module.

Now everything that is needed to complete the diagnosis is available. The candidates with left-hand sides satisfied remain the resulting diagnosis hypotheses and are presented immediately in a special history pane in the user interface of the on-line diagnosis module, see Fig. 2.



```
12/05/90 14:41:11 DD-RUNNER-22
12/05/90 14:30:02 DD-RUNNER-21
12/05/90 14:29:58 DD-RUNNER-20
12/05/90 14:22:04 DD-RUNNER-19
12/05/90 13:13:21 DD-RUNNER-18
12/05/90 13:11:26 DD-RUNNER-17
```

```
DD-RUNNER-20
Completed at 12/05/90 14:29:58, with a total delay of 1 second since it
was triggered by an alarm from EFD.

RD50-LEAK-1 says: "Leak from feedwater line into heater RD51",
suggesting that you might want to consider the following advice:
"Isolate the RD51 heater" to rectify the situation.

The subsystem reported by EFD to be in error was RD50, and the values for
the detected deviations were:

D_RN63F001V = 0
D_RN63S001 = 0
D_RN63F001H = 0
D_RN62F001 = 0
D_RN62S001 = 0
D_RN61F001 = +
D_RN53S001 = 0
D_RN61S001 = 0

The first inspection of the knowledge base pointed to the possible rule
candidate(s): RD50-LEAK-1.

Some process variable values were requested by the diagnosis.  The values
returned were:

A_RD51L001_GRADIENT = 0.001123
A_RN61S001_GRADIENT = 0.400066
```

```
Wed 12 Dec 11:14:39 eyvind
```

Figure 2. MMI of DD Runner.

2.2.2. *The man–machine interfaces* Two user-interfaces have been developed, one for the operator and one for the process-expert or knowledge engineer. These interfaces are described in some detail below.

2.2.3. *The operator interface* The on-line diagnosis module, called DD Runner, has a simple user interface. Figure 2 shows a screen dump of it.

As explained above, a diagnosis is triggered by an alarm from the early detection system and performed independently of operator input. Completed diagnoses are collected in a stack, most recent on top, and appear in an overview window, called History Pane, of the operator interface. The operator may then inspect any of these diagnoses by clicking on the wanted item in this overview. The result of the selected diagnosis is then presented in the big middle pane of Fig. 2, called Current Diagnosis. Here the operator can find all data associated with that particular diagnosis. If a new diagnosis completes while the operator is busy inspecting a previous one, he is not forced to switch attention to the new one. Only the overview window is altered without operator control. Also the operator can mark a diagnosis as 'acknowledged' when he has finished his inspection of it. When acknowledged the diagnosis item in the overview window is shown in normal video, while unacknowledged diagnoses appear in reverse video. This way the operator is provided a means by which he can easily distinguish between new and old diagnosis information.

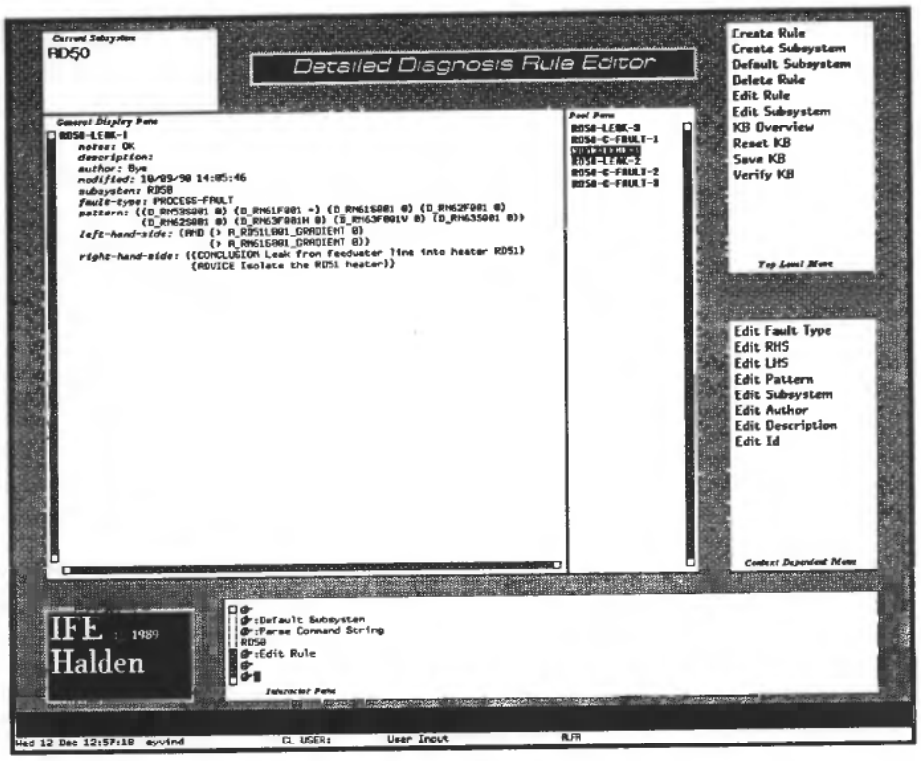
2.2.4. *The developer interface* The module called DD Rule Editor has a lot more complex user interface. Figure 3 is a screen dump of a typical display configuration.

