Exception Handling in Object Oriented Systems: Towards Emerging Application Areas and New Programming Paradigms

Alexander Romanovsky[1], Christophe Dony[2], Anand Tripathi[3] and Jørgen Lindskov Knudsen[4]

[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

[4]Mjølner Informatics A/S
Helsingforsgade 27
DK-8200 Århus N, Denmark
jlk@mjolner.dk

Abstract. Exception handling continues to be a challenging problem in object oriented system development. One reason for this is that today's software systems are getting increasingly more complex. Moreover, exception handling is needed in a wide range of emerging application areas, sometimes requiring domain-specific models for handling exceptions. Moreover, new programming paradigms such as pervasive computing, service oriented computing, grid, ambient and mobile computing, web add new dimensions to the existing challenges in this area. The integration of exception handling mechanisms in a design needs to be based on well-founded principles and formal models to deal with the complexities of such systems and to ensure robust and reliable operation. It needs to be pursued at the very start of a design with a clear understanding of the ensuing implications at all stages, ranging from design specification, implementation, operation, maintenance, and evolution. This workshop was structured around the presentation and discussion of the various research issues in this regard to develop a common understanding of the current and future directions of research in this area.

1 Summary of Objectives and Results

There are two trends in the development of modern object oriented systems: they are getting more complex and they have to cope with an increasing number of exceptional situations. The most general way of dealing with these problems is by employing exception handling techniques. Many object oriented mechanisms for handling excep-

tions have been proposed but there still are serious problems in applying them in practice. These are caused by

- complexity of exception code design and analysis
- not addressing exception handling at the appropriate phases of system development
- lack of methodologies supporting the proper use of exception handling
- not developing specific mechanisms suitable for particular application domains and design paradigms.

Following the success of ECOOP 2000 workshop[1], this workshop aimed at achieving better understanding of how exceptions should be handled in object oriented (OO) systems, including all aspects of software design and use: novel linguistic mechanisms, design and programming practices, advanced formal methods, etc.

The workshop provided a forum for discussing the unique requirements for exception handling in the existing and emerging applications, including pervasive computing, ambient intelligence, the Internet, e-science, self-repairing systems, collaboration environments. We invited submissions on research in all areas of exception handling related to object oriented systems, in particular: formalisation, 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 (Java, Ada, Smalltalk, Beta), open software architectures, aspect oriented programming, fault tolerance, component-based technologies.

We encouraged participants to report their experiences of both benefits and obstacles in using exception handling, reporting, practical results in using advanced exception handling models and the best practice in applying exception handling for developing modern applications in the existing practical settings.

The workshop was attended by 18 researchers who participated in the presentation and discussion of ten position papers and one invited talk. These presentations and discussions were grouped into four thematic sessions. The first session (the invited talk and one presentation) addressed engineering systems incorporating exception handling. The focus of the second session (three presentations) was on the specific issues related to object-orientation. In the third session (three presentations) issues related to building novel exception handling mechanisms for distributed and mobile systems were discussed. The topic of the fourth session (three presentations) was exception handling and component based system development.

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

[1] A. Romanovsky, Ch. Dony, J. L. Knudsen, A. Tripathi. Exception Handling in Object Oriented Systems. In J. Malenfant, S. Moisan, A. Moreira. (Eds.) "Object-Oriented Technology. ECOOP 2000 Workshop Reader". LNCS-1964. pp. 16-31, 2000.

understanding exception handling: Why programmers and practitioners often believe that it complicates the system design and analysis? What should be done to improve the situation? Why exception handling is the last mechanism to learn and to use? What is wrong with the current practice and education?

We invited the researchers interested in this workshop to submit their position papers aiming at understanding why exception handling mechanisms proposed and available in earlier OO languages (discussed, for example, at ECOOP 1991 Workshop on Exception Handling and Object-Oriented Programming[2]) are not widely used now. We were interested in papers reporting practical experiences relating both benefits and obstacles in using exception handling, experience in using advanced exception handling models, and the best practices in using exception handling for developing modern applications in existing practical settings.

