A Systems Approach to Exception Handling

Herbert Hecht
SoHaR Incorporated
herb@sohar.com

Abstract

Exception handling is an important function in high assurance systems and also a significant contributor to system failures. Yet, system aspects of exception handling, and particularly the formulation of requirements, have received little coverage in the literature and practices vary widely.

To progress beyond that stage the paper analyzes the sources for exception handling requirements and the format in which the requirements can be inserted into the system design. From this a plan has been generated for presenting exception handling requirements and monitoring their implementation during system development. It is intended that this is a first step towards a discussion of the need for a systems approach to exception handling.

1. Introduction

The essential characteristics of a high assurance system are (a) fault avoidance by use of highest quality components, well trained and motivated personnel plus extensive inspection and test resources, and (b) fault tolerance provisions to prevent system failures due to "escapes" from fault avoidance or unexpected environmental conditions. This paper is concerned with the fault tolerance aspects of high assurance systems and particularly with its control mechanism -- exception handling (EH). The key quality of fault tolerance is "to provide, by redundancy, service complying with the specification in spite of faults having occurred or occurring"[1]. The redundancy can be static (without switching) or dynamic (with switching). The former is primarily used for power supplies (with diode isolation) and computers (in the triple modular redundancy or TMR configuration). Dynamic redundancy is more widely applicable; it requires a switching decision which is an instantiation of EH and is the target of the following discussion. Practically all decision processes for EH are currently implemented in software (including firmware) even when the final switching element is a relay or similar component. Therefore EH in software is emphasized here.

EH is a frequent cause of system failures because it receives insufficient attention during design (where the emphasis is on achieving the primary system function) and in testing because of the difficulty in constructing test cases that cover all the exceptional circumstances that need to be handled. Thus it is not surprising that as part of the Y2K software reviews it was recognized that errors in exception handling were responsible for most failures. "The main line software code usually does its job. Breakdowns typically occur when the software exception code does not properly handle abnormal input or environmental conditions – or when an interface does not respond in the anticipated or desired manner"[2]. Similar observations had also been reported earlier, with the added factor that failures due to EH were proportionately more frequent in software controlling critical system functions[3]. Thus EH is of importance to high assurance systems both as a potential cause of failures and as a means for their prevention.

The system approach to EH is necessary because

- EH can exist at multiple levels of the system hierarchy and interference between EH provisions must be avoided,
- some needs for EH arise from top level system safety or availability requirements while others originate from characteristics of sensors or processors that are defined at much later stages of the design but must be integrated with the higher level EH provisions
- the implementation and refinement of EH provisions is carried out in many specialist organizations and requires continuous system level review to avoid lapses in coverage, undesired overlap, and the potential for interference.

The body of the paper addresses the current practice of EH and the problems arising from that and then points

out the need for a systems approach. From there it develops a plan for generating EH requirements in the life cycle and monitoring their implementation. Finally we suggest future steps to a systems oriented approach to EH.

2. The Scope of Exception Handling

The programmer views EH as a task that involves detecting an abnormal condition, stopping the normal execution, saving the current program state, and locating the resources required for continuing the execution. Examples of the fairly extensive publications on this aspect of exception handling are [4] [5]. In this paper the emphasis is on the system aspects, and particularly the requirements for EH. It will be shown that needs for EH originate from multiple sources and will typically become known over an extended period of the system development. These circumstances have made it difficult to view the full scope of requirements for EH that will now be discussed.

The "system" in this context is a computer based component of a "plant", such as a flight control system of an airliner, a weapons control system of a military aircraft, or the emissions control system of a refinery. The placement of EH in this hierarchy is shown in Figure 1. In high assurance systems most needs for EH will come from one of the following areas

- Plant functions
- Plant protective measures
- Computing environment anomalies
- Monitoring and self-test
- Primary software function anomalies

Details of each of these will now be discussed.

2.1 Plant Functions

The most frequently encountered plant functional needs for EH arise from start-up and shut-down conditions, e. g. the action to be taken when required power and communication lines are not available or when start-up of a companion program has been delayed. Calibration, mode change commands and support for exception handling at the plant level may also be sources for EH. These needs for EH are usually known early in the life cycle and are less likely to change than needs from othersources.

