A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing*

Julia Hielscher[1], Raman Kazhamiakin[2], Andreas Metzger[1], and Marco Pistore[2]

[1] SSE, University of Duisburg-Essen, Schützenbahn 70, 45117 Essen, Germany
{hielscher,metzger}@sse.uni-due.de
[2] FBK-Irst, via Sommarive 18, 38050, Trento, Italy
{raman,pistore}@fbk.eu

Abstract. Service-based applications have to continuously and dynamically self-adapt in order to timely react to changes in their context, as well as to efficiently accommodate for deviations from their expected functionality or quality of service. Currently, self-adaptation is triggered by monitoring events. Yet, monitoring only observes changes or deviations after they have occurred. Therefore, self-adaptation based on monitoring is reactive and thus often comes too late, e.g., when changes or deviations already have led to undesired consequences. In this paper we present the PROSA framework, which aims to enable proactive self-adaptation. To this end, PROSA exploits online testing techniques to detect changes and deviations before they can lead to undesired consequences. This paper introduces and illustrates the key online testing activities needed to trigger proactive adaptation, and it discusses how those activities can be implemented by utilizing and extending existing testing and adaptation techniques.

1 Introduction

Service-based applications operate in highly dynamic and flexible contexts of continuously changing business relationships. The speed of adaptations is a key concern in such a dynamic context and thus there is no time for manual adaptations, which can be tedious and slow. Therefore, service-based applications need to be able to self-adapt in order to timely respond to changes in their context or their constituent services, as well as to compensate for deviations in functionality or quality of service. Such adaptations, for example, include changing the workflow (business process), the service composition or the service bindings.

In current implementations of service-based applications, monitoring events trigger the adaptation of an application. Yet, monitoring only observes changes or deviations *after* they have occurred. Such a reactive adaptation has several important drawbacks. First, executing faulty services or process fragments may have undesirable consequences, such as loss of money and unsatisfied users. Second, the execution of adaptation activities on the running application instances can considerably increase execution time, and therefore reduce the overall performance of the running application. Third,

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

P. Mähönen, K. Pohl, and T. Priol (Eds.): ServiceWave 2008, LNCS 5377, pp. 122–133, 2008.

it might take some time before problems in the service-based application lead to monitoring events that ultimately trigger the required adaptation. Thus, in some cases, the events might arrive so late that an adaptation of the application is not possible anymore, e.g., because the application has already terminated in an inconsistent state.

Proactive adaptation presents a solution to address these drawbacks, because – ideally – the system will detect the need for adaptation and will self-adapt before a deviation will occur during the actual operation of the service-based application and before such a deviation can lead to the above problems.

In this paper we introduce the *PROSA* framework for *PRO*-active *S*elf-*A*daptation. PROSA's novel contribution is to exploit online testing solutions to proactively trigger adaptations. Online testing means that testing activities are performed during the operation phase of service-based applications (in contrast to offline testing which is done during the design phase). Obviously, an online test can fail; e.g., because a faulty service instance has been invoked during the test. This points to a potential problem that the service-based application might face in the future of its operation; e.g., when the application invokes the faulty service instance. In such a case, PROSA will proactively trigger an adaptation to prevent undesired consequences.

The remainder of the paper is structured as follows: In Section 2 we give an overview of current research results on using monitoring to enable (reactive) adaptation and of the state-of-the-art in online and regression testing. In Section 3 we present the PROSA framework. While describing the key elements of the framework, we discuss how those could be implemented by utilizing or extending existing testing and adaptation techniques. Section 4 introduces several application scenarios to illustrate how PROSA addresses different kinds of deviations and changes. Finally, Section 5 critically reviews the framework and highlights future research issues.

2 State-of-the-Art

2.1 Monitoring for Adaptation

Existing approaches for adaptation of service-based applications rely on the possibility to identify and realize – at run-time – the necessity to change certain characteristics of an application. In order to achieve this, adaptation requests are explicitly associated to the relevant events and situations. *Adaptation requests* (also known as adaptation requirements or specifications) specify how the underlying application should be modified upon the occurrence of the associated event or situation. These events and situations may correspond to various kinds of failures (like application-level exceptions and infrastructure-level failures), changes in contextual settings (like execution environment and usage context), changes among available services and their characteristics, as well as variations of business-level properties (such as key performance indicators).

