Current Trends in Exception Handling

Dewayne E. Perry, *Member*, *IEEE Computer Society*, Alexander Romanovsky, and Anand Tripathi, *Member*, *IEEE*

1 Introduction

The importance of exception handling is well-recognized by system designers and software engineers. Exception handing is very often the most important part of the system because it deals with abnormal situations. The goal of exception handling mechanisms is to make programs robust and reliable. However, for a variety of reasons, not the least among which is the fact that more than half of the code is often devoted to exception detection and handling, many failures are caused by incomplete or incorrect handling of these abnormal situations. Analysis of accidents in computer controlled systems has shown that very often their causes tend to be in improper dealing with exceptional situations (see, for example, [1], [2]). The requirements for correct system behavior during exception handling are in some sense even higher than for the system operating in a normal mode. Even the programs that are generally considered to be highly robust, such as operating systems, can be significantly error prone, as discussed in the paper by Koopman and DeVale. The implication is that exception handling needs are more pervasive than what many people may think, and that little can be assumed to work correctly.

In the 1970s, research on exception handling was conducted mainly in the context of language design. Now the situation is different: exception handling issues are being studied and researched in the context of a variety of disciplines, such as system structuring, fault-tolerant computing, dependability, system specification, software engineering, object-oriented systems, programming language design, real-time systems, CSCW, process and workflow systems, etc. Many of these disciplines have faced unique requirements in developing exception handling models.

Different application-domains, programming languages, and computation models have generally led to variants of exception handling models and mechanisms, making it difficult at times to transfer knowledge from one domain to another. Researchers working in different areas very often have no opportunity to exchange their ideas and results. There is hardly any conference or journal in which topics related to exception handling are not discussed. However, the research efforts tend to be scattered and not recognized by a wide audience as an important direction on its own. In a way, this research is considered to be subsidiary to that on languages, system structuring, dependability, and system specification. This special issue has been motivated by the need to bring together the important strands of research and practice in various disciplines.

We started our work on this issue nearly two years ago. We received 39 submissions by February 1999. After an extensive review process, involving help from more than 110 reviewers, we selected nine high quality papers to appear in this special issue. These papers represent the important research directions and issues in exception handling in a broad range of disciplines, such as operating systems, programming languages, workflow systems, spreadsheet programs, requirements specifications, etc. The uniqueness of this special issue is reflected by this broad range of areas of computer science covered by the papers in this special issue. We did not make any attempts to impose a unified understanding or terminology, and this issue, obviously, shows different perspectives on exception handling problems in the various disciplines. We hope that the main objective of this special issue of bringing together the important strands of research and practice in regard to exception handling in various disciplines of computer science has been achieved, and that it will help in developing a common understanding and cross fertilization in this research.

2 The Articles

This special issue is published in two parts. The first part includes papers which focus on traditional issues related to programming languages, program design testing, OS level issues, and empirical studies.

In the paper "Advanced Exception Handling Mechanisms," Peter Buhr and Russell Mok address issues related to exception handling in concurrent programming languages supporting coroutines and tasks. This paper examines interactions between exception handling mechanisms and other language mechanisms in concurrent systems. Traditionally, exception handling mechanisms in sequential programming languages have been based on control flow models, such as termination, resumption, replacement, etc. Buhr and Mok examine both termination as well as resumption models in the context of uC++ language. They present a unified framework for supporting both these

- *D.E. Perry is with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712. E-mail: perry@ece.utexas.edu.*
- *A. Romanovsky is with the Department of Computing Science, University of Newcastle upon Tyne, Newcastle upon Tyne NE1 7RU, UK. E-mail: alexander.romanovsky@newcastle.ac.uk.*
- *A. Tripathi is with the Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455. E-mail: tripathi@cs.umn.edu.*

For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 111496.

0098-5589/00/$10.00 © 2000 IEEE

models. They show how exception handlers can be partitioned to support both these models.

The paper by Philip Koopman and John DeVale, "The Exception Handling Effectiveness of POSIX Operating Systems," will be of interest to readers involved in studying, developing and using operating systems. The Ballista testing system has been developed and used by the authors to analyze the behavior of 15 widely used POSIX implementations in situations when exceptional input parameter values are passed in POSIX function and system calls. It has been found that these implementations correctly report an error code in only 55—76 percent of such exceptional calls. The paper thoroughly analyzes different types of exceptional system behavior and the prevalent sources of robustness failures caused by them. One of the highly commendable Ballista characteristics is its openness: there is a site on which one can test any of these implementations via the Internet. The paper discusses one of the first results of the extensive testing of several off-the-shelf implementations of an operating system which demonstrates to which extent one can rely on such systems and emphasizes again the significance of rigorous exceptional system behavior specification.