2.2 Plant protective measures

This heading covers the most diverse and demanding needs for EH. It includes protection against unsafe or

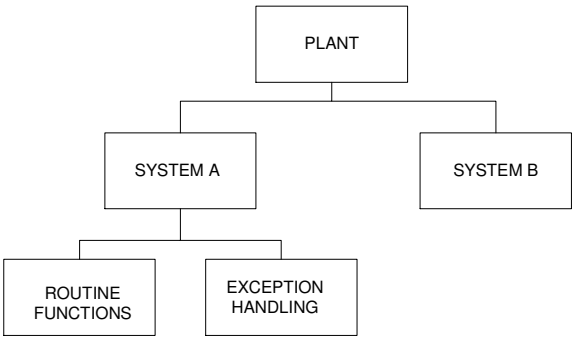

Figure 1 Hierarchy for Exception Handling

undesirable states of the plant, redundancy management and activation of alarms and physical barriers. Aerospace systems that are "essential for continued safe flight and landing [of aircraft][6]" are frequently made redundant in their entirety or on a channel basis. Exception handling is required for isolation of the failed component or channel and reconfiguration of the remaining units into a survivable structure. In process control applications similar redundancy management may be applied to sensors, and shut-down may have to be initiated for failures where no redundant coverage is available. EH may be required to avoid undesirable states (e. g., excess fuel consumption) by initiating corrective measures or by activating an alarm.

EH under this heading may also be required to protect from unsafe or undesirable operator actions or to prevent unauthorized personnel from operating the system or equipment. Similarly, protection against inappropriate maintenance actions may use EH.

Requirements for exception handling originating from plant protective measures are usually known at least in outline form early in the development cycle but will evolve and possibly change through much later stages.

2.3 Computing environment anomalies

Some requirements for EH arise from the computer hardware, such as handling of memory errors, divide-by-zero exceptions, and overflows or underflows. These requirements are usually known as soon as the hardware is specified and can be expected to remain stable unless the hardware is replaced.

A substantial part of EH arises from the software component of the computing environment, including the executive or operating system and middleware. For real-time systems that operate on fixed cycle times these software components signal when time limits are

about to be violated and they may also invoke their own EH to deal with these conditions. Because the higher level applications are usually more informed about the consequences of a missed cycle and about alternative means of accomplishing a function it may be desirable to disable the low level EH interfere with the more beneficial actions programmed in the applications. Commercial operating systems include watchdog timers and resource monitors that serve a similar purpose. Again, requirements for EH in the system under development must be formulated with awareness of all these native protective measures lest they interfere with the specifically programmed ones.

Requirements for software environment protective measures will usually follow those arising from the hardware environment and are likely to change throughout the application software development.

2.4 Monitoring and self-test

The original system design may include monitoring or self-test provisions to facilitate the operator's task or to aid in maintenance. Frequently such functions are added or enhanced during the course of the development. Monitors are usually active at all times and EH is required only when they indicate anomalous conditions. Self-test may be invoked for any system function but it is particularly important where passive sensors monitor critical system states, such as an over-temperature sensor for a temperature sensitive electronic component. Self-test may be initiated autonomously, as a result of an indication by a monitor or due to an exception condition encountered by the application. In either case the requirements for EH must provide for appropriate action under all outcomes of the self-test.

2.5 Primary software function anomalies

The exception conditions under this heading arise directly from an anomalous state or transition of the application software but indirectly they may respond to any anomaly in the plant. A source for the types of software failures that need to be detected and mitigated can be found in a taxonomy compiled by senior figures in the field of dependable computing[7]. The reference lists 12 types of software faults that can impede the intended execution of a program (Figure 5 of the reference), not all of which may be applicable in a given environment but that need to be at least considered. Included in the list are maliciously introduced faults that can be expected to affect the system in a particularly critical operational state. The reference also provides a taxonomy of recovery provisions in which a distinction is made between *error handling* (replacing erroneous data values) and *fault handling* (the isolation, removal and replacement of permanently damaged data stores or instructions).

Because these exception conditions are closely tied to the software development the requirements they can only be formulated in very general terms in the development stages prior to software design and are also the most likely to change in test and operation.

