

https://dspace.org/

Fault recovery in process control

Horn, Timothy Andrew

2015-02-05
http://hdl.handle.net/10539/16894

Downloaded from a DSpace repository. https://dspace.org/

processors. All processors can perform fault detection, diagnosis and recovery with duplicated software, eg. TMR systems. Also, by structuring the Inputs and Outputs, (I/O), of the system, the distributed control of all sub-systems can produce better fault-tolerance and cross-checking of other sub-systems.

There are three classes of errors in computer systems, [5], ie. internal, external and pervasive errors. These determine how a computer system reacts to them.

Internal errors or faults are handled by the process or SRU in which the error occurs, whereas external errors require that other processes or sub-systems help in handling the fault, eg. in TMR systems, a fault is typically handled by the two healthy processors which isolate the third faulty process, or by copying " good " data to the faulty processor.

A pervasive error or fault is one which affects other processes, causing complete maloperation of the control system. This is done by passing corrupted data to the other processes. Recovery from these types of errors is typically performed by restart vectors or safe shutdown.

Another concept of exception handling for design of fault-tolerant software is evident, [13]. An exception can be similar to all internal, external and pervasive errors as detailed above.

Internal and external errors are dealt with by anticipated and programmed exception handlers, which allow for or expect an error or fault, and this is written into the software. This programmed exception handling takes the form of masking, consistent state recovery, and signalling, [13]. Masking and state restoration were discussed in section 3.1 . Signalling is a simpler concept which detects and reports and/or displays a fault condition for the operator to correct the fault.

processors. All processors can perform fault detection, diagnosis and recovery with duplicated software, eg. TMR systems. Also, by structuring the Inputs and Outputs, (I/O), of the system, the distributed control of all sub-systems can produce better fault-tolerance and cross-checking of other sub-systems.

There are three classes of errors in computer systems, [5], ie. internal, external and pervasive errors. These determine how a computer system reacts to them.

Internal errors or faults are handled by the process or SRU in which the error occurs, whereas external errors require that other processes or sub-systems help in handling the fault, eg. in TMR systems, a fault is typically handled by the two healthy processors which isolate the third faulty process, or by copying " good " data to the faulty processor.

A pervasive error or fault is one which affects other processes, causing complete maloperation of the control system. This is done by passing corrupted data to the other processes. Recovery from these types of errors is typically performed by restart vectors or safe shutdown.

Another concept of exception handling for design of fault-tolerant software is evident, [13]. An exception can be similar to all internal, external and pervasive errors as detailed above.

Internal and external errors are dealt with by anticipated and programmed exception handlers, which allow for or expect an error or fault, and this is written into the software. This programmed exception handling takes the form of masking, consistent state recovery, and signalling, [13]. Masking and state restoration were discussed in section 3.1 . Signalling is a simpler concept which detects and reports and/or displays a fault condition for the operator to correct the fault.

Pervasive errors could typically arise from design faults. These design fault exceptions are only dealt with when detected during debugging, commissioning and at test intervals of the plant hardware and software. Also faults which are not catered for in the exception handlers can occur, and give rise to design fault exceptions.

In simple sequential machines, error correcting can be effected using error correcting (n,k) linear codes instead of Reed-Muller code and Perfect Hamming codes, [11]. This is less complex and easier to implement, but should only be restricted to large database systems, where coding systems similar to simple parity checking are vital.

To enable databases to be fault-tolerant, or error-free, data structure correction principles can be implemented, [14]. Data partitioning for separate processes or SRU's should be performed, and each process must have it's database protected from corruption by others, ie. definite memory boundaries must be set up. Common data is then copied to other processes and SRU's so that if corruption occurs, good data can be recopied from the source data.

Fault dictionaries can be used to detect errors and initiate fault recovery techniques. Iterative tests, such as random guesses or pattern tests, should be performed to detect any faulty data.

In the handling of faults in computer systems, the removal of faults in non real-time and real-time systems are similar to that required in industrial processes.