In order to detect these events and situations, the majority of adaptation approaches resorts to exploiting *monitoring* techniques and facilities, as they provide a way to collect and report relevant information about the execution and evolution of the application. Depending on the goal of a particular adaptation approach, different kinds of events are monitored and different techniques are used for this purpose.

In many approaches (e.g., [1,2,3,4]) the events that trigger the adaptation are failures. These failures include typical problems such as application exceptions, network problems and service unavailability [1,4], as well as the violation of expected properties and requirements. In the former case fault monitoring is provided by the underlying platform, while in the latter case specific facilities and tools are necessary. In [2] Baresi et al. define the expected properties in the form of WS-CoL assertions (pre-, post-conditions, invariants), which define constraints on the functional and quality of service (QoS) parameters of the composed process and its context. In [5] Spanoudakis et al. use properties in the form of complex behavioral requirements expressed in event calculus. In [3] Erradi at al. express expected properties as policies on the QoS parameters in the form of event-condition-action (ECA) rules. When a deviation from the expected QoS parameters is detected, the adaptation is initiated and the application is modified. In such a case, adaptation actions may include re-execution of a particular activity or a fragment of a composition, binding/replacement of a service, applying an alternative process, as well as re-discovering and re-composing services. In [6] Siljee et al. use monitoring to track and collect the information regarding a set of predefined QoS parameters (response time, failure rates, availability) infrastructure characteristics (load, bandwidth) and even context. The collected information is checked against expected values defined as functions of the above parameters, and in case of a deviation, the reconfiguration of the application is triggered.

Summarizing, all these works follow the reactive approach to adaptation, i.e., the modification of the application takes place *after* the critical event happened or a problem occurred.

The situation with reactive adaptation is even more critical for approaches that rely on post-mortem analysis of the application execution. A typical monitoring tool used in such approaches is the analysis of process logs [7,8,9]. Using the information about histories of application executions, it is possible to identify problems and non-optimalities of the current business process model and to find ways for improvement by adapting the service-based application. However, once this adaptation happens, many process instances might have already been executed in a "wrong" mode.

2.2 Online Testing and Regression Testing

The goal of testing is to systematically execute service instances or service-based applications (service compositions) in order to uncover failures, i.e., deviations of the actual functionality or quality of service from the expected one.

Existing approaches for testing service-based applications mostly focus on testing during design time, which is similar to testing of traditional software systems. There are a few approaches that point to the importance of online testing of service-based applications. In [10] Wang et al. stress the importance of online testing of web-based applications. The authors, furthermore, see monitoring information as a basis for online testing. Deussen et al. propose an online validation platform with an online testing component [11]. In [12] metamorphic online testing is proposed by Chan et al., which uses oracles created during offline testing for online testing. Bai et al. propose adaptive testing in [13,14], where tests are executed during the operation of the service-based application and can be adapted to changes of the application's environment or of

the application itself. Finally, the role of monitoring and testing for validating service-based applications is examined in [15], where the authors propose to use both strategies in combination. However, all these approaches do not exploit testing results for (self-)adaptation.

An approach related to online testing is regression testing. Regression testing aims at checking whether changes of (parts of) a system negatively affect the existing functionality of that system. The typical process is to re-run previously executed test cases. Ruth et al. [16,17] as well as Di Penta et al. [18] propose regression test techniques for Web services. However, none of the techniques addresses how to use test results for the adaptation of service-based applications.

Summarizing, in spite of a number of approaches for online testing and regression testing, none of these approaches targets the problem of proactive adaptation. Still, several of the presented approaches provide baseline solutions that can be utilized and extended to realize online testing for proactive adaptation. This will be discussed in the following section.

3 PROSA: Online Testing for Proactive Self-adaptation

As introduced in Section 1, the novel contribution of the PROSA framework is to exploit online testing for proactive adaptation. Therefore, the PROSA framework prescribes the required online testing activities and how they lead to adaptation requests. Figure 1 provides an overview of the PROSA framework and how the proactive adaptation enabled by PROSA relates to "traditional" reactive adaptation which is enabled by monitoring.

