Designing Self-Adaptive Service-Oriented Applications
G. Denaro, M. Pezzè, D. Tosi

research report n. 4
september 2006
Table of Contents

Designing Self-Adaptive Service-Oriented Applications ................. 7
Giovanni Denaro, Mauro Pezzè and Davide Tosi

1 Introduction .................................................. 7
2 The Service Mismatch Problem ................................ 9
3 A Self-Adaptive Approach ..................................... 10
4 Design Time Support ......................................... 13
  4.1 Integration Mismatch Taxonomy ......................... 14
5 A Framework for Developing Self-Adaptive Service-Oriented
  Applications .................................................. 18
  5.1 The SHIWS Eclipse plug-in ............................... 18
  5.2 The runtime infrastructure .............................. 20
6 Preliminary Results and Lessons Learned ......................... 21
  6.1 The Methodology ........................................ 21
  6.2 Preliminary Validation Results ......................... 23
7 Related Work ................................................ 27
8 Conclusions .................................................. 28
Designing Self-Adaptive Service-Oriented Applications

Giovanni Denaro\textsuperscript{1}, Mauro Pezzè\textsuperscript{1} and Davide Tosi\textsuperscript{1}

Università degli Studi di Milano - Bicocca
Via Bicocca degli Arcimboldi 8 20126 Milano, Italia

Abstract. The integration of third-party web services helps solve complex business problems and reduce risks, costs and time-to-market. However, the task of the integrators is challenged by services that are maintained by different organizations, and may evolve dynamically and autonomously. The impossibility of statically determining which service implementation will be bound at runtime may lead to unexpected failures.

This paper presents a novel approach for designing self-adaptive service-oriented applications, which autonomously react to changes in the implementation of the services, automatically detect possible integration mismatches, and dynamically execute suitable adaptation strategies. The solution proposed in this paper, is based on a runtime infrastructure that automatically tests remote web services, uses test results to diagnose service mismatches, and executes adaptation strategies to overcome the revealed problems without user intervention.

1 Introduction

Web services and service-oriented architectures are emerging technologies for integrating enterprise applications, leveraging electronic B2B and B2C solutions, and extending the life of legacy software. In a nutshell, web services are remote programs invoked over the Internet, using standard protocols (e.g., HTTP and XML) for exchanging data between requesters and providers. Web services allow enterprises to export functionality outside the enterprise bounds, thus enabling stakeholders of different domains to rapidly and seamlessly integrate third-party expertise into their applications. For example, airlines and hotel chains can export services for online booking; so that travel agencies can combine these and other services to optimize travel plans. Service-oriented architectures help solve complex business problems, decreasing risks, costs and time-to-market.

The integration of third-party web services is challenged by the difficulty of keeping consistency between software systems that are maintained by different organizations and may evolve dynamically and independently, because of both changes in the services and the dynamic discovery of new services. Service providers may change the implementation independently from clients, e.g., to correct faults or meet new requirements. For example, this is the case of old
versions substituted with new ones. Moreover, in many service-oriented architectures, clients can locate services dynamically, using service discovery mechanisms that allow clients to discover and connect web services based on machine-readable descriptions. This means that clients may use different web services in different invocations depending on the choice of a service broker. As illustrated in Figure 1, a broker matches client requests with available services published by providers according to common protocols (e.g., SOAP, WSDL and UDDI), and each time applications reconnect to web services through brokers, they can get different matches and thus use different implementations of the requested services [5].

**Fig. 1.** Main entities and interactions in a service oriented architecture

In this paper, we focus on integration problems that derive from dynamic changes of the invoked services. Different services or service implementations that can be invoked to satisfy a given request must comply with a contract that indicates the characteristics of the required service ¹. In principle, services that comply with the same contract should be equivalent, but in practice contracts tend to specify little more than the service syntax and parameters, leaving many semantic details unspecified and thus implementation-dependent. For example, we have been using web services for obtaining the weather temperatures in US districts on the basis of a contract that required the target location to be indicated with the zip-code and the temperature to be returned as a floating point value, but did not indicate the measurement unit of the return temperature. This contract matched many services that returned temperatures expressed in different measurement units, e.g., Fahrenheit and Celsius, thus leading to client-side failures.

¹ Contracts are usually expressed in standard machine-readable languages, e.g. the Web Service Description Language (WSDL) [3].
We propose a solution that exploits a self-adaptive approach based on a mechanism for revealing possible mismatches between requested and provided services, and for dynamically adapting the client application accordingly. An integration fault taxonomy helps service integrators identify possible integration mismatches, generate test cases for revealing integration problems, and design recovery actions. Integrators can code test cases and adaptation strategies in separate modules, which we refer to as adaptation aspects. We automatically weave adaptation aspects into the client applications, so that the modified client application executes test cases whenever a new service implementation is detected to reveal possible mismatches, and triggers suitable adaptation mechanisms accordingly. We refer to the proposed approach as adaptive integration of web services.

This paper is organized as follows. Section 2 describes mismatch problems that may derive from the integration of web services, referring to a working example that will be used throughout the paper. Section 3 presents an overview of our approach to adaptive integration of web services. Section 4 and Section 5 respectively introduce details of the approach discussing the methodology and the framework for developing self-adaptive service-oriented applications. In Section 6 we summarize the results of our experiments. We discuss related work in Section 7, and finally we conclude in Section 8.