The original plan was to have up to 20 participants. We asked each participant to present his/her position paper, and discuss its relevance to the workshop and possible connections to work of other attendees. The members of the organizing committee reviewed all submissions. The papers accepted for the workshop sessions were 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/ehoos2003.html

The proceedings of the workshop are published as a technical report TR 03-028 by Department of Computer Science, University of Minnesota, Minneapolis, USA:
A. Romanovsky, C. Dony, J. L. Knudsen, A. Tripathi. *Proceedings of the ECOOP 2003 Workshop on Exception Handling in Object-Oriented Systems: Towards Emerging Application Areas and New Programming Paradigms. 2003.*

3 List of the Workshop Presentations

The workshop started with an invited talk delivered by William Bail (Mitre) on *Getting Control of Exception.*

After that following position papers were discussed:

1. Ricardo de Mendonça da Silva, Paulo Asterio de C. Guerra, and Cecília M. F. Rubira (U. Campinas, Brazil). *Component Integration using Composition Contracts with Exception Handling.*

2. Darrell Reimer and Harini Srinivasan (IBM Research, USA). *Analyzing Exception Usage in Large Java Applications.*

---

[2] Dony, Ch., Purchase, J., Winder. R.: Exception Handling in Object-Oriented Systems. Report on ECOOP '91 Workshop W4. OOPS Messenger **3**, 2 (1992) 17-30

3. Peter A. Burh and Roy Krischer (U. Waterloo, Canada). *Bound Exceptions in Object Programming*.

4. Denis Caromel and Alexandre Genoud (INRIA Sophia Antipolis, France). *Non-Functional Exceptions for Distributed and Mobile Objects*.

5. Tom Anderson, Mei Feng, Steve Riddle, and Alexander Romanovsky (U. Newcastle, UK). *Error Recovery for a Boiler System with OTS PID Controller*.

6. Joseph R. Kiniry (U. Nijmegen, Netherlands). *Exceptions in Java and Eiffel: Two Extremes in Exception Design and Application*.

7. Giovanna Di Marzo Serugendo (U. Geneva, Switzerland) and Alexander Romanovsky (U. Newcastle, UK). *Using Exception Handling for Fault-Tolerance in Mobile Coordination-Based Environments*.

8. Frederic Souchon, Christelle Urtado, Sylvain Vauttier (LGI2P Nimes, France), and Christoophe Dony (LIRMM Montpellier, France). *Exception Handling in Component-based Systems: a First Study*.

9. Johannes Siedersleben (SD&M Research, Germany). *Errors and Exceptions – Rights and Responsibilities*.

10. Robert Miller and Anand Tripathi (U. Minnesota, USA). *Primitives and Mechanisms in the Guardian Model for Exception Handling in Distributed Systems*.

4 Summary of Presentations

The first sessions focused on software engineering issues in exception handling. It included two talks. William Bail (Mitre) presented the invited lecture titled *Getting control of exceptions*. In his talk he noted that the past developments in this field have allowed programmers to define and use exceptions and this has led a significant advantage in being able to write more reliable software. While not explicitly helping us avoid errors, they enable us to detect their presence and control their effects. Yet they act in opposition to much of what we have learned is good software design - simple structures with well-defined control flows. In addition, they complicate the process of performing formal analyses of systems. This talk elaborated on this issue and projected some potential ideas to help reconcile these challenges, especially with the use of OO concepts. The second talk in the first session was given by Johannes Siedersleben (SD&M Research, Germany). He presented the paper *Errors and Exceptions - Rights and Responsibilities*. The talk emphasized the strict separation of errors to be handled by the application and the true exceptions which require recovering and restart mechanisms. It suggested the use of the term "emergency" for the exceptions of the second type because in many programming languages, exceptions can and are used for many non-exceptional situations. The paper also describes a component-based strategy to handle emergencies using so called safety facades.