Adapting this, (from [5]), to recover from faults in non real-time processes involves substitution of control information and retrying, ie. using backward recovery, recovery blocks and/or programs. In multi-tasking systems such as distributed control, the Domino effect must be avoided.

For fault recovery in real-time systems substitution is done by reconfiguring the SRU's in the system for graceful degradation with active or standby redundancy, and using forward recovery. Another approach is to skip frames and use old inputs and retry, but persistent errors requires that proper forward recovery must be used.

Fault-tolerant software should be structured and modular, allowing distributed fault recovery and back-up systems. Two main software approaches are evident , [27], viz. recovery blocks and N-version programming. N-version programming has been used in the SIFT computer system, and involves fault-tolerance achieved by using different fault-checking algorithms, and sometimes implemented on differing makes of microprocessor, (see Appendix C.).

3.3 Fault Recovery Evaluation

The two factors particular to fault recovery techniques, viz. recovery strategies and fault tolerance have been discussed.

Recovery strategies can be applied in a simplistic or complex way. In general, simple recovery can be seen as normal control strategies, whereas complex recovery involves more than one recovery principle.

Redundancy techniques require complex interconnection software techniques for the control of duplicated sub-systems to create the effect of bumpless switchover.

State restoration techniques cover the more inherent software principles underlying the software techniques to control both the plant and computer hardware for bumpless switchover to redundant sub-systems.

Fault tolerance is the tying together of both the redundancy and state restoration principles. The fault tolerance involves detecting the faults, deciding on the redundant sub-system involved, and performing redundancy and state restoration techniques to dynamically reconfigure the overall control system and plant sub-systems to create bumpless control.

	SIMPLE FAULT RECOVERY
Digital	eg. close input valve when level high detected
Analogue	eg. adaptive control of hot water control valve with PID control and set point and feedback
	COMPLEX FAULT RECOVERY
Digital & Analogue combinations	eg. adaptive control (PID) with emergency shutoff when very high temperature reached
Back-up systems, Redundancy	eg. switchover to standby pump if operating pump fails
State restoration	eg. safe shutdown, abort sequence and start from beginning

TABLE 3.1 SIMPLE AND COMPLEX FAULT RECOVERY STRATEGIES

4. SYSTEM DESIGN CONCEPTS.

4.0 Introduction

Fault detection and diagnosis systems must be designed with regard to the following principles, [1] :-

Fault detection involves the detection of incipient faults and the early detection of an abnormal state occurrence.

Fault diagnosis involves real-time determination of the cause of the malfunction and predicting a trend of the process to abnormality. From this, the most effective fault strategy must be selected. After the fault has been rectified, post-failure diagnosis should be done to determine the cause of the failure and possible future safeguards that can be introduced to prevent it happening again.

To determine how fault recovery systems are designed, the following will be covered in this section :-

- hardware considerations;
- software considerations; and
- overall system implementation.

4.1 Hardware Considerations

There are a number of ways in which to implement control systems, ie. :-

(1) Dedicated instumentation with relay switching. These systems are only implemented in small systems, or in critical sections of the plants, eg. safety and backup. Any changes to the operating principle of the system is difficult to implement.

(2) Programmable Logic Controllers, (PLC's). These tend to be used where flexibility of logic usually performed by relay logic is required. Most PLC's can now perform both digital and analogue functions, allowing far superior control than " old-school " relay systems, eg. PID control, speed ramping, delay functions, etc..

(3) Large PLC's and computer control systems. These form the higher part of process control for large plants, fast and powerful to perform many functions quickly. These typically include graphic video systems, and data logging and capture for later trend analysis.

A control system can be divided into levels, [3], with each level encompassing the entire process, and an operator interface, as well as unique or shared hardware and software. This will be adapted to perform the fault recovery function as well as the normal control functions.

Level 0 is the process itself. Process equipment must be designed with appropriate safety margins to minimise the consequences of control failure at any other level.