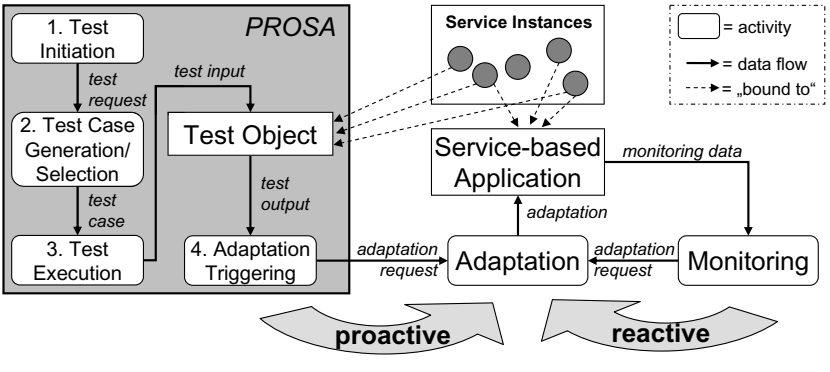

Fig. 1. The PROSA Framework

The PROSA framework prescribes the following four major activities:

1. *Test initiation*: The first activity in PROSA is to determine the need to initiate online tests during the operation of the service-based application. The decision on when to initiate the online tests depends on what kind of change or deviation should be uncovered (see Section 3.1).

2. *Test case generation/selection*: Once online testing has been initiated by activity 1, this second activity determines the test cases to be executed during online testing. This can require creating new test cases or selecting from already existing ones (see Section 3.2).
3. *Test execution*: The test cases from activity 2 are executed (see Section 3.3).
4. *Adaptation triggering*: Finally, an analysis of the test results provides information on whether to adapt the service-based application and thus to create adaptation requests (see Section 3.4).

It should be noted that – as depicted in Figure 1 – online testing does not interfere with the execution of the actual application in operation, i.e. with those instances of the application which are currently used by actual users. Rather, online testing performs tests of the constituent parts of the service-based application (e.g., individual services or service compositions) independent from and in parallel to the operating applications.

Details about the above activities and how those can be implemented with existing techniques are discussed in the remainder of this section.

3.1 Test Initiation

In order to initiate the actual online testing activities (PROSA's activities 2 and 3), two questions need to be answered: "When to test?" and "What to test?". The answer of these questions depends on the kinds of changes or deviations that should be *proactively* addressed in addition to reactive techniques like monitoring. Those possible kinds of changes are listed in Table 1.

To give an answer to the question "When to test?", Table 1 provides an explanation when to initiate online testing depending on the kind of change or deviation. Those kinds of changes and deviations are illustrated in more detail in Section 4, where different application scenarios for PROSA are introduced.

Table 1. Different cases for initiating online testing

Case	*Why to initiate online testing?*	*When to initiate online testing?*	*What to test?*
1	Uncovering failures introduced due to the adaptation of the service-based application.	Once the respective adaptation (e.g. binding of a new service) has been performed.	service or composition
2	Detecting changes in the service-based application or its context that could lead to failures in the "future".	Once monitoring has detected a change that does not reactively trigger an adaptation.	service or composition
3	Identifying failures of an application execution.	Periodically (e.g. randomly or by testing future service invocations along the execution path of the application).	composition
4	Uncovering failures (i.e., deviations from expected functionality or quality) or unavailability of constituent services.	Periodically (e.g., randomly or by predicting future service invocations along the execution path of the application).	service

To provide an answer to the question "What to test?" (i.e., to determine the test object), we have considered the following two major strategies that can be performed in order to uncover the different kinds of changes or deviations (Table 1 shows what strategy could be followed depending on the kind of change or deviation):

- *Testing constituent service instances:* Similar to unit or module testing, the individual, constituent service instances of a service-based application can be tested (i.e., the service instances that are or will be bound to the service-based application).
- *Testing service compositions:* Similar to system and integration testing, the complete service composition of a service-based application or parts thereof can be tested.

To implement activity 1 of PROSA, one can rely on information provided by existing monitoring techniques for case 2 (see Table 1) or adaptation techniques for case 1. The other cases require new and specific techniques, which can be very simple (like randomly triggering the tests) or more challenging (like predicting future service invocations along the execution path of the application).