The theme of the second session centered on exception handling in OO Systems. It included three papers. The first talk in this session was by Harini Srinivasan (IBM Re-

search, USA), who presented the paper *Analyzing Exception Usage in Large Java Applications*. This talk emphasized that proper exception usage is necessary to minimize time from problem appearance to problem isolation and diagnosis. It discusses some common trends in the use of exceptions in large Java applications that make servicing and maintaining these long running applications extremely tedious. The talk also proposes some solutions to avoid or correct these misuses of exceptions. The second presentation was by Roy Krischer (U. Waterloo, Canada) on the paper entitled *Bound Exceptions in Object Programming*. Many modern object-oriented languages do not incorporate exception handling within the object execution environment. Specifically, no provision is made to involve the object raising an exception in the catching mechanism in order to allow discrimination among multiple objects raising the same exception. The notion of bound exceptions is introduced, which associates a 'responsible' object with an exception during propagation and allows the catch clause to match on both the responsible object and exception. Multiple strategies for determining the responsible object were discussed in this talk, along with extending bound exceptions to resumption and non-local propagation among coroutines/tasks. The third speaker in this session was Joseph R. Kiniry (U. Nijmegen, Netherlands) who presented his paper *Exceptions in Java and Eiffel: Two Extremes in Exception Design and Application*. His focus was on an analysing the exception handling mechanisms in the Java and Eiffel languages and on contrasting the style and the semantics of exceptions in these two languages. The talk showed how the exception semantics impacts programming (technically) and programmers (socially). According to the author the primary result of this analysis is that Java checked exceptions are technically adequately designed but are socially a complete failure. This position is supported by an analysis of hundreds of thousands of lines of Java and Eiffel code.

The third session had three talks on exception handling in mobile and distributed systems. Alexandre Genoud (INRIA Sophia Antipolis, France) presented the paper *Non-Functional Exceptions for Distributed and Mobile Objects*. He proposed the notion of non-functional exceptions to signal failures occuring in non functional properties (distribution, transaction, security, etc.). He described a hierarchical model based on mobile exception handlers. Such handlers, attached to distribution-specific entities (proxies, futures), are used to create middleware-oriented handling strategies. The handling of exceptions can indifferently be at non-functional or application-level. The second speaker in this session was Alexander Romanovsky (U. Newcastle upon Tyne, UK), who presented the paper *Using Exception Handling for Fault-Tolerance in Mobile Coordination-Based Environments*. Mobile agent-based applications very often run on a mobile coordination-based environment, where programs communicate asynchronously through a shared memory space. The aim of this paper is to propose an exception handling model suitable for such environments. It is our view that it is conceptually wrong to treat such exceptions as usual events or tuples. This is why in the model proposed a local handler agent is created each time when an exception is signalled: this guarantees handling, allows exceptions and handlers to be dynamically associated and decreases the overall overheads. The third talk in this session was presented by Anand Tripathi (University of Minnesota, Minneapolis, USA) on *Primitives and Mechanisms in the Guardian Model for Exception Handling in Distributed Systems*. In this talk he elaborated on notion of the guardian for encapsulating exception handling policies in a

distributed application. He presented the core set of primitives of the guardian model which allow the programmer to specify and control the recovery actions of cooperating processes so that each process performs the required exception handling functions in the right context. This talk elaborated on how the various other existing models for distributed exception handling can be implemented using the guardian model.

