Dynamic Service Substitution in Service-Oriented Architectures

Manel Fredj
INRIA-Rocquencourt
Manel.Fredj@inria.fr

Nikolaos Georgantas
INRIA-Rocquencourt
Nikolaos.Georgantas@inria.fr

Valerie Issarny
INRIA-Rocquencourt
Valerie.Issarny@inria.fr

Apostolos Zarras
Department of Computer Science
University of Ioannina - Greece
zarras@cs.uoi.gr

Abstract

The problem we deal with in this paper is the dynamic substitution of stateful services that become unavailable during the execution of service orchestrations. Previous research efforts focusing on the reconfiguration of conventional distributed systems enable the substitution of system entities with other prefabricated passive entities that serve as a backup. Nevertheless, the problem of service substitution is far more complex. In SOA, we can assume the possible existence of several semantically compatible services capable of performing the same or similar tasks. However, each one of them constantly serves requests and cannot be considered as a passive backup for other services. Therefore, we propose the SIROCO middleware platform, enabling the runtime, semantic-based service substitution. The basic concepts of SIROCO are discussed along with an experimental evaluation of our first prototype. Our findings show that SIROCO provides the necessary means for achieving dynamic service substitution with a reasonable expense on the execution of service orchestrations.

1. Introduction

In Service Oriented Architecture (SOA), services evolve independently. A service may be deployed, or un-deployed at anytime. Its implementation along with its interface may change without prior notification. *The particular problem we deal with in this paper is the reconfiguration of a set of executing orchestrations upon the unavailability of a service that is required for the execution of these orchestrations.* The goal of the reconfiguration is the dynamic substitution of the unavailable service with an available one. Dealing with the dynamic substitution of stateless services is more or less straightforward. Thus, we concentrate on the worst case that involves the dynamic substitution of stateful services. According to the standard WS-Resource Framework [7], we assume that service state descriptions may be provided, along with the service interface descriptions.

In the field of dynamic reconfiguration of conventional distributed systems, several approaches tackle the issue of substituting an entity with another prefabricated backup entity [5, 3, 6, 9, 1, 2]. However, the problem of service substitution is far more complex. In SOA, we can assume the possible existence of several semantically compatible services capable of performing the same or similar tasks. However, each one of them constantly serves requests and cannot be considered as a passive backup for other services. Therefore, the reconfiguration process that we are after consists of: *(1) discovering candidate substitute services out of a set of semantically compatible services that can be used in place of a service, which becomes unavailable*, and *(2) trying to identify amongst them the one that can be used as an actual substitute; in the best case the selected substitute service must be such that the current state of its resources can be synchronized with the state of the resources used by the service that is substituted.* Based on the above, *our contribution is SIROCO, a middleware infrastructure that enables the reconfiguration of service orchestrations upon the unavailability of services used in these orchestrations*.

The rest of this paper is structured as follows. Section 2 discusses our approach for dynamic substitution of stateful services in SOA. Section 3 provides our conclusions.

2. Reconfiguration in SOA

In conventional distributed systems, dynamic reconfiguration relies on a *generic reconfiguration cycle*, which provides an abstract view of various reconfiguration approaches that have been proposed in the past (the interested reader may refer to [10] for a more detailed survey). Con-

978-0-7695-3286-8/08 $25.00 © 2008 IEEE
DOI 10.1109/SERVICES-1.2008.52

ceptually, the basic entities involved in the reconfiguration process of proposed approaches are the *Reconfigurable System (RS)*, its *Context* or *Environment (CE)*, and the *Reconfiguration Management System (RM)*.

In the problem that we investigate, the configuration of a SOA-based RS consists of a set of executing orchestrations that combine the functionalities of a set of services. The CE comprises Web services that have been independently developed and deployed in certain sites. Certain of these services may potentially serve as candidate substitutes of services used in the orchestrations of RS.

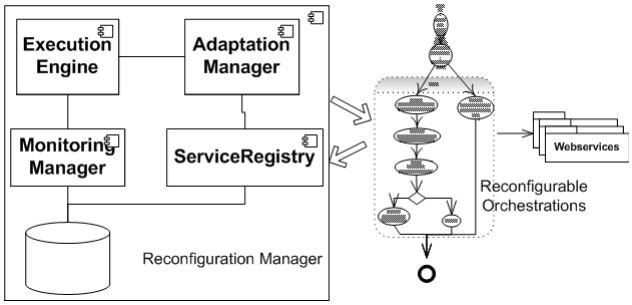

Figure 1. Overview of SIROCO.

SIROCO (Fig. 1) offers a RM which consists of a *service-registry* that manages information concerning Web services that are available in CE, a BPEL *execution-engine* that executes RS orchestrations, a *monitoring-manager* that inspects the execution of these orchestrations and an *adaptation-manager* that dynamically reconfigures the orchestrations when necessary. Without loss of generality, in this paper we assume that RM is in charge of executing all the orchestrations that involve the services that are available in CE. However, the proposed approach can be extended in a quite straightforward way towards a coordinated set of RMs that deal with service substitutions.