Saurabh Sinha and Mary Jean Harrold in "Analysis and Testing of Programs with Exception-Handling Constructs" are concerned about the use of analysis techniques such as control flow, data flow, and control dependence in Java and C++ programs where the effects of exception occurrences and exception handling must be accounted for. Since these analysis techniques are used in a large variety of software engineering tasks, there is a need to find representations for programs where explicit exceptions are raised and handled, and then to define algorithms that use these representations to perform these kinds of analyses. To support the need for such work, the authors have included the results of three empirical studies to illustrate the frequency of exception use in a set of Java programs, the relatively low frequency in which type inferencing is needed for throw statements, and the effects of exceptions on control dependence analysis.

"Exception Handling in Java—Meaning and Compiler Correctness," by Egon Boerger and Wolfram Schulte describes a rigorous framework for language and platform independent design and analysis of exception handling mechanisms and their implementations. They model the Java exception mechanism and a stripped down version of the Java Virtual Machine (JVM) as Abstract State Machines. The two models are related to each other by a functional description of the compilation from Java exception handling to JVM code. They then show that the design of these pieces of Java/JVM is semantically correct. They view exception handling as a runtime problem, and as such, needs different techniques to show semantic correctness. The authors' use of these techniques also supports the claim that practitioners can document their designs and justify that design's appropriateness in a rigorous way.

Roy Maxion and Robert Olszewski in their paper, "Eliminating Exception-Handling Errors with Dependability Cases: A Comparative, Empirical Study," report a well thought-out experiment—testing the hypothesis that robustness for exception failures can be improved through the use of various coverage enhancing techniques: N-version programming, group collaboration, and dependability cases. The results demonstrate that although all techniques show improvements over control conditions in increasing robustness to exception failures, dependability cases proved the most efficient in balancing cost and effectiveness (the authors carefully discuss assumptions and conditions under which the experiments are run and this conclusion is drawn). The concept of dependability cases, introduced by the authors, relies on a methodology based on structural taxonomy and memory aids for helping software designers to improve exception handling coverage. The results obtained are amazing and worrying at the same time: even though a simple, non-real-world task was used, a huge number of mistakes and exceptions were found. This can be seen as another warning against expecting anything better in the real world without taking measures to improve exception handling.

ACKNOWLEDGMENTS

We would like to thank the authors of all submitted papers for their interest in our special issue. The support and cooperation from the former *IEEE Transactions on Software Engineering* Editor-in-Chief A. Kemmerer and the IEEE staff has helped us a lot in the preparation of this issue. Finally, we wish to thank all reviewers whose time and efforts have made this issue possible: A. Acharya, M. Aksit, L. Alvisi, A. Appel, J. Arlat, A. Arora, B. Balzer, P. Banerjee, R. Barga, S. Baruah, F. Bastani, I. Bate, L. Bellissard, B. Bershad, E. Bertino, G. Bolcer, A. Borgida, C. Bron, T. Budd, A. Carzaniga, E. Chi, S. Chiba, P. Ciancarini, G. Cugola, R. Cytron, P. Dasgupta, A. Datta, R. de Lemos, F. Di Giandomenico, E. Di Nitto, C. Dony, S. Easterbrook, C. Ellis, P. Ezhilchelvan, P. Felber, T. Finholt, I. Forgacs, J. Gannon, V. Garg, D. Georgakopoulos, L. George, D. Hamlet, D.K. Hammer, B. Harper, M. J. Harrold, L. Hatton, M. Heimdahl, T. Herman, M. Hidehiko, Y. Hur, M. Jones, G. Kaiser, P. Kammer, M. Kandemir, K. Kim, S. Kirani, J. Knudsen, A. Koenig, M. Koutny, D. Lea, I. Lee, F. Leymann, D. Locke, D. Long, M. Lyu, J. Magee, J. McHugh, R. Miller, B. Moo, J. Moore, D. Mosse, K.-I. Murata, A. Nocolau, G. Nutt, R. Olsson, L. Osterweil, M. Pezze, J. Rieke, N. Ramsey, A. Ricciari, D. Richardson, J. Riedl, C. M.F. Rubira, A. Saeed, R. Schlicting, K. Schwan, S. Shekhar, A. Shith, J. Simeon, A. Singh, M. Sloman, A. Somani, J. Srivastava, A. Stoyenko, L. Strigini, B. Stroustrup, S. Sutton, P. Tarr, A.M. Tyrrell, R. van Renesse, J. Voas, R. Voyles, P. Wadler, Y.-M. Wang, L. Welch, A. Wellings, S. Wheater, J. Wu, P.-C. Yew, and M. Young.