Level 1 is the level of hard-wired safety systems. Interlocks, sensors, and limit-sensing devices communicate with the operator by means of indicators, switches and annunciators. Characteristics of this level should include simplicity, redundancy and independence of functions, power supplies, cabling, etc., to ensure that the process maintains a safe condition regardless of failures and errors at a higher level.

Figure 4.1 shows the general relationship between control levels. Each succesive level performs increased plant-orientated and eventually company-orientated functions. The higher level may send instructions, eg. alarm limits, to the lower one. If the higher level becomes unavailable, the lower one should operate with its current instructions until changed by the operator. The lower level must never depend on the higher level for the performance of it's functions. In general, a higher level should use lower levels for the outputting of it's control functions.

Level 2 has safety systems as well, but allows less independence of function while permitting greater use of computing ability, ie. safety features in software instead of hard-wired devices. As an example, at the level 1, a reaction may be shut down if its temperature exceeds a certain limit value. In level 2, the rate of temperature rise may be used to stop the process.

Level 3 is the lowest level at which plant may be controlled to it's intended purpose. Manipulation of plant I/O is achieved through "manual" operations by operators, eg. open valve A. Safety functions incorporated prevent the operator from placing the system in a dangerous condition.

Level 4 is where automatic control is available. Control at this level, by computer control or by the operator, is done by setpoints, eg. control tank level between B and C. The fault handling at this level would include commands to the lower levels to start or stop pumps, or open or close valves at the limit values if the control at level 3 fails to do this correctly.

Level 5 is where co-ordinated control is affected, and loops are combined, eg. cascade, feedforward and bang-bang control.

Level 6 is where sequential control is added to the control system, eg. open valve A, then control tank level between B and C.

Level 7 is the limit of direct process control, eg. run

Above these eight levels, are the management levels, which are scheduling of batch sequences, (Level 8), maintenance and establishment of recipes, (Level 9), and corporate demands for production and profit, (Level 10 and above). These do not necessarily affect the fault recovery system design philosophies.

The level system as discussed above lends itself to multi-processor control systems, and ultimately to distributed computer control systems. The level system is heirarchical in nature, a particular feature conducive to distributed computer control systems, (DCCS). DCCS are dealt with in great detail in [4]. DCCS offer increased reliability, availability and maintainability of the process being controlled, which thus offers increased fault handling capabilities.

Distributed control imposes various functional requirements on the architecture of the distributed system, [16]. The most important of these are modularity, expandability and dependability. Dependability is part of the fault-tolerance requirements for reliability, availability and maintainability.

To accomplish distributed control, the process plant must be analysed and partitioned into it's functional sub-systems. These must then be allocated independent function computer controllers, in various forms to cover level 1 to level 3 of the above level system, ie. dedicated instrumentation and possibly up to "medium" size PLC's. The higher levels can then be implemented with "large" PLC's and mini-computers, and even up to mainframes for the levels 10 and above. All dedicated instruments, PLC's and computers must be interconnected to complete the DCCS. The system should also include facilities for expanding for completeness of the DCCS. Figure 4.2 shows a simple structure for the development of a complete level-based or heirarchical DCCS.

How is the hardware designed to be fault-tolerant ??

Hardware fault-tolerance, [2], must incorporate error detection by hardware in real-time and at regular test intervals, by dedicated instrumentation and by software. The dedicated hardware involves limit switches, sensors, etc. as described in level 1. The software required involves watchdog timing and regular testing of all I/O hardware under operating system testing sequences. Redundancy of function and communication links at all levels should be included, for better fault-tolerance. Real-time hardware recovery is required, eg. limit switch interlocks a particular control function, and software recovery, eg. tripping PLC watchdog interlock to control outputs to put plant into safe state. The fault must then be isolated, and this can involve switching to standby plant equipment and redundant control equipment. The fault can then be repaired and the system restored to it's fully operational state.

