Chapter 19
A Method to Support Fault Tolerance Design in Service Oriented Computing Systems

Domenico Cotroneo
Università degli Studi di Napoli Federico II and Complesso Universitario Monte Sant’Angelo, Italy

Antonio Pecchia
Università degli Studi di Napoli Federico II, Italy

Roberto Pietrantuono
Università degli Studi di Napoli Federico II, Italy

Stefano Russo
Università degli Studi di Napoli Federico II and Complesso Universitario Monte Sant’Angelo, Italy

ABSTRACT
Service Oriented Computing relies on the integration of heterogeneous software technologies and infrastructures that provide developers with a common ground for composing services and producing applications flexibly. However, this approach eases software development but makes dependability a big challenge. Integrating such diverse software items raise issues that traditional testing is not able to exhaustively cope with. In this context, tolerating faults, rather than attempt to detect them solely by testing, is a more suitable solution. This paper proposes a method to support a tailored design of fault tolerance actions for the system being developed. This paper describes system failure behavior through an extensive fault injection campaign to figure out its criticalities and adopt the most appropriate countermeasures to tolerate operational faults. The proposed method is applied to two distinct SOC-enabling technologies. Results show how the achieved findings allow designers to understand the system failure behavior and plan fault tolerance.

INTRODUCTION
Service Oriented Computing (SOC) is emerging as a leading paradigm in the context of scalable distributed software development. As a matter of fact, it enables complex applications to be seamlessly developed by integrating services and components rather than building them entirely from scratch. This significantly reduces development cost and time to market.

To achieve flexible, interoperable, and massively distributed applications, the SOC paradigm relies on a number of support technologies, middleware platforms and ad-hoc protocols, which allow
developers to focus mainly on service development and composition. However, the integration of these heterogeneous software items raises significant dependability challenges that need to be addressed for achieving trusted SOC applications. In fact, such items are usually conceived to be integrated with other systems, hence developed without any specific operational context in mind. Consequently, testing activities carried out during their development may be not enough to guarantee a proper service during operations (Weyuker, 1998; Moraes, Durães, Barbosa, Martins, & Madeira, 2007), because of unforeseeable interactions with the rest of the system integrating them.

A viable solution to cope with this issue is tolerating residual faults rather than trying to avoid or to remove them before the operational phase. N-version programming (Avizienis & Chen, 1977; Avizienis, 1985) and recovery blocks (Randell, 1987; Kim & Welch, 1989) are well known software fault tolerance strategies; however, for many industrial purposes, these techniques have been neglected due to their high cost (Saha, 2006) and to the lack of data about their effectiveness (Anderson, Barrett, Halliwell, & Moulding, 1995). Rather than developing additional versions of the target program, focusing on the improvement of its single-version (e.g., by adding extra code to handle exceptional bad situations (Bishop & Frincke, 2004)) is a more suitable solution in the context of SOC technologies. This strategy can allow system designers to achieve a good trade-off among cost, time to market and a proper dependability level.

Designing fault tolerance actions to improve the single-version of the program presents several challenging issues. First, tracking and handling all potential conditions to be tolerated may be not feasible in practice, especially when dealing with large size software systems. Indeed, this would lead to programming cost comparable with (or even greater than) the implementation of additional versions. Moreover, a mere brute-force addition of fault-handling code may inadvertently lead to (i) the introduction of new faults and to (ii) heavy performance depletion. As a consequence, a crucial task, which cannot be driven only by the developer’s experience or attitude, is the choice of the most appropriate fault tolerance actions to apply where actually needed.

This requires a deep knowledge about the system features and its behavior in presence of faults, which probably is the real challenge to be addressed. Without such knowledge, the single-version fault tolerance approach, even if potentially more suitable than multiple-version solutions (especially for business-critical contexts), would be not applicable. This paper proposes a method to support the design of fault tolerance actions specifically tailored for the system being developed. Through an extensive software fault injection campaign we identify the most critical software modules, as well as related fault types, which heavily impact the correct behavior of a system during the operational time. This knowledge allows designers to figure out (i) where to place additional code, by exploiting the information about critical modules and (ii) what to place, in terms of the specific action to implement, by exploiting the information about the fault types. Our ultimate aim is to drive improvements by applying the proper fault tolerance action only where actually needed. Developers would thus characterize, in terms of failure behavior, the underlying context-independent software items before their deployment in a SOA, with the possibility to effectively intervene by enriching their fault tolerance ability.

The proposed method is applied to two distinct SOC enabling technologies (i) the popular Apache Web Server, whose availability level is crucial for many web-based service-oriented applications, and (ii) the TAO Open Data Distribution Service (DDS). Experimental campaigns show how these systems have been characterized through an in-depth injection of software faults, and how the achieved findings can be used to tailor the design of fault tolerance strategies to the target system.
Related Content

An OO Methodology Based on the Unified Process for GIS Application Development
[www.igi-global.com/chapter/methodology-based-unified-process-gis/28119?camid=4v1a](www.igi-global.com/chapter/methodology-based-unified-process-gis/28119?camid=4v1a)

Object-Oriented Cognitive Complexity Measures: An Analysis
[www.igi-global.com/chapter/object-oriented-cognitive-complexity-measures/117923?camid=4v1a](www.igi-global.com/chapter/object-oriented-cognitive-complexity-measures/117923?camid=4v1a)

How to Systematically Embed Cycles in Balanced Hypercubes
[www.igi-global.com/article/how-to-systematically-embed-cycles-in-balanced-hypercubes/169917?camid=4v1a](www.igi-global.com/article/how-to-systematically-embed-cycles-in-balanced-hypercubes/169917?camid=4v1a)

Web Services Description and Discovery for Mobile Crowdsensing: Survey and Future Guidelines
[www.igi-global.com/article/web-services-description-and-discovery-for-mobile-crowdsensing/178563?camid=4v1a](www.igi-global.com/article/web-services-description-and-discovery-for-mobile-crowdsensing/178563?camid=4v1a)