Exception Handling in Object Oriented Systems: Developing Systems that Handle Exceptions

Alexander Romanovsky[1], Christophe Dony[2], and Anand Tripathi[3]

[1]School of Computing Science, University of Newcastle upon Tyne
Newcastle upon Tyne, NE1 7RU, UK
alexander.romanovsky@ncl.ac.uk

[2]Université Montpellier-II and LIRMM Laboratory,
161 rue Ada, 34392 Montpellier Cedex 5, France
dony@lirmm.fr

[3]Department of Computer Science, University of Minnesota
Minneapolis, MN 55455, USA
tripathi@cs.umn.edu

Abstract. Exception handling is an important part of software and system architectures. The scale of operations of modern software systems in complex network applications and embedded systems raises numerous possibilities of failures and exception conditions. Over the past two decades, object-oriented design principles and programming techniques coupled with sophisticated engineering tools have significantly reduced the complexity of the design and implementation process for modern software systems. However, integration of suitable exception handling techniques with object-oriented design methodologies and programming models has continued to raise many important research challenges. Whereas several such challenges arise in the context of fundamental constructs in object-orientation, many new research issues are stemming from the unique nature of emerging application environments, requiring domain-specific models for exception handling. This workshop was organized to facilitate discussion of such issues and related techniques in order to develop a common understanding of the current and future direction of research in this field.

1 Summary of Objectives and Results

Modern systems are becoming increasingly more complex and the number of exceptional situations they have to cope with is increasing. The most general way of dealing with these problems is employing the exception handling techniques. While a number of object-oriented mechanisms for handling exceptions are supported in contemporary programming languages and design methodologies, there are still serious problems with applying them in practice due to several reasons such as the complexity of the design and analysis of exception code, failure to employ exception handling at the appropriate development phases, and lack of adequate methodologies supporting correct use of exception handling as well as lack of mechanisms specific to various application domains and design paradigms. Several new and emerging paradigms of computing and application environments, such as pervasive computing, ambient computing, sensor networks, embedded systems are making increased demands for application robustness through adaptation to handle failures and exceptional conditions.

Building on what was achieved in ECOOP'2000 and ECOOP'2003 workshops on this topic, this workshop provided a forum to discuss research on exception handling and fault tolerance in all areas of object-based and component-based software development and use. We solicited research and experience papers in all areas of exception handling in object oriented systems, in particular: formalization, distributed and concurrent systems, practical experience, mobile object systems, new paradigms (e.g. object oriented workflows, transactions, multithreaded programs), design patterns and frameworks, practical languages, open software architectures, aspect-oriented programming, and component-based technologies. One of the aims of the workshop was to encourage the researchers in this field to look at exception handling issues in the context of the entire software life cycle as well as techniques and mechanisms for domain-specific exception handling models. The participants were encouraged to report their experiences of both benefits and obstacles in using exception handling, practical results in using advanced exception handling models, and the best practices in applying exception handling techniques in novel applications environments.

The workshop was attended by nineteen researchers who participated in the presentation and discussion of 13 position papers and two invited talks. These presentations and discussions were grouped into four sessions. The workshop started with the invited lecture by Bertrand Meyer on *Disciplined Exceptions*. It was followed by two presentations – one was on exception handling in asynchronous systems and the second presentation on testing of error recovery code in Java applications. The presentations in the second session addressed a variety of topics including the use of conversations in ambient computing systems, exception handling in mobile agent applications, techniques for improving performance of exceptions handling code, issues with handling exceptions during callback, and handling of concurrent exception in C++ using futures. The third session began with two presentations. The first presentation was on modeling of exception handling in UML 2.0 and the other addressed specification of exception condition in object-oriented systems. This session was followed by the second invited lecture by Andrew *Black on Exception Handling: The Case Against*. The last session consisted of four presentations on topics related to exception handling issues in context-aware systems and embedded real-time systems, the use of aspects in exception handling, and effective techniques for dealing with "out of memory" errors.

2 Summary of the Call-for-Papers

The call-for-papers for this workshop emphasized its broad scope and our desire to focus the workshop discussions on problems of perceived complexity of using and understanding exception handling. The call-for-papers invited the participants to contribute their research results and position papers on the following topics:

- Modeling and applications with exceptions. How can such applications be checked, verified and proved? Which formal models are appropriate? We encourage submissions focusing on modeling system exceptional behavior with UML-like languages and tools.
- Perceived complexity of using and understanding exception handling. Do programmers misuse exception handling? If so, why? Why do programmers and practitioners often believe that exception handling complicates system design and analysis? Why is exception handling the last mechanism for programmers to learn and to use?
- Novel exception handling and fault tolerance solutions for the new computing contexts: mobile and agent-based systems, pervasive and ambient environments, self-repairing, adaptive and open systems, distributed and asynchronous systems, transaction-based and multi-threaded programs.
- Software architectures which support exception handling and design patterns which help to develop systems that handle exceptions systematically.
- Programming constructs for exception handling. Are Java checked exceptions an appropriate solution for module specification? What are the post-Java constructs for exception handling? Which forgotten or unused past solutions should be brought back? Are they suitable for Java or C# like statically typed languages?
- Experience reports which illustrate benefits to be derived from as well as difficulties involved in using exception handling, summarize practical results of employing advanced exception handling models and the best practices in applying exception handling for developing modern applications.

