# Modular Exception Handling in Context-Aware Applications

Nelio Cacho
Computing Department, Lancaster University
United Kingdom
n.cacho@lancaster.ac.uk

Alessandro Garcia
Computing Department, Lancaster University
United Kingdom
a.garcia@lancaster.ac.uk

## ABSTRACT

Exception handling systems are designed to promote enhanced program modularization in the presence of faults, by supporting a well-defined lexical separation of normal and abnormal code. However, mobile applications seem to require specialized exception handling mechanisms due to their features of context-awareness and asynchronous communication. Hence this research investigates the impact of such features in the proper definition of an exception handling system. In addition, innovative aspect-oriented abstractions and mechanisms will be defined and assessed in the implementation of modular context-aware exception handling.

## Keywords

Exception handling, context-awareness, middleware, fault tolerance, aspect-oriented programming.

## 1. INTRODUCTION

The recent advances on mobile computing have enabled the development of a wide variety of context-aware applications, which are able to monitor and exploit dynamically-changing contextual information from users and surrounding environments. As a result, a number of middleware systems and programming languages have started to emerge to support the implementation of context-aware modules [13, 22, 23]. Nevertheless, with mobile software systems becoming applicable in many sectors, such as ambient intelligence, manufacturing, and healthcare, they have to cope with an increasing number of erroneous conditions and satisfy stringent dependability requirements on fault tolerance, integrity, and availability.

In order to deal with erroneous conditions, exception handling techniques [7, 12] are the conventional solutions to provide the fundamental abstractions and mechanisms for constructing robust modular systems. They ensure system modularity in the presence of errors by offering abstractions for: (i) representing *erroneous situations* of system modules as exceptions, (ii) encapsulating exception handling activities into *handlers*, (iii) defining parts of the system modules as *protected regions* for handling exception occurrences, (iv) associating these protected regions with handlers, and (v) explicitly specifying *exceptional interfaces* of modules.

In addition, exception handling systems promote more reliable programming by hiding from programmers a plethora of complexities relative to error handling. They provide a core set of mechanisms for managing the exceptional control flow. First, they supply explicit support for both systemic and application-specific exception detection. Second, they implement automatic deviation from the normal control flow to the exceptional one, thereby automatically searching for suitable handlers upon exception occurrences. Once handlers are successfully executed, the system is seamlessly returned to its normal operation.

However, the design of exception handling mechanisms needs to be tightly integrated with the underlying module system. In fact, exception handling mechanisms have systematically changed over the last three decades according to the appearance of new programming paradigms and associated communication patterns [2]. Therefore as the advent of context-oriented middleware [10, 25] and programming languages [13, 22, 23] is impacting on the way system modules are structured and interact with each other, suitable exception handling abstractions and mechanisms need to be investigated. In this context, our research is focused on understanding how exception handling is impacted by context-aware systems and its unreliable wireless communications.

This paper is structured as follows. Section 2 describes the open issues in designing exception handling systems suitable for the development of robust context-aware applications. Section 3 lists the expected contributions of this research. Section 4 discusses related work. Section 5 describes our hypothesis and the methodology to achieve our goals. Section 6 presents the achieved preliminary results.

## 2. PROBLEM DESCRIPTION

Context-aware applications adapt its behavior on the basis of user and environmental changes [5]. The treatment of exceptional conditions in context-aware mobile applications seems to indicate a number of open issues related to two fundamental parts of an exception handling mechanism: (i) modularity issues, and (ii) the management of the exceptional control flow. Modularity problems may be in turn classified in different categories according to the previously mentioned exception handling abstractions (Section 1), as follows:

- *Representing and detecting erroneous conditions*. As the characterization of an error itself may depend on the contextual conditions, proper abstractions are required to support the definition of exceptions. A new representation of exceptions is also required because the exceptional application contexts may be a result of the composition of concurrent events in the code embedded in multiple mobile devices. Moreover, the code needed to check invariants and detect context-sensitive exceptions is typically scattered all over the program modular units.
- *Protected regions*. Conventional *try-catch* blocks provide a certain lexical separation between the normal and exceptional code. However, other types of protected regions

may need to be associated with other levels of abstraction in a mobile system, such a group of mobile devices located at the same physical location. These other types of exception handling scopes are essential to support the proper propagation of exceptions at run-time, as devices can asynchronously enter and leave a certain physical region.