REFERENCES

[1] J.L. Lions (chairman), "Ariane 5 Flight 501 Failure: Report by the Inquiry Board," European Space Agency, Paris, July 19, 1996.
[2] "Report on AT&T U.S. Long-Distance Network Outage in January 15, 1990," *Software Eng. Notes,* vol. 15, no. 2, p. 12, 1990.

Dewayne E. Perry is currently the Motorola Regents Chair of Software Engineering at the University of Texas at Austin (UT Austin). The first half of his computing career was spent as a professional programmer, with the latter part combining both research (as a visiting faculty member in computer science at Carnegie-Mellon University) and consulting in software architecture and design. The last 16 years were spent doing software engineering research at Bell Laboratories in Murray Hill, New Jersey. His appointment at UT Austin began January 2000.

His research interests (in the context of software system evolution) are empirical studies, formal models of the software processes, process and product support environments, software architecture, and the practical use of formal specifications and techniques. He is particularly interested in the role architecture plays in the coordination of multisite software development, as well as its role in capitalizing on company software assets in the context of product lines.

His educational interests at UT include building a great software engineering program, both at the graduate and undergraduate levels, creating a software engineering research center, and focusing on the empirical aspects of software engineering to create a mature and rigorous empirical software engineering discipline. He is a coeditor-in-Chief of Wiley's *Software Process: improvement and Practice*; a former associate editor of the *IEEE Transactions on Software Engineering*; a member of the ACM SIGSOFT and the IEEE Computer Society; and has served as organizing chair, program chair, and program committee member on various software engineering conferences.

Alexander Romanovsky received the MSc degree in applied mathematics from Moscow State University in 1976 and the PhD degree in computer science from St. Petersburg State Technical University in 1988. He was with St. Petersburg State Technical University from 1984 until 1996, doing research and teaching. In 1991, he worked as a visiting researcher at ABB Ltd. Computer Architecture Lab Research Center, Switzerland. In 1993, he was a visiting researcher at Istitutodi Elaborazione della Informazione, CNR, Pisa, Italy. In 1993-94, he was a postdoctoral fellow at the Department of Computing Science, University of Newcastle upon Tyne, UK. From 1992-1998, he was involved in the Predictably Dependable Computing Systems (PDCS) ESPRIT Basic Research Action and the Design for Validation (DeVa) ESPRIT Basic Project. From 1998-2000, he worked on the Diversity in Safety Critical Software (DISCS) EPSRC/UK Project. He is currently a senior research associate with the Department of Computing Science, University of Newcastle upon Tyne, working on the Dependable Systems of Systems (DSoS) EC IST RTD Project. His research interests include software fault tolerance, software diversity, concurrent programming, concurrent object-oriented and object-based languages, real time systems, exception handling, operating systems, software engineering, and distributed systems. He has coauthored more than 120 scientific papers, book chapters, reports, and a patent in these and related areas.

Anand Tripathi obtained his BTech degree in electrical engineering from the Indian Institute of Technology, Bombay, 1972. He received the MS and PhD degrees in electrical engineering from the University of Texas at Austin, in 1978 and 1980, respectively. From 1981 through 1984, he was a senior principal research scientist at Honeywell Computer Science Center in Minneapolis. There he led research and development projects in distributed operating systems with a focus on fault tolerance and distributed object management. He also was involved in the development of TDC-2000, a distributed process control system developed by Honeywell. He joined the University of Minnesota in 1984. While at the University of Minnesota from 1985 to 1989, he led the design and development of a middleware system called Nexus for distributed object-based computing. From 1995 to 1997, he also served as the program director for the Computer Systems Software program at the U.S. National Science Foundation. Currently, he is an associate professor in the Departmentof Computer Science Engineering at the University of Minnesota. He has published more than 60 research papers in various journals and conferences in the areas of distributed systems, object-oriented systems and languages, and fault-tolerant computing. His current research focus is on system level mechanisms for building robust and secure distributed applications using mobile agents. He is a member of the IEEE, the editorial board of *IEEE Concurrency*, and the editor for the education column of *IEEE Concurrency*. He is also currently serving as an IEEE distinguished visitor.