All submitted papers were reviewed by the organizing committee and were found relevant to this workshop. As part of the invitation letters sent to the participants, the organizing committee provided some initial feedback and comments to the authors to suitably refine their papers for inclusion in the proceedings of the workshop. The workshop proceedings were published as a technical report by the Department of Computer Science, LIRMM. Montpellier-II University (France):

Romanovsky, A., Dony, C., Knudsen, J. L., Tripathi, A. (Eds.) *Developing Systems that Handle Exceptions. Proceedings of ECOOP 2005 Workshop on Exception Handling in Object Oriented Systems.* Technical Report No 05-050. Department of Computer Science. LIRMM. Montpellier-II University. 2005. July. France.

This report was posted on the workshop webpage so the participants were able to review the entire set of papers before attending the workshop.

Additional information can be found on the workshop web page:
http://www.cs.ncl.ac.uk/~alexander.romanovsky/home.formal/ehoos2005.html

3 List of the Workshop Presentations

The workshop program included two invited lectures:

- The first lecture on *Disciplined Exceptions* was presented by Bertrand Meyer (ETH Zurich and Eiffel Software)
- The second lecture entitled *Exception Handling: The Case Against* was presented by Andrew P. Black (Portland State University).

Besides, the following position papers were presented and discussed:

1. Denis Caromel, Guillaume Chazarain (INRIA Sophia Antipolis). *Robust Exception Handling in an Asynchronous Environment*
2. Chen Fu, Barbara G. Ryder (Rutgers University). *Testing and Understanding Error Recovery Code in Java Applications*
3. Stijn Mostinckx, Jessie Dedecker, Tom Van Cutsem, Wolfgang De Meuter (Vrije Universiteit). *Conversations for Ambient Intelligence*
4. Jan Ploski, Wilhelm Hasselbring (University of Oldenburg). *The Callback Problem in Exception Handling*
5. Matti Rintala (Tampere University of Technology). *Handling multiple concurrent exceptions in C++ using futures*
6. Michael J. Zastre, R. Nigel Horspool (University of Victoria). *Two Techniques for Improving the Performance of Exception Handling*
7. Alexei Iliasov, Alexander Romanovsky (University of Newcastle upon Tyne). *CAMA: Structured Coordination Space and Exception Propagation Mechanism for Mobile Agents*
8. Alfredo Capozucca, Barbara Gallina, Nicolas Guelfi, Patrizio Pelliccione (University of Luxembourg). *Modeling Exception Handling: a UML2.0 Platform Independent Profile for CAA*
9. Donna Malayeri, Jonathan Aldrich (Carnegie Mellon University). *Practical Exception Specifications*
10. Anand Tripathi, Devdatta Kulkarni, Tanvir Ahmed (University of Minnesota). *Exception Handling Issues in Context Aware Collaboration Systems for Pervasive Computing*
11. Luis E. Leyva del Foyo, Pedro Mejia-Alvarez, Dionisio de Niz (CINVESTAV-IPN and ITESO). *Aligning Exception handling with design by contract in embedded real-time systems development*
12. Fernando Castor Filho, Cecilia Mary Fischer Rubira, Alessandro Garcia (University of Campinas and Lancaster University). *A Quantitative Study on the Aspectization of Exception Handling*
13. John Tang Boyland (University of Wisconsin-Milwaukee). *Position Paper: Handling "Out Of Memory" Errors*

4 Summary of Invited Presentations and Discussions

Professor Bertrand Meyer of ETH Zurich and Eiffel Software presented an invited lecture in the morning. It was a privilege for us to have Bertrand Meyer discuss the

Eiffel exception handling system in the scope of this workshop. The title of his talk was *Disciplined Exceptions*. Following is the abstract of his lecture:

> *The general idea behind exceptions is to notify programs of abnormal cases occurring execution, and allow them to recover. The proper use of exceptions, and the proper design of an exception mechanism in a programming language, require a precise definition of what makes a case "abnormal". The Eiffel approach follows from an analysis of these issues, based on the concept of contract. Essentially, a contract violation causes the failure of an operation, which in turns interrupts the current execution by triggering an exception. The exception handling mechanism is also a consequence of these observations. Starting from these ideas, this talk presented a general enquiry into the notion of error, and examine how one should handle exceptions in concurrent object-oriented programming*

