Towards Correctness Assurance in Adaptive Service-Based Applications*

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

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

Abstract. Service-based applications (SBAs) increasingly have to become adaptive in order to operate and evolve in highly dynamic environments. Research on SBAs thus has already produced a range of adaptation techniques and strategies. However, adaptive SBAs are prone to specific failures that would not occur in "static" applications. Examples are faulty adaptation behaviours due to changes not anticipated during design-time, or conflicting adaptations due to concurrently occurring events. For adaptive SBAs to become reliable and thus applicable in practice, novel techniques that ensure the correctness of adaptations are needed. To pave the way towards those novel techniques, this paper identifies different kinds of adaptation-specific failures. Based on a classification of existing adaptation approaches and generic correctness assurance techniques, we discuss how adaptation-specific failures can be addressed and where new advanced techniques for correctness assurance of adaptations are required.

1 Introduction

A wide range of research approaches addresses the problem of adaptation in Service-Based Applications (SBAs). Those include solutions for the definition and realization of adaptation requirements as well as strategies for adapting the applications to new situations and to react to various events and failures. As SBAs increasingly have to operate and evolve in highly dynamic environments and must dynamically and autonomously accommodate for various changes and events, dynamic adaptation is particularly important since it allows for automated and timely modification of the underlying application.

In general, the approach to dynamic SBA adaptation may be described as follows: At design time, the service integrator prescribes the *adaptation specification*, by (i) identifying the dynamic part of the application (i.e., what can change or can happen in the application or its environment that requires the application to adapt), (ii) defining the adaptation requirements that the underlying application should respect, and (iii) selecting the strategies to realize these requirements. During run-time, the application detects the changes and executes the selected adaptation strategies according to the adaptation specification.

* 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. 25–37, 2008.

While the functional behavior of an SBA typically is controlled by user input, dynamic adaptation is triggered by additional information about the application and its context (e.g., failures of constituent services or different network connectivity). As a consequence, dynamic adaptation may lead to new kinds of failures. Those adaptation-specific failures include:

- the execution of the adaptation specification may fail, when the application encounters a situation that is not covered by the adaptation specification, i.e. which has not been taken into account during the design phase;
- the adaptation actions are concurrent with the other events and activities of the application leading to unpredictable results;
- inadequate adaptation strategies are chosen over and over again (i.e., the adaptation enters a life-lock), wasting resources without achieving expected results.

Those kinds of adaptation-specific failures are not explicitly addressed by traditional correctness assurance techniques. Thus, novel means for the correctness assurance of adaptations are needed. In order to provide such correctness assurance techniques for adaptive SBAs, the specific aspects of one or another adaptation approach as well as the specific characteristics of the adaptation-specific failures need to be taken into account.

To pave the way towards those techniques, this paper thus sets out to

- provide a classification of existing adaptation approaches (Section 2);
- identify and classify potential adaptation-specific failures (Section 3);
- provide our vision on how to address these adaptation-specific failures (Section 4).

2 Adaptation in SBA

Depending on the type of adaptation, different adaptation-specific failures may occur and may have a different impact on the functionality of the application. In this section we will thus provide an overview of the different types of adaptations for SBAs. We remark that in this work we focus on the problems related to the adaptation performed automatically at run-time, since the adaptation performed at design-time may be analyzed by conventional quality assurance means, such as verification or testing.

We distinguish between configuration (or parameter) adaptation and composition adaptation. *Configuration adaptation* includes modification of certain application parameters and properties (e.g., non-functional properties) or modification of services involved in the composition (e.g., replacement of one service with another). *Composition adaptation* deals with changes in the application structure, i.e., the flow of the activities is modified. As examples, additional activities are invoked, previous operations are undone, or completely different composition fragments are executed.

Another important factor – orthogonal to the above one – concerns the way the adaptation strategy is determined and realized. In some approaches the strategy is *predefined* at design time, in other approaches the strategy is defined *dynamically* depending on the concrete situation. While the former approaches are easier to realize, they are less flexible at run-time. On the contrary, the latter approaches are more appropriate at run-time, but require complex algorithms and may be time consuming.

