ABSTRACT

Service Oriented Systems (SOSs) based on the SOA paradigm are becoming popular thanks to a widely deployed internetworking infrastructure. They are composed by a possibly large number of heterogeneous third-party subsystems and usually operate in a highly varying execution environment, that makes it challenging to provide applications with Quality of Service (QoS) guarantees. A well-established approach to face the heterogeneous and varying operating environment is to design a SOS as a runtime self-adaptable software system, so that a prospective enterprise willing to realize a SOA application can dynamically choose the component services that best fit its requirements and the environment in which the application operates. In this chapter, the authors first review some representative frameworks that have been proposed for SOSs able to adaptively manage a SOA application with QoS requirements. These frameworks are commonly architected as self-adaptive systems following the MAPE-K (Monitor, Analyze, Plan, Execute, and Knowledge) reference model for autonomic computing. The chapter organizes the review using a specific taxonomy for each MAPE-K phase, with the aim to classify the different strategies and mechanisms that can be applied. Even if a self-adaptive system requires every MAPE-K phase, the authors then focus on the Plan phase, which is the core of each adaptation framework, presenting both optimal and sub-optimal approaches that have been proposed to effectively face the adaptation task at runtime.
INTRODUCTION

As a case study of SOS for the adaptive management of service composition, we present the main features of a prototype that follows the MAPE-K reference model. We analyze through a set of experiments the different degrees of reliability achieved by a SOA application able or not to detect and adapt its behavior with respect to the churn of the services used to compose it. Our experimental results show that the SOA application managed by the SOS achieves a reliability improvement up to 20% with respect to its unmanaged counterpart.

In computer science, Service-Oriented Architecture (SOA) is now a mature reference paradigm for developing network accessible, service-based applications. The main goal of designing applications following the SOA paradigm is to achieve a better degree of interoperability with respect to legacy distributed applications, which are tied up by constraints, such as programming languages and specific protocols and technologies. SOA applications are built up by composing black-box services that can be discovered and invoked using standard protocols, therefore hiding possibly different technologies. The service composition is usually described by a workflow representing the actual business logic of the application, defining both the execution and data flow.

SOA applications have the clear advantage over legacy applications to be easily reused because they can be published as services in a standard registry, where other applications can discover them for further invocation. As a consequence, the focus in developing a SOA application is shifted to activities concerning the identification, selection, and composition of services offered by third parties rather than the classic in-house development.

Systems realized using the SOA paradigm take the name of Service Oriented Systems (SOSs). They benefit from the SOA flexibility as well as from the presence of a widely deployed networking infrastructure. The diffusion of systems deployed using the SOA paradigm is leading to the proliferation of service marketplaces (such as SAP Service Marketplace and Windows Azure Marketplace), where an enterprise can find every component needed to build its SOA applications. With an ever increasing number of service providers on the global market scene, it is becoming easy to find multiple providers implementing the same functionality with different quality levels, e.g., different providers can exhibit different response times or costs for services that present the same logic. Therefore, depending on the needs of the SOA application, it is possible to dynamically select the services that best fit its (possibly changing) requirements.

However, several problems arise when a SOA application, which is offered using third party services, needs to fulfill non-functional requirements, because existing services may disappear or their performance may quickly fluctuate over time, due to the highly varying execution environment. The SOA paradigm easily allows to replace services with equivalent ones, but this task could be very challenging for a human being, especially when several services must be replaced at the same time. Similarly, when the service composition logic needs to be partially or even entirely modified in order to account for changes in the functional requirements, it is hard to manually choose among several alternative workflows, considering also the non-functional requirements.

In addition, the management complexity of SOSs rapidly grows as the number of services involved in the compositions increases. To tackle such complexity, to reduce management costs, and to provide better operativeness, a common and well-established approach is to design SOSs as runtime self-adaptable software systems (Salehie & Tahvildari, 2009), that is, software systems able to detect changes in the environment and to properly reconfigure themselves.

In the field of self-adaptable software systems, the main research branches that have been pursued regard the functional and non-functional requirements of SOA applications. In this chapter, we