# FAULT IDENTIFICATION IN ACCELERATOR CONTROL

Philip Duval, Ursula Laaströer, Rüdiger Schmitz, DESY, Hamburg, Germany

*Abstract*

Accelerator control differs from, say, nuclear power control in that the control parameters are frequently changing and evolving and are often themselves under study. Provisional control applications (written in haste) are often used because they are urgently needed to study an emerging problem. Double (triple?) redundancy is infrequently applied to the control points and the control system is often dependent on the Ethernet and is therefore partly administered by the site's IT division. Accelerator operators desire the same reliability found in nuclear power control but are nonetheless often confronted with control problems. For instance, communication with a front end computer (FEC) might suddenly suffer frequent timeouts. Is it a network problem, a software problem, a hardware problem, or another kind of front-end or synchronization problem? Looking in the wrong place is likely to increase frustration and delay operation.

In this paper we report on variable tools and techniques for identifying control system faults at their different levels (site infrastructure, control infrastructure, applications, incomplete operator training) and we discuss various remedies for correcting them and strategies for avoiding them.

## INTRODUCTION

Our experience with faults arises from the control systems for the accelerator chain at DESY [1]. The main characteristics of these systems in this context are listed in brief below:

- Multiple operating systems in use on front-end computers but exclusive usage of Microsoft Windows (NT XP) at the operator consoles. Table 1 gives a summary of the operating systems involved and their contribution to the total number of computers in the control system
- Integration of two control-system architectures: one based on TINE [2] the other mainly using Novell's IPX broadcasts [1]
- There are several hundred applications, mainly written in Visual Basic (3.0 and 6.0) but C and C++ are used as well.

Table 1: Usage of Operating systems

| **Operating system** | **Number of computers** |
|---|---|
| Windows XP | 64 |
| Windows NT | 197 |
| Windows 95 | 10 |
| Windows 3.11 | 18 |
| DOS | 14 |
| Linux | 63 |
| Solaris | 3 |
| HP UX | 5 |
| VxWorks | 15 |
| LynxOS | 1 |
| **Total** | **390** |

## FAULTS AND FAULT HANDLING

A few points concerning control system faults should be noted here. First, faults appear with different signatures and are at first glance statistically distributed in time. Second, different fault-signatures may in fact have a common cause. And third, from the control-room-operator's point of view a "fault" is deemed to have occurred whenever the machine cannot be operated normally.

Most faults originate in application programs written by the controls people.. All too often, changes of a "running system" have to be urgently made in order to rectify an emerging problem in the accelerator hardware, or to address the dire needs of the operators. Software faults of this nature are bound to happen and don't necessarily reflect "shoddy" programming, but are rather due to the effects of unforeseen degrees of freedom in the enormous "phase-space" of accelerator control parameters. To a large extent, standardized automated testing mechanisms, which might catch some of these faults 'waiting to happen', are missing as is an acceptable amount of test-time at the accelerator facility itself.

### *Detection*

The principal fault 'detectors' are the operators. If a program shows a malfunction they register a fault, one way or another, either by inserting a logbook entry or email or verbal notification to those responsible.

From the early days of PCs controls in the DESY accelerator chain where 16-bit Windows was in vogue to the present, where Windows NT/XP is in common use, the operators have obligingly recorded most manners of server faults in a special logbook. This logbook is primarily devoted to the Windows servers in operation within the IPX-based control system of the pre-accelerator chain, and therefore only infrequently records faults pertaining to other servers. The errors recorded typically include but are not limited to

- Application Runtime errors
- Access faults.
- Operating System errors.
- Network errors

- Disk errors
- Hardware errors

Even though not all faults are guaranteed to have been documented, the number of log entries gives a rough indication where action is required. Table 2 shows that the number of faults per day is nearly independent of the machine's schedule (roughly constant across shutdown periods) and of a server's operating systems, to the extent the log entries concern only Windows 3.11, NT, and XP servers. However the results shown below are nevertheless instructive.

Table 2: Faults reported by operators over 116 Device-Server-PCs running Windows 3.11/NT/ XP

| Year | Number of faults | Faults/Day |
|---|---|---|
| 2001 | 227 | 0.63 |
| 2002 | 190 | 0.52 |
| 2003 | 145 | 0.40 |
| 2004 | 188 | 0.52 |

In order to automate fault-detection we have installed several systematic procedures, some of which are still in the commissioning phase. Still missing, for instance is an acceptable notification mechanism, as is a central control-system fault help-desk.

The following control-system services are checked automatically. These checks help to reconstruct the history of fault propagation and at the moment provide a tool to manually detect faults and ascertain their causes:

- **Connectivity to Device Server**: An IP checker regularly pings all addresses in all control-subnets (18) and provides information on each IP-address, such as: ping reply ok, date ping switched to no reply, date switched to reply, days not seen, and so on. Likewise, an IPX checker checks on the availability of each of the 118 IPX device-server-PCs and generates list of downtimes.
- **Dynamic-Mac-Address-Tables** of our controls-network-switches read out and conversion into hostnames and IP-Addresses.
- **Fileserver Check** Tool tests the availability of important control system file servers.
- **Application Watchdogs** provide an automatic restart of control-servers-applications and logging of application state changes.
- **Alarm System:** shows current or archived alarms detected in server-processes or central alarm servers.
- **Statistics System:** regularly acquires statistics counters from device servers, such as busy time, restarts, number of timeouts, and so on.

