The Journal of Systems and Software 000 (2016) 1–16

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A Formal Approach to implement java exceptions in cooperative systems

Simone Hanazumi, Ana C.V. de Melo*

Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão 1010, Cidade Universitária, São Paulo,05508-090, Brazil

ARTICLE INFO

Article history:
Received 18 May 2015
Revised 13 July 2016
Accepted 24 July 2016
Available online xxx

Keywords:
Coordinated atomic actions model
concurrent exception handling
java framework
program verification

ABSTRACT

The increasing number of systems that work on the top of cooperating elements have required new techniques to control cooperation on both normal and abnormal behaviors of systems. The controllability of the normal behaviors has received more attention because they are concerned with the users expectations, while for the abnormal behaviors it is left to designers and programmers. However, for cooperative systems, the abnormal behaviors, mostly represented by exceptions at programming level, become an important issue in software development because they can affect the overall system behavior. If an exception is raised and not handled accordingly, the system may collapse. To avoid such situation, certain concepts and models have been proposed to coordinate propagation and recovering of exceptional behaviors, including the Coordinated Atomic Actions (CAA). Regardless of the effort in creating these conceptual models, an actual implementation of them in real systems is not very straightforward.

This article provides a reliable framework for the implementation of Java exceptions propagation and recovery using CAA concepts. To do this, a Java framework (based on a formal specification) is presented, together with a set of properties to be preserved and proved with the Java Pathfinder (JPF) model checker. In practice, to develop new systems based on the given coordination concepts, designers/programmers can instantiate the framework to implement the exceptional behavior and then verify the correctness of the resulting code using JPF. Therefore, by using the framework, designers/programmers can reuse the provided CAA implementation and instantiate fault-tolerant Java systems.

1. Introduction

The computational systems today are no longer single elements, they are mostly made of a set of components working in a cooperative way Jonkers et al. (2004); Papazoglou et al. (2008). It is a reality for all systems in Service-Oriented Architecture (SOA) Ni and Fan (2010) and for Component-based Systems Coronato and De Pietro (2010), mainly developed in Object-Oriented languages. The common concept behind all these systems is the distribution of tasks among elements that need to work together to provide systems requirements. At the same time that components cooperate to provide features that satisfy system requirements, they must give a cooperative mechanism to recover from errors Randell and Xu (1994). Guaranteeing the dependability and, consequently, the reliability of systems has become an important issue for the success of many software enterprises since unreliable systems may result in serious consequences involving material and life losses Board (2012). All this depends on good development methods and how systems failures can actually be prevented.

Systems failures can arise as a result of physical or software faults. Fault-tolerant systems can be created to deal with software faults and solutions for them can be given at both programming and architectural levels. At programming level, fault tolerance can be treated by exception handling mechanisms that provide a clear separation of codes for error recovery and normal behavior, and helps in decreasing code complexity and software design mistakes. The software systems must then comprise the *normal behavior* code, in which errors are detected and the corresponding exceptions are raised, and the *exception handler* code Cristian (1982), in which the exceptional behavior is treated resulting in the error recovery.

Despite exception handling mechanisms being a step forward for explicitly detecting and recovering systems from errors, they need to be properly used to provide fault-tolerant systems. Exceptions can slow down programs if they are not correctly used Doshi (2012): memory and processors time are required to create, throw,

* Corresponding author. Fax: +55 11 30916134.
E-mail addresses: hanazumi@ime.usp.br, acvdemelo@gmail.com (S. Hanazumi), acvm@ime.usp.br, acvdemelo@gmail.com (A.C.V. de Melo).

http://dx.doi.org/10.1016/j.jss.2016.07.033
0164-1212/

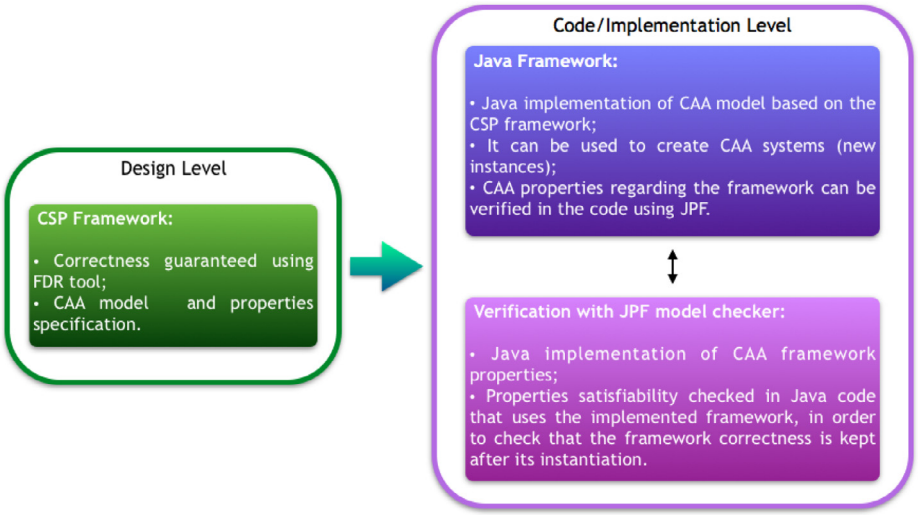

Fig. 1. Work overview: the focus is at the code/implementation level.