This module is used by the process expert to edit the contents of the diagnosis knowledge base. As can be read from the menu pane in the upper right corner of Fig. 3, he can add, delete and update rules and plant subsystem characteristics. A complete functional description is outside the scope of this paper.

2.2.5. *Rules and subsystems* There are two main objects of manipulation interest to the user: rules and subsystems. Rules are the fundamental building blocks of the knowledge base. Subsystems are just collections of rules belonging to the same process area, or otherwise related in some way. Common rule characteristics are defined at subsystem level, while the details of each rule remain attached to the rule itself. The top-level menu contains the functions necessary for creating, deleting and modifying rules and subsystems. In addition the top-level menu has some auxiliary functions and storage of the knowledge base.

2.2.6. *MMI interaction principles* Besides the functions available from the menus, there are functions activated with direct mouse-clicks on so-called mouse-sensitive objects. These objects respond to the mouse-clicks by performing some associated action upon themselves. For instance, rule-objects presented in the pool pane (next to the menus) respond to mouse-clicks by displaying themselves in the display pane, while subsystem-objects respond to mouse-clicks by showing their associated rules in the pool pane.

2.2.7. *Verification and validation* Simple provisions have been made for doing automatic or manual consistency tests on the knowledge base. Manually, the developer may request to perform a complete go-through of all the rules in the KB by using the



```
RD50-LEAK-1
    notes: OK
    description:
    author: Bye
    modified: 10/09/90 14:05:46
    subsystem: RD50
    fault-type: PROCESS-FAULT
    pattern: ((D_RM53S001 0) (D_RM61F001 -) (D_RM61S001 0) (D_RM62F001 0)
              (D_RM62S001 0) (D_RM63F001H 0) (D_RM63F001V 0) (D_RM63S001 0))
    left-hand-side: (AND (> R_RD51L001_GRADIENT 0)
                         (> R_RM61S001_GRADIENT 0))
    right-hand-side: ((CONCLUSION Leak from feedwater line into heater RD51)
                      (ADVICE Isolate the RD51 heater))
```

```
:Default Subsystem
:Parse Command String
RD50
:Edit Rule
```

Figure 3. MMI of DD Rule Editor.

Verify KB option in the menu shown in Fig. 3. Each and every rule is then scrutinized by a testing procedure involving, among other things, verification of the validity of all external references in the condition part of a rule (Left-Hand side), other syntax checks on the condition part, and check for the validity of the error-pattern.

In addition to the manual testing, automatic rule checks are triggered whenever a rule is modified or created. A field called Notes is associated with every rule. If the rule passes the test criteria, it becomes immediately available to the on-line module, DD Runner. If, however, some field of the rule fails the tests, a reason for it is given in the Notes field and the rule becomes inaccessible to the on-line module.

This kind of meta-reasoning is easy to implement in LISP, because there is no artificial barrier between code and data. The LISP code itself can be handled as data—which is very useful in this connection—and vice versa, which we will see later is very handy in communication connections.

2.2.8. *The error pattern* A field called Pattern is also associated with rules and contains the characteristic fault pattern for the error hypothesis covered by the particular rule. For every pattern element there exists a deviation between the value of a process variable as obtained from a measurement and a calculation. The deviation can be ignorable (0), negative (−), positive (+), or significant, but of unknown direction (×).

