Error-Prone Exception Handling in Large Ada Systems

Gary N. Bundy and Diane E. Mularz
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102 USA
gbundy@mitre.org and mularz@mitre.org

Abstract. An ongoing research project of the MITRE Corporation is investigating the use of exception handling in large-scale Ada applications. Through manual and static analysis of artifacts, we have identified patterns of use that can lead to defective behavior in the event of an exceptional condition. We classify these patterns as error-prone. As a guide for future developers of Ada applications we describe these error-prone patterns, how such usage can lead to defects, alternative implementation strategies where known, and how static analysis can be effectively used to identify these error-prone patterns.

1 Introduction

Exception handling was introduced into the Ada language as a set of constructs that could effectively separate normal from exceptional processing (Ichbiah, 1979). What constitutes exceptional processing has been zealously debated in many forums with no clear conclusions. While these debates make for interesting discourse, the lack of consensus places the burden of appropriate usage on each developer. Given this situation, it is useful for the community to understand the nature of exception handling, the usage patterns that are employed in the development of large-scale Ada applications, and where the pitfalls are in the exception handling design of an application. This is the premise of ongoing research being conducted within MITRE.

Previous studies of defects in applications point to exception processing as the root cause of a significant number of reported defects (Mays, 1990; Perry, 1985). Not only is exception handling a source of error, but these errors tend to exhibit what Parnas describes as weak-link behavior i.e., a defect in an isolated portion of an application can have global impact (Parnas, 1990). Incorrect exception handling in Ada can exhibit this weak-link behavior due to uncontrolled propagation semantics (Howell, 1991). These facts, coupled with our findings, suggest that early consideration of exception processing at the conceptual design stage, along with articulation of an exception handling policy at the application level, is important. Unfortunately, exception handling decisions are easily deferred to the unit level in Ada designs where individual, ad hoc decisions will result:
It is easy to think of exceptions as a kind of 'add on' feature of a software design, i.e., a facility whose use is left to the designers of individual modules and routines. In fact, exceptions are used most effectively only if the strategy guiding their use is developed in the earliest stages of a design. Moreover, effective use of exceptions requires attention throughout a design. (Goodenough, 1991).

Results of our study of several large-scale applications support Goodenough's conclusions. Exception handling tends to be a weak link in the architecture of an application. Design policies regarding exceptions are generally not well-formed or nonexistent. Many exception handling flaws go unnoticed through the development and testing phases. What is needed by the software engineering community is an increased emphasis on exception handling design as well as analysis capabilities for detecting flaws in the design. Our research has focused on an analysis capability with the goal of producing design guidance as a by-product. We have constructed a set of diagnostics for analyzing exception handling in Ada systems. These diagnostics detect patterns of use that upon further investigation have been shown to result in defective behavior in the event of an exception. This paper presents these error-prone patterns as guidance to future designers.

The remainder of this paper provides a brief overview of the terminology used in this paper, a description of the recurring error-prone patterns found in the applications examined, an overview of the tools used to support this analysis, and some conclusions based on these findings.

2 Terminology

An exception is a dynamic object that is associated with one or more anticipated events and is used to notify the application that the event has occurred. Exceptions have static and dynamic properties, such as their scope and effect on normal program control flow. Five predefined exceptions are specified for the language in package STANDARD and visible to all units; other system exceptions are declared in the predefined packages IO_EXCEPTIONS and CALENDAR (MIL-STD-1815A, 1983). User-defined exceptions are declared as needed by the application. Visibility to user-defined exceptions follows the same rules as other declarations.

The events that an exception is associated with are referred to as exceptional conditions. In an exceptional condition status of an attempted computation will take precedence over the results of a computation in a given application (Howell, 1992). Declaring an exception associates an indicator with the exceptional condition that might occur. Raising an exception transfers control to the appropriate exception