3.2 Test Case Generation/Selection

In Section 3.1 two strategies for online testing were introduced. In order to implement these two different strategies and thus to realize activity 2 of the PROSA framework, different kinds of techniques for determining test cases have to be employed:

- *Testing constituent service instances:* For testing constituent service instances, existing techniques for test case generation from service descriptions, like WSDL, can be exploited (e.g., [19,20,21]). Additionally, test cases from the design phase can be re-used if such test cases exist. However, usually the test cases from the design phase will not suffice, because typically at that time not all services are known due to the adaptation of a service-based application that can happen during run-time.
- *Testing service compositions:* For testing service compositions, test cases can be generated from composition specifications, like BPEL (e.g., [22,23]). If a set of test cases for testing service compositions already exists, online testing has to determine which of those test cases to execute again (i.e., test cases have to be selected). This is similar to regression testing, which has been discussed in Section 2.2. Consequently, existing techniques for regression testing of services (like [16,17,18]) can be utilized.

A more detailed survey on existing test case generation and selection techniques for service-based applications can be found in [24].

3.3 Test Execution

The responsibility of activity 3 in the PROSA framework is to execute the test cases that have been determined by activity 2. This means that the test object (which is either a service instance or a service composition) is fed with concrete inputs (as defined in the test cases) and the produced outputs are observed.

The test execution can be implemented by resorting to existing test execution environments, e.g., the ones presented in [19,18]. It is important to note that invoking services can lead to certain "side effects" which should not occur when invoking the service for testing purposes only (this problem is also discussed in [22]). As an example, when invoking the service of an online book seller for testing purposes, one would not like to have the "ordered" books actually delivered. Thus, it is necessary to provide certain services with a dedicated test mode. As an example, one could follow the approaches suggested for testing software components, where components are provided with interfaces that allow the execution of the component in "normal mode" or in "test mode" (see [25]).

3.4 Adaptation Triggering

The final activity 4 of PROSA determines whether to issue an adaptation request, which ultimately leads to the modification of the service-based application. Such an adaptation request should be issued when the observed output of a test deviates from the expected output, i.e., whenever a test case fails. This includes deviations from the expected functionality as well as from the expected quality of service.

As has been discussed above, existing adaptation solutions rely on monitoring to issue adaptation requests whenever a deviation is observed (see reactive loop in Figure 1). In order to exploit those existing solutions (see Section 2.1), triggering of adaptations based on online testing should conform to the requests from the monitoring component. Thereby, activity 4 could be implemented within a unified adaptation framework.

To achieve such a unification, the following two issues need to be resolved: First, specific adaptation requests should be explicitly assigned to individual test cases. In reactive approaches such adaptation requests are assigned to certain monitoring events. The events may represent application or network failures (e.g., service is unavailable), violation of assertions (e.g., post-condition on data returned by service call) or even of complex behavioral properties (e.g., if flight is found but there are no rooms available, the trip plan can not be created). In a similar way, test cases represent dedicated execution scenarios, where specific deviations or changes can be checked (this has been highlighted in Table 1). If the test fails, this is similar to the occurrence of a monitoring event, and thus the adaptation assigned to the test case is triggered.

Second, it may be necessary to modify the adaptation requests from monitoring in order to take into account the specifics of proactive adaptation. Indeed, some adaptation requests from monitoring might specify instructions that are not applicable in proactive adaptation (e.g., "retry" operation, or "rollback to safe point"). Therefore, the specification should be changed such that these instructions do not appear when used for proactive adaptation. An interesting line of future work in these regards could be to devise means to automatically derive adaptation requests for proactive adaptation from the adaptation requests already available for monitoring.

4 Application Scenarios

In this section we illustrate how PROSA enables the proactive adaptation of a service-based application. For this purpose we introduce an example application based on which

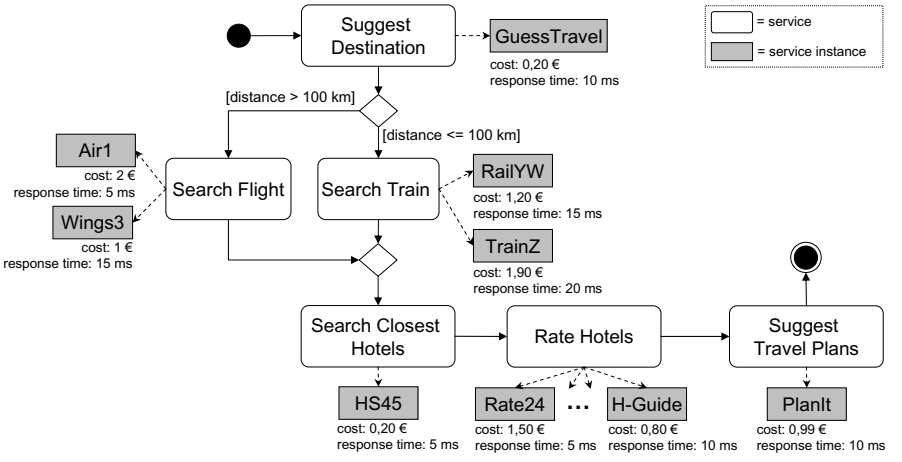

Fig. 2. Example Application: "Travel Planning"

we describe scenarios that demonstrate how PROSA can be applied to the different cases for online testing introduced in Table 1. The service composition of the example and possible constituent service instances are depicted in Figure 2.