- *Handlers.* The exception handling activities should be well encapsulated in separate modules in order to reduce their coupling to the normal-processing code. Also, some exception handlers may also be activated only upon certain contextual situations. As a result, handler interfaces need to explicitly expose those contextual conditions in which they should be triggered.

- *Exceptional interfaces.* In traditional distributed systems, the exception interface associated with each component is well known in advance. However, behaviors of context-aware modules are activated and deactivated in the control flow of a running program [22]. When a behavior is activated due to relevant contextual variations, such partial definitions become part of the module until this behavior is deactivated [22]. This dynamic composition of the module interface makes unclear how the exceptional interface of a context-aware module should be specified.

In addition to the modularity problems, the unreliable wireless communication in mobile applications makes the management of exceptional control flow more difficult. The main reason is that connection instability impedes the utilization of synchronous communication protocols in order to deviate from the normal to the exceptional control flow. On the other hand, exception handling mechanisms in object-oriented languages [2] and their antecessors have traditionally relied on synchronous communication forms. Synchronous communication patterns make the deviation of the normal control flow easier. In the presence of asynchrony, it is not clear which program modules should be notified upon an exception occurrence.

## 3. CONTRIBUTIONS

This work is completing one year of research in November 2006. This research addresses the identification of the impacts caused by context-aware mobile applications on the design of an exception handling mechanism. The primary and original contribution of this research is the proposal of an exception handling model that introduces a set of abstractions supporting development of robust context-aware applications. These abstractions can be used by developers to design well-structured resilient context-aware applications which interlink in an intuitive and effective ways separated descriptions of the normal and abnormal control flow. These abstractions should be integrated with an existing context-aware middleware [10, 25] or a programming language [13, 22, 23].

## 4. RELATED WORK

Since the publication of seminal papers on exception handling [7, 12], this area has received considerable attention from researchers across different communities, such as Software Engineering, Programming Languages, Dependability and Distributed Systems. As consequence, exception handling techniques for sequential programs were been initially designed to support the abstractions and mechanisms mentioned above (Section 1). Their designs have also been systematically enhanced in order to promote their integration with mainstream development paradigms, such as object-oriented [2], and characteristics of modern applications, such as concurrency and distribution [15]. For instance, exception mechanisms in distributed concurrent systems rely on advanced transaction mechanisms in order to both scope with erroneous conditions in multi-thread collaborations and support proper modular units for error confinement, handler attachments, exception resolution, and exception handling scopes.

In terms of unreliable wireless communication in mobile applications, some related areas such as exception handling in multi-agent systems there have been a number of interesting results. A scheme in [9] supports exception handling in systems consisting of agents that cooperate by sending asynchronous messages. It allows handlers to be associated with services, agents and roles, and supports concurrent exception resolution. Some exception handling issues in a context-aware application are discussed in [4]. The paper discusses issues that arise due to security and coordination particularities. Klein et al. [18] defines a specialized service fully responsible for coordinating all exception handling activities in multi agent systems. Although this approach does not scale well, it supports separation of the normal system behavior from the abnormal one as the service curries all fault tolerance activities: it detects errors, finds the most appropriate recovery actions using a set of heuristics and executes them.

Moreover, the growing of the context-awareness popularity stimulates the appearance of context-oriented languages [22, 23] which define context as a first-class construction. Despite the introduction of this new paradigm, these languages do not support context-aware exception handling mechanisms. Additionally, AmbientTalk [13], an ambient-oriented language provides an extension [24] to support ambient-oriented exception handling whose the focus is dealing with asynchronous propagation and concurrent exception resolution. Nevertheless, this approach does not address context-awareness, such as context-aware exception propagation and definition

As opposed to the previous schemes, which do not explicitly introduces the concept of the exception handling context (scope), the CAMA framework [1] supports (nested) scopes, which confine the errors and to which exception handlers are attached. However, CAMA and the other previously mentioned mechanisms do not address a fully modular, context-aware, and asynchronous features required by context-aware applications that are going to be explored by this proposal.

## 5. HYPOTHESIS AND PROPOSED METHODOLOGY

The main challenge addressed by this research is the provision of support for the modularization of context-aware exceptional code. In particular, we are going to define and assess to what extent aspect-oriented abstractions [11] allow for modular error handling in context-sensitive mobile applications. Aspect-oriented programming (AOP) is an emerging programming technique with the goal of improving the modularization of widely-scoped program concerns [14], such as exception handling. The hypothesis to be tested is that: “program modularity and

management of exceptional control flow are enhanced by aspectizing context-sensitive exception handling".