It is up to EFD to decide what constitutes a significant deviation, DD only considers the qualitative value (i.e. the sign) of the deviations.

2.2.9. *Multiple conclusions or no conclusions* In our system there may perfectly well be more than one diagnosis hypothesis of a particular fault. This could be due to multiple simultaneous unrelated faults, or insufficient number of sensors to locate the faulty component. Also, we may end up with no hypothesis at all because of an incomplete or inaccurate diagnosis knowledge base. However, the basic information about fault location and detected deviations remains valid and is valuable in itself when the operator is seeking the cause of the fault. This makes the system robust to unexpected and low-probability events.

2.2.10. *Uncertainty* In the current version of our system we have not included any remedies for doing reasoning on incomplete or uncertain information. The rules themselves are built on absolute knowledge (e.g. physical laws) and is certain enough, but the input data to a diagnosis could be subject to further inspection. For instance, the degree of deviation between a measured and calculated process variable value could be reflected in the faith we assign to a particular hypothesis dependent on this deviation. Currently, we just decide whether there is or is not a significant deviation, and if it is, we only look at the sign of it when comparing it to the error pattern of the rules. A possible extension to this approach could be to employ fuzzy logics for deciding the significance of deviations.

A provision for uncertainty handling could also be sneaked in using the current framework through extensions of the condition parts (Left-Hand sides), since there is no inherent restriction on the content of a rule LHS as long as it can be treated as a LISP expression returning true (not nil) or false (nil).

2.2.11. *Fault categories* We have distinguished between three different types of faults: Measurement Faults, Control Faults and Process Faults. A measurement fault involves sensor malfunction, a control fault implies an error in the control system, e.g. a dead valve controller, and a process fault amounts to everything else, e.g. tank leaks, stuck valves and broken pipes.

This distinction only serves as a classification of hypotheses—there is no intricate logic behind this scheme employed in any of the reasoning, but it could convey useful information to the operator about the nature of the error he is faced with.

2.2.12. *Real-time diagnosis* Expert systems for diagnosis is by far the most common in the field of artificial intelligence applications. However, almost all of them rely on a static set of inputs and the analysis is often done off-line. While suitable and acceptable in maintenance or other non time-critical situations, this approach fails when the analysis is focused on the early detection and diagnosis of faults in a rapidly changing process environment.

We have taken this non-monotonicity into account by using trend information (e.g. gradients) and instantaneous values for process variables during a particular diagnosis. Also, each diagnosis is an independent, time-stamped object completing within seconds from the point of invocation. Parallel diagnosis invocations may exist, facilitating the simultaneous analysis of multiple faults.

2.2.13. *Explanation* All relevant data used in a particular diagnosis is readily available to the operator. By means of a simple click-and-display facility he can have a look at the specific process variable used in the diagnosis.

Theoretically, he should then be able to trace the course of events during the diagnosis. As we have not provided support for a complete backtrace function in the current implementation, we suspect however, that it would be too cumbersome a project for the operator to carry out in practice. Before transferring the diagnosis module to a real plant we will have to put more efforts into the design of a good explanation facility.

2.2.14. *Example diagnosis* Using the high-pressure preheaters in the feedwater process selection of a pressurized water reactor (PWR) as the subject for our example, we can show how a diagnosis is made as seen from a qualitative point of view. It should not be necessary to know the details of this particular process section to see how the diagnosis is performed. The example illustrates how one can discriminate between faults which give the same error pattern. Consider an error pattern, *ep*, for subsystem RD10† where:

D_RN13S001 = 0
D_RN21S001 = 0
D_RN22S001 – 0
D_RN23S001 = 0
D_RN21F001 = 0
D_RN22F001 = 0
D_RN23F001 = +

Informally, we can then make the following statement: If the error pattern for RD10 is (000000+), i.e. identical to *ep* above, indicating unexpectedly high outlet condensate flow for the high-pressure pre-heater, we may have the following 4 fault hypothesis candidates

(1) flow-sensor failure for the high-pressure outlet condensate
(2) temperature-sensor failure for the high-pressure outlet condensate
(3) temperature-sensor failure for the inlet feedwater
(4) process failure with leakage of feedwater into preheater tank

Next, we check the values of the process measurements of interest for the candidates at hand. In this case one non-zero gradient were fetched—a positive value for a level gradient:

A_RD13L001_GRADIENT = 0.2

When there is a sensor failure (1, 2 or 3) the gradient of one particular instrument has to dominate, while in case of a leakage (4) increased water-level in the tank should be observed. This makes it possible to discriminate between process failures and measurement faults which is generally difficult in most cases. In this case it turned out to be a leakage.