and catch exceptions. Also, the overuse of exceptions may obfuscate code, making it difficult to be used and the client code ignoring the raised exceptions. Programming communities have employed a great effort to establish a discipline on exception handling mechanisms use Longshaw and Woods (2004); Patterns (2012); Wirfs-Brock (2006); however no patterns handbook is established so far for them. In practice, many novice or mid-level software engineers are not able to treat faults properly McCune (2006), either because no priority is given to this activity or they do not follow a style to define, raise and handle exceptions. Some reasons can be pointed out. First, most systems do not treat their exceptional behavior at the design level. As a result, exceptions are treated by demand, in the way developers understand it must be done. Second, many companies do not adopt a style to treat exceptions and each programmer decides particular cases.

In principle, most of these problems could be sorted out at the design level, using patterns, having companies adopting certain styles and training their developers appropriately. However, these solutions are applicable to exceptions locally treated by single components while, in practice, systems today are made of a set of cooperating components. Then, faults embedded in concurrent and distributed systems, where an exception raised during an operation may cause a propagation of failures affecting the reliability of the whole system, must be managed at architectural level to prevent systems suddenly stopping an ongoing computation. In these cases, apart from locally treating failures, their propagation must be treated by the entire system. This requires models to recognize and recover from errors, providing the corresponding policies to bring the whole system to a stable state in a coordinated manner. Hence, to guide the development of fault-tolerant systems, a number of models and approaches were proposed, such as transaction Gray and Reuter (1992), conversation Randell (1975) and coordinated atomic actions (CAAs) Xu et al. (1995). CAAs have been used in real critical systems Beder et al. (2000); Capozucca et al. (2005) and provides a formal model to manage exception handling and systems recovery.

Coordinated atomic actions (CAAs) provide a conceptual model to interrelate propagation of exceptions in a cooperative manner. This model can guide users to treat exceptions in a well-organized way and maintaining the whole system in stable states. However, even if users have the intention to follow the CAA concepts, certain inconsistencies may arise depending on the system complexity. A step forward to strictly follow CAA concepts is predefining a standard architectural model through formal means Pereira and de Melo (2010). The formal strategy for specifying fault-tolerant systems helps in the identification of ambiguities, omissions and inconsistencies at the specification level, and the formal verification serves as a fault-prevention mechanism to better eliminate design mistakes. Although with the use of such a framework one can make an instance of the formal model and then check certain CAA properties, it is done at the design level and the gap of checking consistency at implementation level remains. So, an implementation of the conceptual CAA model is required. Moreover, whenever an implementation of a design model is provided, new elements need to be inserted as you go down to code. Due to these new elements in the more concrete model, certain properties guaranteed at the specification level might no longer be preserved in the code counterpart.

The main contribution of the current work is twofold (Fig. 1 - right-hand side). First, it provides an implementation of a formal model to handle exceptions at an architectural level, based on CAAs. This implementation framework can be reused and instantiated to actually provide an implementation of a coordinated exception handling mechanism for a system. Second, since new elements need to be inserted as the implementation framework is instantiated, a set of CAA properties are defined to be checked with the *Java Pathfinder* model checker (core version) Team (2012) to guarantee that the framework correctness is kept after its instantiation.

A preliminary version of this paper appeared in the proceedings of QUATIC'12 (8th International Conference on the Quality of Information and Communications Technology) Hanazumi and de Melo (2012). This paper includes the following additional contributions: a discussion regarding this paper results and related work; an overview of the CAA architectural model in CSP (Communicating Sequential Processes); a complete description of the Java framework elements and how they are related to the CAA formal model; a full example source code presenting how one can use the Java framework; details concerning to the CAA properties specification and how they were implemented in the JPF model checker.

The remaining of this paper is organized as follows: Section 2 describes related work; Section 3 presents the coordinated atomic actions concepts; Section 4 describes the CSP

Please cite this article as: S. Hanazumi, A.C.V. de Melo, A Formal Approach to implement java exceptions in cooperative systems, The Journal of Systems and Software (2016), http://dx.doi.org/10.1016/j.jss.2016.07.033