## *Identification*

Typically, when a fault is detected it is part of a fault "tree," where finding all the causes which lead to the "symptom" require good logging and archive systems. To this end, we also make use of several tools which are used "after-the-fact". These are listed below.

- **Network-Overview** pings the critical elements in the controls network topology, thereby quickly isolating the source of a network failure.
- **Logfiles** generated either by the control system kernel or by the control system application are usually very instructive in isolating a control system fault. Most helpful is a central tool which can scan all relevant log entries over a particular time span. Logfiles generated by the operating system (e.g. Windows Event-Log or syslog).
- **Archives** should contain not only machine parameters but hardware settings. Archive viewers should be able to correlate any entries stored either locally at the device server or centrally. Furthermore, post-mortem or "event-driven" archives are indispensable when trying to find the cause of a sudden beam loss. This generally includes transient recorders and other sorts of hardware triggered fast data acquisition systems.
- **FEC-remote-control** operating system independent tool, to observe and control TINE-processes.
- **FEC-Statistic** displays data from the statistics system: CPU-Load, restarts, Network-Timeouts etc.

The originally observed fault will leave its signature within the corresponding log files and archives. This signature servers as the fault's identification.

## *Isolation*

By "identifying" the fault, we have not necessarily found its cause. "Whenever I do 'A' then 'B' happens" might suggest that we avoid doing "A" if we don't want "B" to happen, but does not by itself explain why "B" happened. This frequently requires further investigation where we need to isolate the fault. In other words, What subset of 'A' makes 'B' happen? Can I remove all extraneous parameters from consideration? Can I generate the fault in a simplified test environment?

## *Reproduction*

If we have successfully isolated the fault, then we should be able to reproduce it ad infinitum. More importantly, we should now have a clear understanding of its underlying causes and be able to repair it.

## *Classification*

It should be clear that control system faults have to be eliminated with all our best efforts. The loss of beam-time due to such faults has to be reduced to a minimum.

From our experience over the last 8 years we can categorize the causes of faults as follows:

- Network-Problems: unrecognised slow increase of network load with statistical peaks cause unacceptable loss of datagrams, leading to faults in our system which relies in IPX-datagrams.
- Computer Hardware, bad memory or disks of a certain delivery caused a lot of work and trouble.
- Operating System Problem, the DLL-hell of Windows makes it very difficult to distinguish between application- or system-program error. New DLLs distributed with e.g. security- or program-updates are sometimes incompatible with the controls applications DLLs.
- Application Program Error. Main area of problems given by the unexpected degrees of freedom an application programmer implicitly used coupled with the 'optimistic' programming (failure to check return codes) which frequently happens when one is trying to work quickly under pressure. Very often the set of values a program can deal with is not selected precisely. That is often the case with values read-out from the fieldbusses if the illegal values are not masked out.
- Changing operating conditions for the machines. This can result in unforeseen behaviour of application programs and may cause harm to the machine.
- Virus or worm attacks: Some of our consoles were attacked by the sasser worm in May 2004, where the virus reached the PCs earlier than the virus-signatures-updates for the installed and active virus-scanner.
- Database inconsistencies. The name resolution service for control-processes is essential und a corrupt or incorrect entry may lead to strange behaviour. Even DNS (IP Domain Name Service) or WINS (Windows Name Service) can cause problems in the control system.

The main contribution to systems faults comes from application programs, from one of several avenues mentioned above. Standardized automated testing mechanisms are currently being commissioned, which should help alleviate this problem.

### *Archiving*

We now archive faults and the measure against in a software electronic logbook, originally developed to document the TTF operation [3]. By so doing, we are able to track a system fault from the point of its inception to (hopefully) its elimination.

## STRATEGY TO AVOID FAULTS

Good fault-statistics should lead to preventive measures to avoid faults. We use different strategies to attack the problems

- **Network:** avoid mixture of old and new technologies; use only TCP/IP-protocol, get rid of IPX-protocol.
- **Operating systems:** support a small number of well-known systems. Configure lean systems**,** don't install unneeded components. Reduce Windows OS for device-servers.
- **Clone systems:** exclude faults from bad system configurations use automatic install procedures to get as identical systems as possible.
- **Hardware variety:** work with large numbers (~20) of identical PC-Hardware to run the processes. By so doing it's easy to demonstrate that you have faulty hardware if the problem exists on only one PC of a family of similar ones.
- **Applications:** use templates and wizards to create applications from scratch. Provide tools to run automatic checks against applications which control the answers returned by requests for standard control-system properties as well as application specific properties.
- **Standards:** established independent from operating systems. So having platform independent tools available to check all levels of control system architecture.

## CONCLUSION

Fault identification plays an important role in establishing reliable and readily available control systems. Such control systems are a prerequisite for highly efficient operation which is required for the new PETRA III accelerator to be built at DESY [4]

## REFERENCES

[1] Rüdiger Schmitz, "A Control System for the DESY Accelerator Chains", proceedings of PCaPAC'99, http://conference.kek.jp/PCaPAC99

[2] TINE (Threefold Integrated Network Environment), http://desyntwww.desy.de/tine

[3] R. Kammering, O. Hensler, K. Rehlich, A. Petrosyan, "Review of Two Years Experience with an Electronic Logbook", ICALEPCS 2003, Gyeongju, Korea

[4] K. Balewski, W. Brefeld, W. Decking, H. Franz, R. Röhlsberger, E. Weckert (Editors), "PETRA III: A Low Emittance Synchrotron Radiation Source", Technical Design Report, DESY 2004-35, February 2004