† Note on the notation used in this example: The syntax 'D_*' is used to indicate a value that is the (qualitative) difference between a measured and a calculated process variable. The syntax 'A_*' refers to the direct measurement of a process variable. The process variables used in this example are named according to the conventions used in the PWR simulator. E.g. 'RN13S001' indicates that this is a variable within the 'RN' subsystem (the High-Pressure Preheaters), the '13' indicates the (component level) area with this subsystem, the 'S' indicates that this particular sensor measures a valve position, and the '001' is simply a sequence number for this sensor, to keep it distinct from similar sensors in the same area.

2.3. *Where is the intelligence?*

We would like to think of EFD and DD as intelligent programs, but it may not be quite clear exactly where the intelligence is. DD without EFD is not much more than a shallow, case–symptom based diagnostic tool, while stand-alone EFD is merely a plain detection system. But EFD operating together with DD is definitely approaching something which could be called intelligent by today's computer program standards.

Due to the fault confinement provided by EFD, diagnosis is restricted to the problem area identified by EFD. This means that the diagnosis becomes both simple and efficient. Also, it becomes accurate because deviations are discovered by EFD at the earliest possible time, before consequence alarms begin to play a role in the game.

In sum, the EFD/DD combination offers automatic, high-level analysis to the operator through the use of model-based detection and diagnosis providing a synthesis of detailed process knowledge and qualitative reasoning techniques.

3. Technological foundation

This part of the paper describes in more detail how EFD and DD are implemented and how they are engineered into an integrated environment of cooperating software modules.

3.1. *Software framework*

EFD and DD play the roles of components within a larger framework for process supervision and control, see Bologna (1991). Currently, EFD and DD operate as independent agents communicating with each other through message passing (TCP/IP based), a common database (Sybase) and through remote procedure calling protocols (RPC). Figure 4 shows how the main components of the on-line system are (conceptually) interconnected.

Both EFD and DD maintain their own man–machine interfaces (MMIs), but also volunteer information to the overall coordinating agent, the Intelligent Coordinator,

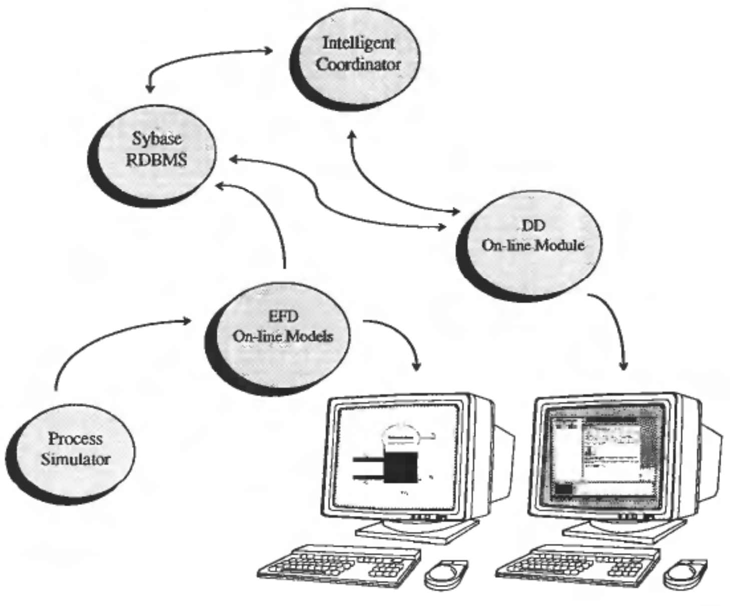


Figure 4. Main components in on-line system (ISACS configuration).

which in turn controls how this information should propagate through to other displays and to other software systems within the framework.

3.2. *Software components*

The EFD and DD software incorporate the application of a number of different underlying software components. The following discuss the role of each individual component and the preliminary experiences gained from employing them in our context.