2.1. Information managed by RM

The information managed by the SIROCO RM consists of (1) BPEL descriptions of the RS orchestrations, (2) SA-WSDL descriptions of the services used in these orchestrations and (3) SA-WSDL descriptions of services that may be used for the reconfiguration of the RS orchestrations.

With BPEL [4], service orchestrations are specified as workflow-structured sets of activities. BPEL supports the specification of fault handling and compensation activities. Such application-specific activities may serve for handling the unavailability of a service. In general, we see these facilities as complementary to our middleware-layer transparent approach.

Besides BPEL descriptions, RM manages the service SA-WSDL description(s) [8]. In SIROCO, we employ se-

Figure 2. SIROCO OWL annotations.

mantic annotations provided by SA-WSDL to categorize semantically compatible Web services with respect to an OWL ontology managed by the SIROCO service-registry. Moreover, we employ semantic annotations for service operations in order to distinguish between read and write operations, respectively annotated with '*QueryState*' and '*UpdateState*'(Fig. 2). This distinction serves for enriching a BPEL orchestration with activities requesting the SIROCO monitoring-manager to checkpoint (if possible) the state of Web services before the execution of activities that invoke operations which change the Web services' state (i.e., *UpdateState*-annotated operations). Checkpointing is possible if there exist descriptions of the resources used by the services involved. Such descriptions are specified using a WS-ResourceProperties document [7]. Providing WS-ResourceProperties documents along with service descriptions is not mandatory in SIROCO. Nevertheless, SIROCO takes advantage of this information, if available, to find the best substitute for an unavailable service.

2.2. SOA Reconfiguration

RS normal execution. As in conventional distributed systems, during this phase RS executes normally. In our case, this means that, during this phase RS may provide as input to RM, descriptions of orchestrations that should be executed. Given a novel orchestration description along with *abstract* SA-WSDL descriptions (i.e., semantically annotated WSDL descriptions that do not contain any binding information) of the services required for the execution of this orchestration, a number of preparatory steps are performed by RM before executing the orchestration.

First, the service-registry is searched for Web services that can be used for the execution of the orchestration. The service-registry maintains a set of service catalogs. Each catalog corresponds to a different semantic category of services, characterized by an OWL semantic class. The particular ontology that characterizes an RS should be provided during the setup phase of SIROCO. Each service catalog is progressively populated (during the lifetime of RS) with *concrete* SA-WSDL descriptions of services (i.e., semantically annotated WSDL descriptions that contain binding information) that are available in CE.

The discovery of services that can be used for the execution of the novel orchestration is followed by the enrichment

of the orchestration activities with additional checkpoints. Technically, the checkpointing activities send towards the service GetResourceProperties messages with respect to the WS-ResourceProperties document of the service. As a result, reply messages containing the state of the service are returned back to the BPEL execution-engine. Then, the engine forwards the state to the monitoring-manager, which stores it persistently.

The preparation for the execution of the checkpoints-enriched orchestration ends up by parsing the orchestration description towards the construction of (1) an abstract control-flow dependency graph (CDG), and (2) an abstract dataflow dependency graph (DDG) which shall serve for the potential reconfiguration of the orchestration. The nodes in both graphs are the basic BPEL activities of the orchestration. Typically, in the control-flow graph a dependency from an activity a to an activity b denotes that the execution of a precedes the execution of b. In the dataflow graph, a dependency from an activity a to an activity b denotes that the output produced by a as a result of interacting with a service is utilized by b as input for interacting with the same or another service. The CDG and DDG are given as input to the adaptation-manager. Finally, the BPEL execution-engine takes in charge of instantiating the novel orchestration. In general, the BPEL execution-engine may be engaged in the concurrent execution of multiple orchestrations that combine available Web services. The same service may be used in more than one orchestration.

A cause for reconfiguration occurs. This phase takes place upon the unavailability of a service involved in the executing orchestrations, requiring thereby a RS reconfiguration. While the BPEL execution-engine of SIROCO executes RS orchestrations, interaction with the Web services is realized through the use of the standard JAXRPC mechanism. For each activity of an orchestration, the execution-engine checks for standard JAXRPC exceptions (e.g., RemoteException, AXISFault, etc.) that may be thrown while the activity attempts to interact with a Web service. If such an exception is caught, the SIROCO adaptation-manager is notified.

Preparing the reconfiguration. This phase begins when the SIROCO adaptation-manager is notified about the occurrence of an exception in the execution of an orchestration. The adaptation-manager checks the set of executing orchestrations for other *affected orchestrations*. The set of *affected orchestrations* consists of the orchestration that failed to interact with the service, and a subset of other orchestrations whose descriptions comprise activities that interact with the unavailable service. The adaptation-manager blocks the execution of the affected orchestrations to prevent the occurrence of further exceptions.