Our example application provides a travel planning service, which includes a combined search for transportation and hotel accommodation. The constituent services of this application are invoked in the following order:

1. *Suggest destination:* First, the user of the application is provided with a suggestion of different travel destinations based on her/his preferences.
2. *Search flight/train:* Once the user has chosen a destination, the application will determine the best way to reach that destination. Depending on the distance to the suggested destination, either an appropriate flight or a train connection is searched.
3. *Search closest hotels:* After a suitable means of transportation has been found, hotels in the vicinity of the airport or the railway station of the destination are located.
4. *Rate hotels:* Using one of the many hotel rating services available, each hotel from the list is checked for its rating and the hotel list, sorted according to the rating, is returned.
5. *Suggest travel plans:* Finally, the first hotel from the sorted list (i.e., the one with the best overall rating) is chosen and the travel information (itineraries, information about the hotel, etc.) is compiled to produce a comprehensive travel plan.

In Figure 2, gray boxes denote concrete service instances that can be bound to the application in order to compute the travel plan. Some of those concrete service instances can already be known at design time, while others are dynamically discovered or added due to adaptations during the operation of the service-based application. The annotated information about cost and response time denotes the negotiated quality for each of the service instances (e.g., by means of service level agreements).

4.1 Case 1: Failure Introduced due to Adaptation

Let us assume that the service instance "H-Guide" was bound to our service-based application at operation time, because the service instance "Rate24" has turned out to be too expensive. The binding of that new service instance is reported by the adaptation component to the PROSA framework. Consequently, PROSA's activity 1 triggers the online testing activities, which react to this adaptation by determining test cases to check whether the newly bound service instance behaves as expected (see Table 1, case 1). Let us say that the expected output of one of those test cases is "Palermo Premium Class Hotel", which clearly is the hotel with the best ratings for the chosen location. Unfortunately, the observed output of "H-Guide" is "Casa Palermo", which is the hotel with one of the lowest ratings (the reason for this presumed failure is that – other than expected – "H-Guide" returns the list of hotels in ascending order, starting with the lowest ratings). Online testing reports this failure to the adaptation component, which can – for example – switch back to the initial service instance "Rate24", which has already been used successfully.

4.2 Case 2: Change That Could Lead to Failures in the Future

Let us assume that a new regulation concerning the pricing of flights enters into force during the operation of the service-based application. The regulation requires that the overall cost of a flight (including taxes) has to be stated and that it may not anymore be stated as the price for the flight with the note "plus taxes". This legal change thus represents a change in the context of the application (see Table 1, case 2). As a result, PROSA will initiate online testing activities – when this new regulation enters into force – in order to determine whether the constituent service instances of the service-based application conform to this new regulation. This means that online tests will be triggered in order to check whether the service instances for flight booking ("Air1" and "Wings3") conform to the new regulation. If one of those service instances does not implement the new regulation, the service-based application will be adapted accordingly before that service instance is invoked during the actual operation of the application.

4.3 Case 3: Failure of an Application Execution

The output of "search train" (resp."search flight") contains the name of the city close to the airport or the railway station. This city name is passed on to "search closest hotels" in order to determine the list of hotels in the vicinity of the destination. Let us assume that the service instance "RailYW" always provides the name of the destination in "short" form, meaning that even if there is more than one city with this name, like "Frankfurt am Main" and "Frankfurt an der Oder", this service instance will always return "Frankfurt". When the hotel searching service "HS24" receives such an ambiguous input, it will terminate with an error message. By running test cases to check deviations in the service composition (see Table 1, case 3), PROSA can uncover such a failure and – as a proactive corrective action – can request that a different service instance is bound to the application (e.g., "TrainZ").