3.2.1. *Sybase* The Sybase relational database is storing process data and individual component system data of common interest to parties within the ISACS framework. EFD is among the software components which periodically feed the Sybase database server with update requests of process variables, both their measured and calculated values. Since there is no direct access from the EFD software currently running on an ND minicomputer to the databases server module running on a Hewlett-Packard Unix workstation, EFD sends its request using the only communication software library common to both platforms, namely TCP/IP. The Sybase server operates surprisingly fast and does not seem to pose problems as a bottleneck in this connection. The communication system, however, gives rise to some more concern. TCP/IP is a comprehensive, connection-oriented protocol with significant overhead and with varying implementation quality. On occasions we have experienced performance problems with this connection largely suspected to be due to TCP/IP implementation anomalies.

3.2.2. *G2* The G2 Real-Time Expert System Shell from Gensym Corp. has been employed to implement the Intelligent Coordinator module responsible for controlling and coordinating the activities among the different operator support systems within the ISACS framework. The coordinator has a variety of different tasks to take care of. Relevant to our discussion here is that the coordinator monitors EFD operation and decides when DD should be triggered with data from an alarm issued by EFD. Furthermore, it compares the output from a diagnosis provided by DD with informations obtained elsewhere—in our case a different diagnosis system.

Of course, EFD and DD may perfectly well run outside the ISACS context with no coordinating module in between, with EFD submitting its alarm reports directly to DD.

G2 has proved to be well suited for implementing coordination tasks. Only minor problems have been encountered due to its limited flexibility in major software reorganization and its lack of meta-reasoning capabilities.

3.2.3. *Symbolics Common LISP* To implement the symbolic manipulation and qualitative reasoning so heavily employed in DD we have utilized the symbolic processing power of a Symbolics LISP workstation running an exceptionally flexible and powerful implementation of Common LISP. The kind of symbolic reasoning and meta reasoning used in DD would be extremely hard to implement within less flexible software environments. For instance, we have taken advantage of the meta-reasoning capabilities of LISP to design an application independent communication paradigm. The encoding and decoding of messages are left to the built-in LISP reader and printer functions, which automatically convert between the internal and external representations of data objects.

Also, we have employed meta-reasoning to make consistency checking and simple rule validation. On-line testing, fine-tuning and modification of the knowledge base is yet another attractive consequence of the inherent flexibility of LISP. The important point is that the implementation tool should allow and encourage the development of software which is powerful enough to put the developer in a position where he can not only create intelligent programs, but also where he can easily extend his program to make useful meta-reasoning upon its own program code.

The New Flavors object-oriented extension to Symbolics Common LISP have been used to create clean and simple data structures and class hierarchies, but not much more. As new standards for object-oriented programming are emerging we intend to migrate the current Flavors-based software to CLOS—the Common LISP Object System—now becoming the generally accepted standard for object-oriented programming in the LISP community. We foresee no major difficulties in this process.

The Symbolics platform has proved to be convenient for real-time, process-oriented applications. Independent processes are easily created, and they share the same data space by default. No extra programming efforts are needed to invent and maintain intricate schemes for dealing with interprocess communication.

3.2.4. *X11* The X Window System, version 11, developed by the MIT X Consortium is the underlying ingredient in all our user-interfaces. Because we are running our applications in a heterogeneous hardware and software environment, and X11 is a platform independent windowing system, we have taken advantage of the system to create a common user interface for most of the software components involved in ISACS. In the case of DD, which originally was designed and developed for running under the platform-specific window system on the Symbolics LISP workstation, X11 is used to export the Symbolics screen out to the Unix workstations running the end-user interfaces. No modification to the DD code was necessary to accomplish this feat.

3.2.5. *TCP/IP* Briefly mentioned above, TCP/IP is our main software and platform independent glue between most of the components involved in ISACS. TCP/IP is the Transport Control Protocol based on the Internet Protocol originally defined by DARPA (Defense Advanced Research Projects Agency), an American government organization. It has become a de facto communication standard endorsed by all major computer vendors. TCP/IP is still the only practical means by which two pieces of heterogeneous hardware can be connected in an application independent way. As mentioned above, TCP/IP is a large, comprehensive protocol with significant overhead related to error-checking and connection management. Originally developed to support connections between highly unreliable hardware components, the TCP/IP software make the communicating applications spend much time checking the validity of each other's data. With the kind of reliable, high-speed local area networks in use today, this seems like a waste of time. Also, the (telephone) concept of connection establishment, synchronization, maintenance, and hang-up seems archaic in the world of computers where the traffic is typically of a 'bursty', short-term nature.