In order to attest our hypothesis, our main methodological steps are based on the initial realization of two distinct sets of experimental studies. The first group of case studies was already partially carried out and aimed at empirically assessing how AOP techniques are feasible to modularize exceptional behaviors in general. Also, the purpose is to verify how pointcut languages scale to specify context-sensitive exception detection and asynchronous deviation of the control flow. Both studies will also assist in the revealing of the exception handling requirements in terms of an aspect-oriented language definition for context-aware applications.

The initial stage of the first group of experiments [8] assessed how exception handling affects the modularity of four different conventional distributed applications. This assessment encompassed the utilization of one widely-used aspect-oriented language, namely AspectJ [3], to move *try-catch*, *try-catch-finally*, and *try-finally* to aspects in order to separate the normal and exceptional code. The quantitative evaluation of this experiment was based on the application of a modularity metrics suite [6] to both the original and modularized version of the selected applications. This suite includes metrics for separation of concerns, coupling, cohesion, and size, which have already been used in various different experimental studies [8, 21].

Hence, the purpose of this initial stage was to first establish the general scenarios where a typical AOP language scales up to promote the lexical separation between error handling code and normal code. Based on these scenarios, the next step includes performing a similar study, but now for two different context-aware mobile applications. This investigation will be useful to identify some possible further limitations of the AspectJ pointcut language in modularizing context-sensitive exceptional code. These findings will allow us to reflect the best strategy for the conception of a specialized pointcut language, alternatively based on a state-based joinpoint model [17]. The idea is to understand which potential language constructs are appropriate to support the quantification of exceptional context-specific conditions over sets of mobile devices.

The second set of empirical studies is aimed at deciding which level - programming language or middleware - will be used to implement the exception handling mechanism. As a starting point, we have selected four representative context-aware applications from different degrees of mobility, collaboration and context-awareness: an ambient intelligence system, a healthcare application, a tourist guide application, and a conference assistant application. These applications were analyzed to identify the character of the exceptions, its occurrence and policy, and whether there are limitations that are neither satisfied by regular use of exception mechanism of programming languages nor addressed by conventional mechanism of the existing context-aware middleware systems. Based on this assessment, it was possible to introduce some abstractions to support error handling in context-aware applications. These new abstractions were evaluated through the implementation of two case studies [16, 20], which use a publish/subscribe middleware to support the exceptional control flow. In order to investigate how emerging context-oriented programming languages support context-aware exception handling, we are going to implement the same case studies using two distinct languages: AmbientTalk [13] and ContextL [22].

## 6. PRELIMINARY RESULTS

Up to now, the methodology adopted in the previous section allowed us to obtain some preliminary results in terms of limitations in the analyzed programming techniques:

- *Modularity:* The first set of experiments [8] assessed the usefulness of aspects for separating exception handling code from the normal application code. We found out that, although the use of AOP to separate exception handling code can be beneficial, it depends on a combination of several factors. For instance, if exception handling code in an application is non-uniform, strongly context-dependent, or too complex, aspectization can bring more harm than good. These factors were categorized in many different scenarios which correspond to the occurrence of exception handling code. Based on this categorization, we identified many limitations of the current aspect-oriented technique that will be used to define the requirements of our modularization technique.

- *Exception control flow*: The second set of experiments [16, 20] allowed us to acquire a set of central requirements on the definition of an exception handling model for context-aware system, which consists of: (i) explicit support for specifying "exceptional contexts"; (ii) context-sensitive search for exception handlers; (iii) multi-level handling scopes that meet new abstractions (such as groups), and abstractions in the underlying context-aware middleware, such as devices, regions, and proxy servers, (iv) context-aware error propagation, (v) volatile exception interface and contextual exception handlers, (vi) proactive exception handling, and (vii) concurrent resolution of exceptional conditions.

## 7. REFERENCES


[1] A. Iliasov and A. Romanovsky. CAMA: Structured Communication Space and Exception Propagation Mechanism for Mobile Agents. ECOOP-EHWS 2005, July 2005, Glasgow.

[2] A. F. Garcia et al. A comparative study of exception handling mechanisms for building dependable object-oriented software. *The Journal of Systems and Software*, 59(2):197–222, 2001.

[3] AspectJ. *The AspectJ Guide*. http://eclipse.org/aspectj/.