Within the primary software there are of course many decision nodes to deal with the diversity of input conditions and computer states and it is necessary to distinguish between EH and functions that deal with the routine variability of program execution. A useful criterion is that EH usually involves transfer of program control to a new context.

3. The Need for a Systems Approach

The variety of sources for EH requirements and the extent of the time interval over which these requirements will be made known are at present being accommodated in an *ad hoc* manner in which some requirements may be part of the system acquisition documentation, some may be incorporated in the software requirements, and others may be transmitted through change orders. EH for the last category mentioned in the previous heading, the primary software function anomalies is frequently generated by analysts and programmers with only minimal guidance from outside the software community. This practice is not efficient, does not promote reliability at either the system or the plant level, and impedes system certification because there is no comprehensive body of requirements against which the correctness and completeness of EH can be evaluated.

The primary purpose of the systems approach advocated here is to support the procurement of a system with a comprehensive plan for EH with emphasis on requirements. At the earliest stage the plan should list for each EH requirement

- The source (as discussed in the preceding section to which other sources may be added)
- The criticality of the requirement (consequences of failure of EH)
- Potential means of detection (precursor events, variables affected, etc.)
- Potential mitigation measures and their cost (negative effect on plant performance)
- Coordination with plant level EH requirements

These will be augmented in time as the plant and system design progress. A suggested time-phased

Table 1. Formulation of EH Requirements

Title	Purpose	Content
Objective	Estimation of effort (procurement)	Conditions to be avoided, criticality, means of detection, potential mitigation
Algorithm	Feedback to procurement Basis for specification	Variables for detection, detection criterion, action for mitigation, verification of mitigation
Design	Basis for implementation	Placement of EH in the program, timing restrictions, prevention of interference with other EH provisions

format for the formulation of EH requirements is shown in Table 1. The *Objective* document will usually be generated by the platform developer's system engineering organization. The other two documents are normally the responsibility of the system developer's system engineering organization, with the *Design* requiring coordination with the software developer.

The Content column lists only the most frequently encountered items. It may need to be augmented project specific needs such as review and verification requirements, multiple mitigation provisions (defense in depth), and changes in EH requirements with usage conditions of the plant or of the system.

4. A Plan for Exception Handling

The discussion of the scope of EH (in Section 2 above) identified five sources from which requirements typically arise:

- Plant functions
- Plant protective measures
- Computing environment anomalies
- Monitoring and self-test
- Primary software function anomalies

This listing represents the order in which requirements usually can be expected to become known. By combining the ordering by sources with the evolution of each individual requirement from Table 1 a comprehensive plan for EH over the life cycle can be formulated. An example of such a plan is shown in Figure 2. The life cycle phases shown along the horizontal axis refer to the system being developed or procured, e. g., a weapon control system. It is seen that EH is a significant activity throughout the development. By identifying the activities in a plan and by controlling their execution in accordance with the plan the following benefits (compared to the *ad hoc* approach) will be achieved:

- Reduction of development cost by coordination of the activities of plant and system developers and by avoidance of emergency change requests
- Improved reliability and quality by reduction of
    - missing EH requirements
    - interference among EH requirements
    - erroneous criticality assignments