The Eiffel exception handling system has largely influenced the field by its originality and semantic simplicity. When it has been proposed around 1987, it was conceptually quite different by many aspects from those of ADA, CLU, Common-Lisp (Flavors) or Smalltalk that represented the well-known solutions at that time. It has generated numerous studies, papers and comparisons. It is based on the Eiffel centric notion of contract and on the constructs for assertions. The central new idea was that exceptions should not be signaled by programmers but should be the result of any attempt to break the software contract defined by a routine. This can be either because an inner operation failed, because a pre or post condition or any assertion was not fulfilled, or because of an access to a void reference. The Eiffel rescue clause is an equivalent of a classic handler the scope of which is the routine body. A rescue clause can modify the routine state so as to terminate it correctly or to retry it.

In his presentation Dr. Meyer recalled the rationale for his specification and gave some reports on the use of the system that have convinced him that the solution is operational. There are not so many other exception handling system that are in use, without major changes, for almost twenty years. Dr. Meyer also gave some indications on his new works on partial operations and on attached types to avoid void references.

In the afternoon, Pr. Andrew Black from Portland State University presented the second invited lecture of this workshop. The topic of his talk was *Exception Handling: The Case Against*. Professor Blake was the Program Chair of this year's ECOOP, and it was a unique opportunity for our workshop to ask him to give his point of view on exception handling, a subject that was the heart of his doctoral dissertation as described in the abstract of his talk.

> *In the early 1980s, Andrew Black wrote his doctoral dissertation at Oxford University; the title was "Exception Handling: The Case Against". The thesis was in part a reaction to the growing complexity of language design, epitomized by the contemporary proposals for what became Ada. The case that the thesis made against exception handling was that (a) exceptions are not an abstract concept capable of rigorous definition, but a subjective classification of program behaviour, and that (b) such a classification could usually be carried out by*

general-purpose language constructs more effectively than by a special purpose exception handling mechanism. In this talk, Andrew Black, re-examined this argument in the light of more than twenty years of experience.

Prof. Andrew Black presented quite a controversial point of view that exceptions, or more precisely specific exception handling systems, are unnecessary and undesirable. They are *unnecessary* because exception handling specific constructs could be provided or subsumed by less specific ones, for example because domain exceptions (*zeroDivide*, *popEmptyStack*) can be avoided with conditionals or handled differently with partial domains (two different methods *pop* on classes *EmptyStack* and *NonEmptyStack*), because range exceptions (*underflow*, *overflow*) can be dealt with without specific constructs thanks to union types , because handlers can be defined by standard inline functions passed to standard higher order ones (as in the *find:ifAbsent:* method of the Smalltalk Collection class) or because the only things that should be specifically handled is not what is exceptional but what is unexpected, i.e. deviation from specifications, and that exception handling systems is not the appropriate construct to do it. It is *undesirable* because exception handling constructs introduce difficulties with programming languages semantics and use, for example because they make a language more complex and more difficult to implement, because they reintroduce various forms of non local exits or because recursive exception handing (exception raised within handlers) is complex.

This presentation has put on the table a large number of the issues that designers of programming languages and/or of exception handling systems should have in mind when thinking to exceptions. It also has, as expected, generated a very large number of questions and reactions that there is no room here to report exhaustively because they involve a lot of concept, of existing languages and solutions for exception and failure handling. It has been said for example that partial domain and union types are constructions absent of most of languages and difficult to fully integrate, that the code that has to be written with union types to handle exceptional cases is at least as long and complicated that the one with try-catch like primitives, that higher-order functions do not prevent programmers to from writing non local exit, that some language (CommonLisp, Smalltalk) have at the same time, higher-order functions and exception handling systems or that there already exist exception handling systems without any specific construct (Beta). To us, one key conclusion of the discussion is the absolute need for a standardization of the terminology related to what is unpredictable, unexpected or exceptional in a program execution. This seems possible. To exactly find and agree on what should be done and how an programs when such things happen is more difficult; but that exception of failure or catastrophes (whatever the term) can be handled without specific languages constructs is clearly something that is wanted.

The participants found this workshop productive and positive in supporting exchange of ideas focusing on research issues and challenges in this area. It also reinforced everyone's view that such a meeting is of a tremendous value to the researchers and engineers developing reliable software as well as to the research community involved in object-oriented and component-based development.

Acknowledgements: We would like to thank all the participants for making this workshop a success with their unique contributions. We are grateful to Andrew Black and Bertrand Meyer for the thoughts-provoking invited talks. This report reflects the viewpoints and ideas that were contributed and debated by all these participants. Without their contributions, neither the workshop nor this report would have materialized.