3.2.6. *RPC* The protocol for Remote Procedure Calls, RPC, as defined in the de facto standard developed by Sun Microsystems, is replacing TCP/IP communication whenever possible. The problem is that not all platforms run compatible versions of this extremely efficient and simple communication software. The concept of remotely callable functions appears also more attractive to software developers not wanting to spend too much time figuring out how to synchronize and maintain an explicit connection between two cooperating agents.

3.2.7. *EFFORT* All EFD modules are currently implemented in EFFORT, for EFD FORTRAN EFFORT is a locally developed extension to standard F77 syntax and semantics, much in the spirit of the Unix utility called Ratfor (Rational Fortran). In addition there is an EFD Database Generator making it possible to have pseudo object-oriented F77 data structures, which is useful when one has to control the huge amount of internal variables in a process model developed in FORTRAN.

EFFORT offers a range of convenience functions, where the so-called *forall* construct has proved most useful in EFD connection. *forall* makes it possible to write code that automatically iterates over a set of FORTRAN data base variables. If EFFORT is meta-FORTRAN, then *forall* is a kind of meta-EFFORT. Typically *forall* is used to generate EFFORT code.

3.3. *Synchronization issues*

The Sybase database server is getting current process variable updates from EFD before clients are allowed access to the data. Read requests are supposed to queue up while EFD runs its update process. This scheme is constructed to ensure consistent use of process variable values in other applications like DD. It is a bit early to conclude anything about the adequacy of this scheme, but all we can say so far, is that it seems to work well.

At the time of writing, EFD runs in the ISACS environment synchronized with the process simulator on a 3 second cycle. Sybase updates occur every third EFD cycle, i.e. about each 9th second with about 400 process variable updates.

4. Conclusions

The EFD system based on decoupled reference models has been implemented and tested in a variety of transients on a PWR simulator in our laboratory. EFD has been installed at the Loviisa Nuclear Power Plant in Finland, and proved its usefulness as the far most sensitive on-line tool to discover small, slowly developing leakages. The DD module has been developed to make detailed assessment of the fault situation within a limited process section of the PWR. DD has only recently become an operational prototype and is currently running in our laboratory within the Integrated Surveillance and Control System (ISACS) environment.

EFD/DD combines the power of quantitative and symbolic reasoning into a coherent unit capable of fast, accurate detection and diagnosis of process anomalies. Integration tests with ISACS, has shown that EFD and DD can operate as units within a larger process control environment.

References

Bologna, S., Berg, Ø., Haugset, K. and Kvalem, J. (1991). *Architectural Foundations, Concepts, and Methods Behind ISACS—A Real-Time Intelligent System for Critical Applications*, presented at the 2nd IFIP Conference, Tuscon, Arizona, USA, February, 1991.

Holmstrøm, C. B., Nelson, R. W., Berg, Ø. and Kaarstad, T. (1989). *The First Experimental Evaluation of DISKET: The DIagnosis System using Knowledge Engineering Techniques*, Work Report, HWR-242, August 1989.

NESS, E., BERG, Ø. and SØRENSSEN, A. (1989). *Early Detection and Diagnosis of Plant Anomalies Using Parallel Simulation and Knowledge Engineering Techniques*, in VTT Symposium 109, Artificial Intelligence in Nuclear Power Plants, vol. 1, pp. 197–219, presented at the IAEA/IWG NPPCI Specialists' Meeting, Helsinki, Finland, 1989.

PATTON, F. and PATTON, C. (1989). *Fault Detection in Dynamics Systems, Theory and Applications*, Prentice-Hall.

SØRENSSEN, A. (1990a). *Early Fault Detection at the Loviisa Nuclear Power Plant by Simulation methods*, presented at European Simulation Multiconference, Nuremberg, Germany, June 1990.

SØRENSSEN, A. (1990b). *An Early Fault Detection System Running on a Live Power Plant*, Work Report, HWR-261, January 1990.