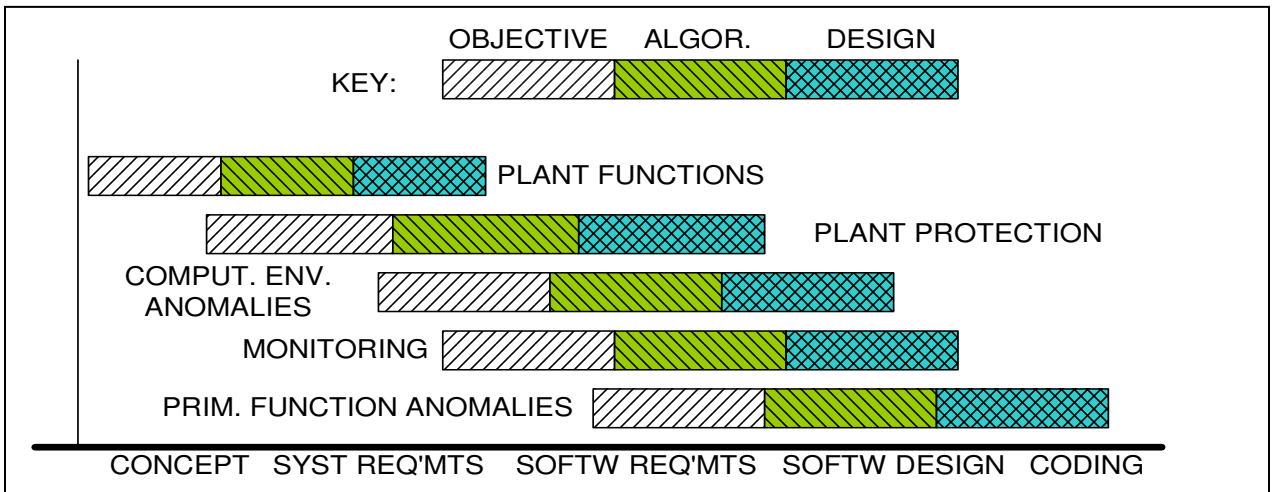

Figure 2 Exception Handling during Development

- Facilitation of verification because there is a listing of all required EH

A comprehensive plan should also include provisions for changing requirements for EH and origination of additional ones during the operation phase.

5. Conclusions

Although EH is recognized as an important element in the design and operation of high assurance systems, and although the software implementation of EH has been discussed in the literature, the system engineering component has not received any attention. Thus, no comprehensive guide for the formulation and review of EH requirements is in common use. As a result, EH is specified and implemented in an *ad hoc* fashion that leaves much room for errors and omissions. The lack of a systematic and generally recognized approach to EH causes problems in verification because there is no single source of EH requirements that can serve as a basis for verification.

As a start towards a systematic approach this paper has investigated the sources from which requirements for EH can arise and has identified the following five as being typical for high assurance systems:

- Plant functions
- Plant protective measures
- Computing environment anomalies
- Monitoring and self-test
- Primary software function anomalies

The order of this listing represents the typical time line in which requirements can be generated. Each set of EH requirements will normally start with a statement of EH *objectives* from which the EH *algorithm* will be developed that will form the basis of the EH *design*. The time ordering of the sources for EH requirements and these three phases of the EH generation process are the essential elements for an EH plan. The preparation of this plan will

- Permit EH to become a part of the overall project planning
    - Management reviews for budgeting, scheduling, staffing
    - Technical reviews for consistency and completeness
- Provide documentation to aid in subsequent steps and to pass on "lessons learned" to future projects
    - Basis for test and verification of EH
    - Communication with sponsor or regulatory agency
    - Modification of approach and tactics if required
- Form the basis for standardization of EH

The material presented here is intended to initiate discussion of the need for systematic EH and the consideration of alternatives to the approach presented above.

Acknowledgement

Portions of the research reported here were supported by Contract W911QX-06-C-0116 with the U. S. Army RDECOM. The encouragement of the Technical Representative, Jeffrey Dehart, is much appreciated.

References

[1] Laprie, Jean-Claude, "Dependable Computing and Fault Tolerance: Concepts and Terminology"', *Highlights from 25 Years – FTCS-25,* IEEE Computer Society Press, June 1995, p.2

[2] C. K. Hansen, "The Status of Reliability Engineering Technology 2001",*Newsletter of the IEEE Reliability Society,* January 2001.

[3] Hecht, Herbert and Patrick Crane, "Rare Conditions and their Effect on Software Failures", *Proceedings of the 1994 Reliability and Maintainability Symposium,* pp. 334 - 337, January 1994

[4] Doshi, Gunjan *Best Practices for Exception Handling*, http://www.onjava.com/pub/a/onjava/2003/11/19/exceptions.html

[5] Christophe de Dinechin, *C++ Exception Handling for IA-64* https://db.usenix.org/events/wiess2000/full_papers/dinechin/dinechin.pdf

[6] Federal Aviation Regulation (FAR) 25.1309

[7] Avizienis, A., J. C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", *IEEE Transactions on Dependable and Secure Computing,* vol. 1, No.1, Jan 2004