4.4 Cases 4: Failure of a Constituent Service

For the booking of an appropriate flight, two service instances are available: “Air1” and “Wings3”. “Air1” is used for premium clients, which are willing to pay more for a shorter response time. “Wings3” is the preferred choice of clients who want to save money. At operation time the online testing component runs several test cases per hour (periodically testing, see Table 1, case 4). Let us assume that one of those tests uncovers that “Wings3” does not respond. PROSA then provides the adaptation component with this information, such that the alternative service instance “Air1” (which is working as expected) is used for all queries.

5 Discussion and Perspectives

This paper has introduced the PROSA framework, which defines key activities for enabling the proactive self-adaptation of service-based applications. The novel contribution of PROSA is to exploit online testing techniques in order to anticipate future deviations or changes of a service-based application and thereby to trigger adaptation requests. In addition to the definition of those key activities, the paper has discussed how those activities can be implemented by building on or extending existing testing and adaptation techniques.

In contrast to the “traditional” form of reactive adaptation (e.g., based on monitoring), PROSA provides the following important benefits: First, changes or deviations from expected functionality or quality of service can be uncovered and addressed before they lead to undesirable consequences. Second, the execution of adaptation activities – if done proactively – does not interfere with the execution of the actual application instances, i.e., the users of the application won’t be affected by the adaptation. Third, proactive adaptation can provide adaptation requests early enough such that an adaptation of the service-based application still is possible (in contrast to reactive adaptation, where the application can have already terminated in an inconsistent state, for instance). Due to these benefits, we are confident that the PROSA framework will enable novel service-based applications that are able to proactively adapt and thus to better meet their expectations.

In addition to uncovering failures, monitoring is also often used to improve (or optimize) a service-based application. Accordingly, online testing could be used in this respect, for instance by determining the best possible alternative for an adaptation decision before the adaptation is executed. This means whenever an adaptation decision is imminent and different alternatives exist, those alternatives could be “pre-tested” and the best one chosen. For example, consider an adaptation specification, where on failure of a service instance three strategies are defined: retry invoking the service instance three times, replace the service instance with another service instance, change the service composition to use different services. Testing can now “simulate” all those three strategies and maybe detect that “change composition” is the only way to successfully drive the adaptation.

Although exploiting only testing for proactive adaptation provides many benefits, we acknowledge at this stage that further work is required in order to demonstrate the applicability of the PROSA idea in practice. One aspect that, for example, has to be

investigated, is the possible impact of the execution of test cases on the performance of the application. Thus, key issues that we will target in our future work are to create a proof-of-concept prototypes based on existing techniques and tools (as discussed in the paper) and to apply these prototypes to realistic cases.

As we have briefly pointed out in the paper, proactive and reactive adaptation may work together in an integrated dynamic adaptation framework. In such a framework, online testing and monitoring could mutually benefit from each other, thereby improving the overall quality and efficiency of adaptation. In further work, we thus plan to investigate on how to best exploit the synergies between monitoring and testing. As an example, the results of monitoring may be used to identify "better" test cases for online testing. When complex behavioral properties are monitored (e.g., see [5,26]), the violations or successful executions are represented as traces containing information of the composition activities. A set of such traces from previous executions may be used to derive new test cases for online testing. Furthermore, monitoring may be used to parametrize the test cases. As the configuration of tests may depend on the operational context of the application, such context information can be provided by monitoring.

References

1. Baresi, L., Ghezzi, C., Guinea, S.: Towards Self-healing Service Compositions. In: First Conference on the PRInciples of Software Engineering (PRISE 2004), pp. 11–20 (2004)
2. Baresi, L., Guinea, S., Pasquale, L.: Self-healing BPEL processes with Dynamo and the JBoss rule engine. In: ESSPE 2007: International workshop on Engineering of software services for pervasive environments, pp. 11–20 (2007)
3. Erradi, A., Maheshwari, P., Tosic, V.: Policy-Driven Middleware for Self-adaptation of Web Services Compositions. In: ACM/IFIP/USENIX 7th International Middleware Conference, pp. 62–80 (2006)
4. Modafferi, S., Mussi, E., Pernici, B.: SH-BPEL: a self-healing plug-in for Ws-BPEL engines. In: 1st workshop on Middleware for Service Oriented Computing, pp. 48–53 (2006)
5. Spanoudakis, G., Zisman, A., Kozlenkov, A.: A Service Discovery Framework for Service Centric Systems. In: SCC 2005: Proceedings of the 2005 IEEE International Conference on Services Computing, pp. 251–259 (2005)
6. Siljee, J., Bosloper, I., Nijhuis, J., Hammer, D.: DySOA: Making Service Systems Self-adaptive. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 255–268. Springer, Heidelberg (2005)
7. van der Aalst, W.M.P., Pesic, M.: Specifying and Monitoring Service Flows: Making Web Services Process-Aware. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 11–55. Springer, Heidelberg (2007)
8. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In: Business Process Management, 5th International Conference, BPM, pp. 328–343 (2007)
9. Nezhad, H.R.M., Saint-Paul, R., Benatallah, B., Casati, F.: Deriving Protocol Models from Imperfect Service Conversation Logs. IEEE Transactions on Knowledge and Data Engineering (TKDE) (to appear, 2008)
10. Wang, Q., Quan, L., Ying, F.: Online testing of Web-based applications. In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC), pp. 166–169 (2004)