[4] A. Tripathi, et al. "Exception Handling Issues in Context Aware Collaboration Systems for Pervasive Computing". In Romanovsky et al (Eds.) Proc. ECOOP 2005 Workshop on Exception Handling in Object Oriented Systems. 2005.France.

[5] B. N. Schilit, N. Adams and R. Want. Context-aware computing applications. *In Proceedings Workshop on Mobile Computing Systems and Applications.* IEEE, December 1994.

[6] C. Sant'Anna et al. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. *Proc. of Brazilian Symposium on Software Engineering (SBES'03)*, Manaus, Brazil, Oct 2003, 19-34.

[7] D. L. Parnas and H. Wurges. Response to undesired events in software systems. *In ICSE '76*, pages 437–446, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[8] F. Filho et al. Exceptions and Aspects: The Devil is in the Details. Accepted to FSE-14, International Conference on Foundations on Software Engineering, November 2006.

[9] F. Souchon et al. Improving exception handling in multi-agent systems. In C. Lucena et al (Eds), Software Engineering for Multi-Agent Systems II, number 2940. Feb. 2004.

[10] G. Cugola and J. E. M. Cote. On Introducing Location Awareness in Publish-Subscribe Middleware. *Fourth International Workshop on Distributed Event-Based Systems* (DEBS) (ICDCSW'05), 2005.

[11] G. Kiczales et al. Aspect-Oriented Programming. *Proc. of ECOOP'97, LNCS 1241*, Finland, June 1997, 220-242.

[12] J. B. Goodenough. Exception handling: issues and a proposed notation. *Commun. ACM*, 18(12):683–696, 1975.

[13] J. Dedecker et al. Ambient-oriented programming. In *Companion To the 20th Annual ACM SIGPLAN*. OOPSLA '05. ACM Press, New York, NY, 31-40.

[14] J. Viega and J. Voas. Can Aspect-Oriented Programming Lead to More Reliable Software?, *IEEE Software*, vol. 17, no. 6, pp. 19-21, Nov/Dec, 2000.

[15] J. Xu et al. "Fault Tolerance in Concurrent Object-Oriented Software through Coordinated Error Recovery," *In Proc. 25th Int. Symp. on Fault-Tolerant Computing*, pp.499-508, Pasadena, June 1995.

[16] K. Damasceno et al. Context-aware exception handling in mobile agent systems: the MoCA case. In *Proceedings of the 2006 international Workshop on Software Engineering For Large-Scale Multi-Agent Systems* (Shanghai, China, May 22 - 23, 2006). SELMAS '06. ACM Press, New York, NY, 37-44.

[17] L. D. Navarro et al. Explicitly distributed AOP using AWED. In *Proceedings of the 5th international Conference on Aspect-Oriented Software Development*. 51-62.

[18] M. Klein and C. Dellarocas. Exception Handling in Agent Systems. Proc. of the 3rd Int. Conference on Autonomous Agents, Seattle, WA, May 1-5, 1999. Pp. 62-6

[19] M. P. Robillard and G. C. Murphy. Analyzing exception flow in Java programs. *In Proceedings of the 7th European Software Engineering Conference and 7th ACM SIGSOFT* September 1999

[20] N. Cacho et al. Handling Exceptional Conditions in Mobile Collaborative Applications: An Exploratory Case Study. *4th International Workshop on Distributed and Mobile Collaboration*, 2006.

[21] N. Cacho et al. Composing design patterns: a scalability study of aspect-oriented programming. In *Proceedings of the 5th international Conference on Aspect-Oriented Software Development*. AOSD '06. 109-121

[22] P. Costanza and R. Hirschfeld. Language Constructs for Context-Oriented Programming, *In Proceedings of DLS 2005*.

[23] R. Keays and A. Rakotonirainy. Context-oriented programming. In *Proceedings of the 3rd* MobiDe '03. ACM Press, New York, NY, 9-16.

[24] S. Mostinckx et al. Ambient-Oriented Exception Handling In *Advanced Topics in Exception Handling Technique*", C. Dony, J. L. Knudsen, A. Romanovsky, A. Tripathi (eds.), Lect. Not. Comp. Sc. 4119, pp. 141-160, Springer Verlag, 2006.

[25] V. Sacramento et al, MoCA: A Middleware for Developing Collaborative Applications for Mobile Users. *IEEE Distributed Systems Online*, vol. 5, no. 10, 2004.