The fourth session in the workshop focused on exception handling issues related to software and systems composition. Paulo Asterio de C. Guerra (U. Campinas, Brazil), presented the paper *Component Integration using Composition Contracts with Exception Handling*. He outlined an architectural solution for the development of dependable software systems out of concurrent autonomous component-systems. The solution is based on the concepts of coordination contracts and Coordinated Atomic (CA) Actions, which are adapted to a service-oriented approach. The second talk in this session was by Mei Feng (U. Newcastle upon Tyne, UK) on the paper *Error Recovery for a Boiler System with OTS PID Controller*. The talk presented the protective wrapper development for the model of the system in such a way that they allow detection and tolerance of typical errors caused by unavailability of signals, violations of range limitations, and oscillations. In the presentation the case study demonstrated how forward error recovery based on exception handling can be systematically incorporated at the level of the protective wrappers. The last talk of this session was given by Christelle Urtadeo (LGI2P Ecole des Mines d'Ales, Nimes, France) on *Exception Handling in Component Based Systems: A First Study*. Christelle Urtado presented a preliminary study on exception handling in component-based systems written by F. Souchon, C. Urtado, S. Vauttier and C. Dony. The talk focused on the category of components that interact in a contract-based manner and communicate asynchronously. According to the authors, exception handling for such components should provide four features: handler contextualization, concurrent activity coordination, exception concertation and exception handling support for broadcasted requests. These requirements have already shown to be pertinent in a similar context: the SaGE exception handling system that has been designed by the authors for multi-agent systems. The speaker has used SaGE implementation to exemplify how exception handling should be managed for the considered components.

5 Summary of Discussions

The workshop has gathered a rich collection of contributions covering many of the subjects that constitute the exception handling and fault tolerance domains. Presentations had generated numerous questions and exchanges. The main goal of the concluding 45 minutes general discussion was to bring to the fore the main conclusions, results, challenging issues and research directions implicitly or explicitly expressed during papers presentations and discussions on the four main subjects addressed during the workshop:

- Software engineering issues in exception handling,
- Exception handling issues in today's standard object-oriented systems
- Reliable mobile, distributed, concurrent systems
- Reliable component-based systems.

Discussion on the first issue led the attendees to a primary and major conclusion that after almost thirty years of research and many years of experimentation with many different languages, it appears that our knowledge of language primitives for exception handling is somehow high but that there is a huge need for standard definitions and for standard analysis, design and programming patterns. Indeed, the primitive for exception handling in today's languages are, in one way or another, evolutions of those proposed in the seminal paper by John Goodenough written in1975. They are primitives for signaling, catching, and handling of exception based different execution models such as entailing termination (or retry), resignaling (or propagation), or resumption. As far as these crucial concepts have been correctly extracted from the foundational research, important progress have been made in their understanding, adaptation, development and implementation. Future research efforts will need to adapt these primitives to tomorrow's needs. Unfortunately, there has never been a basic agreement, a norm, on the definition of the terms "exception", "exception handling", "fault tolerance".

Early terms such as "domain", "range" or "monitoring" exceptions as proposed in Goodenough's paper are not standardised mainly because they do not reflect the concept complexity. Researchers thus have to choose or to re-invent ever again their own definitions : consider the following terms, all coming from previous works, which have been used in our workshop contributions to denote either the same thing or subtly different things : exception, error, warning, condition, alarm, emergency, etc. There is neither a general agreement on the standard patterns to handle exceptions or to write fault-tolerant or defensive programs. When architectural solutions exist, they are not known of today's developers because there is no reference book where such patterns are described. It is interesting to note that this issue was quoted as an open issue in our ECOOP 2000 workshop conclusion, and that no significant advance has been made in that direction. However, this year, new interesting papers have reported experiences on how developers, either experienced or not, deal with exceptions, how they sometimes reinvent known solutions and make known mistakes, how they sometimes misuse or misunderstand language constructs. Everybody has thus agreed to stress the need for dedicated design patterns and to present this as a major issue.

The second main point in the discussion concerned the use of exception handling techniques in today's object-oriented systems in general and more particularly in Java as far as this language has been at the heart of the debates. Java is today "de facto" a vast field of experimentation for exception handling because it makes it mandatory for programmers to deal with exceptions, especially for the so-called "checked exceptions" (see section 4). Dealing with exception at a large scale, as experienced for example by Ada developers, had never been done by object-oriented developers because no language before Java mandated it. As William Bail noticed in the workshop introductory talk, the history of exceptions is a love/hate relationship and programmers generally do not like to handle exception for various good or wrong reasons: it makes programs longer to write, breaks code harmony, is boring, seems useless, or seems to slow execution time down.