11. Deussen, P., Din, G., Schieferdecker, I.: A TTCN-3 based online test and validation platform for Internet services. In: Proceedings of the 6th International Symposium on Autonomous Decentralized Systems (ISADS), pp. 177–184 (2003)
12. Chan, W., Cheung, S., Leung, K.: A metamorphic testing approach for online testing of service-oriented software applications. International Journal of Web Services Research 4, 61–81 (2007)
13. Bai, X., Chen, Y., Shao, Z.: Adaptive web services testing. In: 31st Annual International Computer Software and Applications Conference (COMPSAC), pp. 233–236 (2007)
14. Bai, X., Xu, D., Dai, G., Tsai, W., Chen, Y.: Dynamic reconfigurable testing of service-oriented architecture. In: Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC), pp. 368–375 (2007)
15. Canfora, G., di Penta, M.: SOA: Testing and Self-checking. In: Proceedings of International Workshop on Web Services - Modeling and Testing - WS-MaTE, pp. 3–12 (2006)
16. Ruth, M., Oh, S., Loup, A., Horton, B., Gallet, O., Mata, M., Tu, S.: Towards automatic regression test selection for web services. In: Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC), pp. 729–734 (2007)
17. Ruth, M., Tu, S.: A safe regression test selection technique for Web services. In: Second International Conference on Internet and Web Applications and Services (ICIW) (2007)
18. Di Penta, M., Bruno, M., Esposito, G., et al.: Web Services Regression Testing. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 205–234. Springer, Heidelberg (2007)
19. Martin, E., Basu, S., Xie, T.: Automated Testing and Response Analysis of Web Services. In: IEEE International Conference on Web Services (ICWS), pp. 647–654 (2007)
20. Bai, X., Dong, W., Tsai, W.T., Chen, Y.: WSDL-Based Automatic Test Case Generation for Web Services Testing. In: Proceedings of the IEEE International Workshop on Service-Oriented System Engineering (SOSE), pp. 215–220. IEEE Computer Society, Los Alamitos (2005)
21. Tarhini, A., Fouchal, H., Mansour, N.: A simple approach for testing Web service based applications. In: Bui, A., Bui, M., Böhme, T., Unger, H. (eds.) IICS 2005. LNCS, vol. 3908, pp. 134–146. Springer, Heidelberg (2006)
22. Lübke, D.: Unit Testing BPEL Compositions. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 149–171. Springer, Heidelberg (2007)
23. Dong, W.L., Yu, H., Zhang, Y.B.: Testing BPEL-based Web Service Composition Using High-level Petri Nets. In: EDOC 2006: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference, pp. 441–444. IEEE Computer Society, Los Alamitos (2006)
24. Pernici, B., Metzger, A. (eds.): Survey of quality related aspects relevant for SBAs. S-Cube project deliverable: PO-JRA-1.3.1 (2008), `http://www.s-cube-network.eu/achievements-results/s-cube-deliverables`
25. Suliman, D., Paech, B., Borner, L., Atkinson, C., Brenner, D., Merdes, M., Malaka, R.: The MORABIT approach to runtime component testing. In: Proceedings of the 30th Annual Int'l. Computer Software and Applications Conference (COMPSAC), pp. 171–176 (2006)
26. Barbon, F., Traverso, P., Pistore, M., Trainotti, M.: Run-Time Monitoring of Instances and Classes of Web Service Compositions. In: IEEE International Conference on Web Services (ICWS 2006), pp. 63–71 (2006)