Planning the reconfiguration actions. With the affected orchestrations blocked, the goal of this phase is to discover candidate substitute services that may replace the unavailable service. To this end, the adaptation-manager requests the service-registry for a list of substitute candidates. Following, the service-registry looks for the catalog corresponding to the OWL semantic class that characterizes the SA-WSDL description of the unavailable service. Within this catalog it searches for SA-WSDL descriptions of services whose WSDL interface matches the interface of the unavailable service. The result set of this task is divided in two categories. The first category contains descriptions of services whose WS-ResourceProperties descriptions exactly match the WS-ResourceProperties description of the unavailable service, while the second category contains all the other services with matching interfaces. If the unavailable service is not accompanied with a WS-ResourceProperties description, the first category of services is empty.

Reconfiguring the system. Given the set of candidate substitute services that resulted from the previous phase, the adaptation-manager tries to select one service out of the set that can actually substitute the unavailable service. First, the adaptation-manager queries the monitoring-manager for the latest state obtained from the unavailable service; the latest state is the result of a checkpointing activity that took place during the failed orchestration, or some other affected orchestration. Following, the adaptation-manager iterates over the 1st category of services; for each service it tries to synchronize the current state of the service with the state of the unavailable service. The semantics of state synchronization is specific to the services involved and must be defined with respect to the WS-ResourceProperties description that characterizes the services. The state synchronization task involves sending a SetResourceProperties message to the candidate substitute service. The result of sending a SetResourceProperties message (i.e., the state synchronization) to the candidate substitute service may be successful or not. In the latter case, the adaptation-manager proceeds with the next service from the 1st category of candidate substitute services. If the state synchronization fails for all candidate services of the 1st category, a service from the 2nd category is randomly selected.

Completing the reconfiguration. The goal of this phase is to put the affected orchestrations back to normal execution. This task highly depends on the outcome of the previous phase. In particular, if the adaptation-manager discovered a service substitute in the 1st category of candidate services, the execution of all the affected conversations is resumed from the points where they were stopped (i.e., from the activities that were blocked or failed).

3. Conclusion

In this paper we presented the SIROCO middleware platform that enables the dynamic substitution of stateful services during the execution of service orchestrations. As opposed to conventional dynamic reconfiguration approaches, the SIROCO reconfiguration process enables semantic-based service substitution of running orchestrations. To assess the basic concepts of SIROCO, an experimental evaluation of our first prototype shows that SIROCO provides the necessary means for achieving dynamic service substitution with a reasonable expense on the execution of service orchestrations.

Nevertheless, the problem of dynamic service substitution involves further challenging issues for future research. Up to now, in SIROCO we select candidate substitute services whose state descriptions exactly match the state descriptions of unavailable services. To further improve our approach we investigate the issue of identifying similarities between service state descriptions towards a systematic mechanism for performing service state transformations. Finally, we work towards a mechanism for the distributed coordination of multiple SIROCO middleware instances.

References

[1] C. Bidan, V. Issarny, T. Saridakis, and A. Zarras. A Dynamic Reconfiguration Service for CORBA. pages 35–42, 1998.

[2] G. S. Blair, L. Blair, V. Issarny, P. Tuma, and A. Zarras. The Role of Software Architecture in Constraining Adaptation in Component-Based Middleware Platforms. pages 164–184, 2000.

[3] K. M. Goudarzi and J. Kramer. Maintaining Node Consistency in the Face of Dynamic Change. In *Proceedings of the 3rd IEEE International Conference on Configurable Distributed Systems*, pages 62–69, 1996.

[4] IBM, Microsoft Corporation and BEA. Business Process Execution Language for Web Service (BPEL4WS) v.1.0. Technical report, IBM, Microsoft Corporation, BEA, 2002. http://www.ibm.com/developerworks/webservices/library/ws-bpel/.

[5] J. Kramer and J. Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, 1990.

[6] N. Minsky, V. Ungureanu, W. Wang, and J. Zhang. Building Reconfiguration Primitives into the Law of a System. pages 62–69, 1996.

[7] OASIS. Web Services Resource Properties (WS-ResourceProperties). Technical report, OASIS, 2004. http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf.

[8] W3C. Semantic Annotations for WSDL and XML Schema. Technical report, W3C, 2007. http://www.w3c.org/TR/sawsdl.

[9] I. Warren and I. Sommerville. A Model for Dynamic Configuration which Preserves Application Integrity. pages 81–88, 1996.

[10] A. Zarras, M. Fredj, N. Georgantas, and V. Issarny. *Rigorous Development of Complex Fault-Tolerant Systems*, volume 4157, chapter Engineering Reconfigurable Distributed Software Systems: Issues Arising for Pervasive Computing, pages 364–386. LNCS, 2006.

Acknowledgments. This work is partly supported by the MobWS GSRT grant for Cooperation in S&T areas with European countries. It was co-funded by the EU in the framework of the project Support of Computer Science Studies in the University of Ioannina of the Operational Program for Education and Initial Vocational Training of the 3rd Community Support Framework of the Hellenic Ministry of Education, funded by national sources and by the (ESF). This research is further partly supported by the European IST PLASTIC project 1(EU-IST-026955).