The discussion thus focused on issues connected to Java design choices and constraints (e.g. static typing) related to exception handling. Some of these issues are already well known and sometimes related solutions exists and have been published years ago; they have however been considered here in the light of new experience reports. A first one is that handling (putting the system back into a coherent state within catch clauses body) cannot be performed in a generic way by sending messages to the exception object. Examples have been presented showing how it can impose to write several catch clauses where one would be enough instead. Besides, the exception objects do not contain enough information.

More generally, and William Bail also indirectly quoted this in his introductory talk, Java's exception handling system is certainly and for many reasons not enough object-oriented. A connected problem is that a *try block* with an empty *catch clauses* (entailing termination) can be understood by beginners, or used by developers in a final project stage, as doing nothing else than magically suppressing compiler errors. This remark introduced the main point of the debate on checked and unchecked exceptions. The question has been raised to know whether to force programmers to trap checked exceptions or to declare them in methods signature is not the cause of many misuses of exception handling in Java. For example, the systematic use of empty catch clauses has been itself reported as a major reason of debugging difficulties in some late large-scale projects. We have certainly not given the final answer to that question. The idea of checked exceptions, introduced by Liskov and Snyder in CLU, seems in itself very coherent, rigorous and fruitful but Java's experience tends to show that it has initially unpredicted consequences. As far as such control on software code is wanted, the alternative solution proposed in Beta to handle with different mechanisms, one static and one dynamic, failures (conceptual equivalent of Java's unchecked exceptions) and exceptions (checked ones) should be considered. The evolution of exception handling models for standard OO programming is still an open issue.

The ECOOP 2000 workshop conclusion stressed the need for new exception handling linguistic mechanisms and patterns that take into account new or difficult programming paradigms used to develop mobile, distributed, concurrent or component based systems. Discussions on those points began by stating that workshop contributions revealed significant advances, primarily for what concerns distributed systems where several deep and complete studies have been achieved. However, many open issues and problems require further investigation into practical applications of some of the emerging models. The most fundamental issues with exception handling in distributed and concurrent systems are related to policies and mechanisms for resolution of concurrent exceptions, determining which exceptions should be signaled in a process that is required to participate in recovery, and the context in which a process should handle the exception delivered to it. This requires models and mechanisms to ensure that each process handles a global exception in the proper context. There are several existing models that facilitate this by imposing certain program structuring disciplines such as conversations or transactions. The Guardian model can be considered as a basic model, addressing some of the basic issues to serve as meta-level model that can be used to implement other models, including the existing ones. However, more detailed and practical studies are needed in this area.

Mobile and component-based systems and applications raise newer, difficult and interesting issues for which propositions or state-of-the-problem have been described. The primary issues arise due to a broad class of exceptions that can arise due to distribution, asynchronous interactions among components, and due to a relatively large number of unanticipated conditions in which the component of an application could be used. Mobile agents represent one class of components, which are active (objects) that are capable of migrating from one execution environment to another. A mobile agent (object/component) executing in a remote environment" may not be able to determine and handle the context of an exception when situated in a remote environment. A number of models and approaches were discussed in this workshop. These include separation between functional and non-functional exceptions, and support for exception handling systems with asynchronous communication based on tuple spaces.

Component programming models extend object-oriented programming and to correctly integrate exception handling and fault tolerance in the design and implementation of new component languages raise issues at all stages. If existing propositions are to be considered, those issues also are as different as today's models. Component can for example be distinguished by the kind of interfaces they propose, the way they interact (e.g. contract-based or event-based interactions) and the way they communicate (e.g. synchronous or asynchronous communications). Some hard point brought to the fore by the discussions, for which extended research efforts will certainly be needed, are for example validation of composites made from components able to signal exceptions or the control of asynchronous distributed components such as Java's message-driven beans. Component based software engineering methods are going to be central in building pervasive computing systems and "smart environments" whose intrinsic model of computation and user-interactions is built upon supporting dynamic discovery of services/components and their interactions with mobile user devices. Fault-tolerant and safe operations of such systems will require addressing of exceptions handling issues of even greater complexities than those being encountered and addressed in today's systems.