The communication system of a DCCS must also be considered. A DCCS relies on a particular communication architecture, based on connection strategies, eg. ring, star, multi-drop and even combinations of these, [4, 5, 16, 19, 32, 33]. A typical implementation would also incorporate redundancy into the communication network, eg. the ISABELLE control system for control of a electron beam accelerator, [35, 36]. The communications and data transfer for ISABELLE is contained on three redundant process data highway rings, and a star control network. Each control computer is triplicated using Triplicated Modular Redundancy (TMR) configuration into a Can't Fail system, [35].

4.2 Software Considerations

Software is implemented differently in the three basic types of control technologies given above, viz. relay and dedicated instrumentation, Programmable Logic Controllers (PLC's), and computer systems.

The smaller computer systems, implemented in the dedicated instruments with relay switching, will typically use small microprocessors with assembled programs stored in EPROM's. Standard features are usually provided, with very little flexibility, and changes are not easily affected with assembled programs in EPROM's.

Programmable Logic Controllers vary considerably. The most common software used is ladder diagram programming, which resembles relay wiring diagrams, as PLC's were first developed as direct relay replacement systems. Other methods available are logic lists and block diagrams. The logic lists are similar to mnemonic code of assembly languages, and block diagrams are drawn up and connected using a video display and software " wiring ". These are also inflexible, but the methods available are typical of how process control systems are conceptualised and developed.

The most sophisticated PLC's are approaching process control computers, with colour graphic video displays, printing and serial communications to other PLC's and computers.

Large process control computers are highly sophisticated, and capable of quality colour graphics, printing, and fast and reliable communications.

High level languages for process control software are usually limited, [2], as real-time interrupts are not easily handled, and interface to input and output devices is difficult. These are usually solved with assembly routines and extensions to the standard languages. The most common high level languages used for process control are tabulated in Table 4.1 , and the differences are given.

With the current trend towards the multi-processor and distributed systems, and for communication between all devices, a network or Local Area Network, (LAN), must be developed, [4, 5, 16, 19, 32, 33]. For process control, this must be a real-time network system, and the communication software must be separated from the other process controlling software. This is usually done with specially designed interface computers. The protocols and the addressing or naming of the nodes of the network must also be considered in the developing of the communication software.

Fault-tolerance of the network should also be considered. This is usually handled by using redundancy in the communication network, ie. duplicated communication hardware and software, and redundant physical links, eg. ISABELLE, [35, 36].

The SIFT computer system, [5, 15], deals with the implementation of fault-tolerance in software and not in hardware. It also uses N-version programming, ie. all functions are redundantly programmed with completely independent tasks in different languages and using different algorithms, and different hardware, with compatible communications. This protects against inherent software errors not being found by adhering to a particular language,algorithm or microprocessor type. As an example, [6], dual dissimilar processes have been implemented in Aircraft control.

With the combination of redundancy and using N-version programming, (SIFT, [15]), and self-stabilising programs, [17], the majority of software bugs and faults can be found quickly. This then protects the control system against internally generated errors, but not external errors, Protection against external errors , [12], is done by defining states for all variable values. This method finds the set of contaminable variables for each possible fault input, and the complimentary non-contaminable set. Only the contaminable set need to be stored in rollback and retry algorithms for recovery from errors. Also, the redundant computer principle requires that each computer have a duplicate copy of the software, ie. each computer can function independently from the others.

Each input is considered error producing, and all variables involving each input becomes part of a list of contaminable variables.

Error checks are then inserted in software, and if failed, error recovery is performed. The prime concern of real-time control systems is the chance of an external error propogating to a dangerous output condition, This must be protected by inserting check phases and exception handling mechanisms into the phase diagraph, (sequence or state diagram), before each dangerous phase to stop errors.

Recovery is typically performed by rollback techniques, and additional phases are inserted to cause saving of contaminable variables at required times. For real-time systems, the only rollback facility is to go to the initial phase and restart, or forward to safe shutdown, else a Domino effect may occur, which eventually ends at the initial phase anyway. A construct for backward and forward recovery software is given in Appendix C.

Author Horn Timothy Andrew
Name of thesis Fault Recovery In Process Control. 1985

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.