Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 302 (2014) 5–28

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Reliable Composite Web Services Execution: Towards a Dynamic Recovery Decision

Rafael Angarita [1]

PSL, Université Paris-Dauphine
75775 Paris Cedex 16
France CNRS, LAMSADE UMR 7243

Yudith Cardinale[2]

Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela

Marta Rukoz [3]

PSL, Université Paris-Dauphine
75775 Paris Cedex 16
France CNRS, LAMSADE UMR 7243
Université Paris Ouest Nanterre La Défense
France

Abstract

During the execution of a Composite Web Service (CWS), different faults may occur that cause WSs failures. There exist strategies that can be applied to repair these failures, such as: WS retry, WS substitution, compensation, roll-back, or replication. Each strategy has advantages and disadvantages on different execution scenarios and can produce different impact on the CWS *QoS*. Hence, it is important to define a dynamic fault tolerant strategy which takes into account environment and execution information to accordingly decide the appropriate recovery strategy. We present a preliminary study in order to analyze the impact on the CWS total execution time of different recovery strategies on different scenarios. The experimental results show that under different conditions, recovery strategies behave differently. This analysis represents a first step towards the definition of a model to dynamically decide which recovery strategy is the best choice by taking into account the context-information when the failure occurs.

Keywords: Fault-tolerance, dynamic, execution, context-aware.

[1] Email: rafael.angarita@lamsade.dauphine.fr

[2] Email: yudith@ldc.usb.ve

[3] Email: marta.rukoz@lamsade.dauphine.fr

http://dx.doi.org/10.1016/j.entcs.2014.01.018

1 Introduction

Nowadays, SOA architecture is used as a platform for business applications for accessing data and services in distributed environments. The constantly increasing number of such applications currently deployed over the Internet is enabled by the latest SOA techniques, such as Web Services (WS) and Web 3.0, and is a consequence of the need for business integration and collaboration. With machine intelligence, users can resolve complex problems that require the interaction among different tasks. One of the major goals of the Web 3.0 is to support automatic and transparent WS composition and execution, allowing a complex user request to be satisfied by a Composite Web Service (CWS). Hence, in a CWS, functionalities of individual WSs (possibly from different providers) are combined to resolve the complex query [2].

Most of the generic or domain-tailored solutions for creating, executing, and managing such CWSs have been extensively treated in the literature, exhibiting sophisticated interfaces and a multitude of connectors to subsystems to represent functional properties, and increasing support for non-functional properties [5,3,13,17,11,6,15]. Nevertheless, recovery of failures for reliable execution have received relatively limited attention.

During the execution of a CWS, different faults may occur that cause a WS failure. However, a fault-tolerant CWS is the one that, upon a service failure, ends up the whole composite service (e.g., by retrying, substituting, or replicating the faulty WS) or leaves the execution in a safe state (e.g., by rollbacking or compensating the faulty WS and the related executed WSs). In this sense, fault tolerant CWS becomes a key mechanism to cope with challenges of open-world applications in dynamic changing and untrusted operating environments to ensure that the whole system remains in a consistent state even in the presence of failures [23].

Several techniques have been proposed to implement fault tolerant CWS execution. In some works, transactional properties of component WSs (e.g., retriable, compensable or not) are considered to ensure the classical ACID (all-or-nothing) properties in CWSs [13,11,6,15,9,4]. In this context, failures during the execution of a CWS can be repaired by backward or forward recovery processes. Backward recovery implies to undo the work done until the failure and go back to the initial consistent state (before the execution started), by roll-back or compensation techniques. Forward recovery tries to repair the failure and continue the execution, using retry and substitution, for example. In previous works, we presented our solutions based on backward and forward recovery [8,10].

However, backward recovery means that users do not get the desired answer to their queries, besides roll-back techniques that claim for logs in persistent storage to enable recovery after a re-start, reboot, or crash. The need for synchronous logging slows down the execution speed during normal operation and the reliability of these mechanisms depends on the reliability of the storage. Forward recovery could imply long waiting times, due of the invested time to repair failures until users finally get the response.