6 Conclusions

Exception handling systems are designed to enhance software reliability, reusability, readability and debugging. During the 1980s and 1990s, exception handling systems were integrated into all object-oriented languages.

The first ECOOP workshop on exception handling and OO programming held in 1991 was oriented towards the specification, understanding and implementation of the first versions of object-oriented exception handling systems which can be classified into various families. The Smalltalk or Clos family which proposed dynamically-typed, very powerful in term of expressive power, fully object-oriented, systems represented in fine by the ANSI Smalltalk Standard in 1997. The C++ and Java family, representing the 1995-2005 tendency, with less expressive power and various restrictions which are either design choices (e.g. Checked exceptions) or consequences of the quest for static and correct typing (e.g. restrictions on interface specialization). Eiffel or Beta

proposed important variations on exception handling in the same context of statically-typed languages.

The second ECOOP workshop on exception handling, held in 2000 in Nice, dealt less with standard languages implementation, it brought to the fore a large panel of open and challenging problems: models for standard languages are were not entirely satisfactory, many important past contributions were forgotten in standard languages, there were a well identified set of research works advocated towards (1) exception handling systems safe and compatible with static and correct typing, (2) new linguistic mechanisms to take into account concurrent, web, mobile, distributed systems, (3) the need for disciplined exception handling at all phases of system development and (4) the need for good development techniques with well-developed architectural and design patterns.

This workshop, three years later, has been a clear success. The introduction talk ideally put the things in situation by recalling basic knowledge and challenges. The range of addressed topics has been wide and the technical level of presentations and discussions high, either for what concerns research contributions or experience reports.

- For the first time we have had detailed and precise developers experience reports on exception handling in object-oriented languages. They have clearly shown us on the one hand that some of Java design choices or typing constraints really induce problematic programming practice that should influence the design of future versions or of future standard languages. On another hand, they also clearly show that there is a huge need, as quoted in 2000, for a book on exception handling design patterns. This now is a crucial point for an actual development of fault tolerant programming.
- Advances, sometimes very important, have been made in some of the problem quoted as challenging in 2000. This is the case for exception handling in distributed, concurrent, asynchronous-communication based or mobile systems. New ideas have been proposed on various fields as e.g. a new original language-level basic mechanisms (so-called bound exceptions, see section 4).
- Finally, many issues remain or have appeared. (1) For example, as far as exception handling is not just a linguistic issue, and we still do not have many models and tools to deal with reliability at all stage of software development. (2) Today's tendency still is to use statically-typed OO languages, albeit other voice still can be heard, but problems related to static-typing are not solved in Java, languages that propose to combine inclusion polymorphism and contravariance (Beta) are not studied enough and multiple dispatch is either too slow or is too space-consuming. (3) The separation of concerns in the case of exception handling would be a great advance but a ECOOP 2000 paper has shown that it was a difficult and controversial issue. Much more work should be put on that point. (4) The representation of exceptions by hierarchically organized classes is now quite a standard but the design issues related to that representation are still not solved and design languages such as UML do not correctly take it into account. There is a lack of proper frameworks and principles for designing and analyzing exception hierarchies and exception code in large-scale systems.

- New difficult issues, connected to the development of new technologies, have been brought to the fore, especially for what concerns component-based systems. It has been shown, for example, that current component systems architectures exception handling systems are quite poor and limited.

This workshop has really been a positive event, and it has reinforced everyone's view that such a meeting is of a tremendous value to the reliable software development and research community involved in object-oriented and component-based development.

Acknowledgements: We would like to thank the participants for making this workshop a success with their unique contributions. 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.