2.1 Configuration Adaptation

There exists a wide range of approaches where the changes do not affect the underlying composition model, but only certain parameters or properties of the application configuration. A typical scenario for such kind of approaches concerns the situation, where there is a change of the constituent services, i.e. the services that are bound to the application. The change may happen due to the fact that constituent services become unavailable or their quality-of-service characteristics degrade. In this case adaptation is performed by replacing "bad" services with other ones according to certain criteria. Another example of such type of changes is the modification of service-level agreements taking place as a result of re-negotiation of certain non-functional parameters.

In some of these approaches the expected configuration values or value bounds are predefined. Typically, these values are specified in service-level agreements; they define appropriate levels of quality-of-service (QoS) parameters [1,2]. The approaches aim to discover and bind the services that satisfy these constraints.

Other approaches do not specify any constraints on the parameters but try to dynamically achieve those values that provide the best possible configuration; e.g. in [3] the optimal value of several QoS metrics is targeted and in [4] the services with the best reputation are chosen.

2.2 Composition Adaptation

In many approaches, adaptation modifies the way, in which an application is executed, i,e., the process / composition model of the SBA is changed. These changes may be entailed by various factors, such as the necessity to customize SBA in order to deal with particular user types, or the necessity to operate in a new environment, where policies and constraints for the given SBA are different. In this case, the application should change the executed flow of activities in accordance to the adaptation specification. This may include, for example, to skip certain activities or to perform additional ones, to execute completely different process fragments, or to re-execute or roll back already performed activities.

A typical scenario for composition adaptation using predefined strategies is process migration: a new process model is defined and all the running instances should be changed in order to correspond to this model [5,6]. Adaptation in this case changes those parts of the running application instance that have not been executed yet. Other approaches define potential variants in the application model and conditions, under which the variant applies [7]. In some approaches the adaptation specification has a form of meta-level instructions to be applied to the process model, such as "redo a part of the process", "roll back to safe point", "skip to a particular execution point", "substitute a part of the process", etc [8].

There exist approaches where the composition is defined dynamically. Using certain application requirements, the composition is created automatically and may be even recomposed in case of certain changes or problems. In [9] the approach relies on a set of functional and non-functional requirements and constraints, based on which the services are composed and mediated dynamically. In [10] a domain process model with the different deviations is used as a basis for the composition.

3 Adaptation-Specific Failures

Adaptation-specific failures may occur – in general – for all types of adaptations, or they might be specific to particular approaches or even application domains. Moreover, they may have different character depending on the way the adaptation problem and realization are defined. This section therefore presents three major classes of adaptation-specific failures, discusses their causes, and relates them to the types of adaptation identified above.

3.1 Failures due to Incomplete Knowledge

When SBAs operate in a very open and dynamic environment, the list of potential events, configurations, and contexts is very broad. This makes problematic the applicability of adaptation approaches, in which the adaptation strategy is prescribed at design-time. Indeed, in this case the designer can consider only a relatively small set of situations, assuming that they are representative for all the possible variants of the execution. If this optimistic assumption is violated at run-time, the execution of the predefined adaptation specification may lead to unexpected and dangerous results. For instance, the inability to complete the adaptation (deadlocks), since the applicability of the actions is violated in a concrete context; or the possibility to end up in a "wrong" state not considered at design time.

We remark that this type of problem is specific to the approaches, where the adaptation actions or parameters are predefined.

Configuration adaptation. When adaptation of application configuration is considered, design-time assumptions refer to the possibility to change the application in such a way, that predefined values for the parameters are satisfied. For example, there exists a possibility to satisfy certain level of QoS properties, to discover a service with a given characteristics, etc. At run-time, however, this assumption may be violated since the values are not realistic in a given context, and adaptation may fail.

Composition adaptation. In case of composition adaptation, the situation is even more complex. For the process migration problem, running application instances may be in a state, where adaptation is not possible, since the resulting execution will neither correspond to the initial model nor to the final one. In case of rule-based or meta-level specifications, an implicit assumption is that the adaptation specification is correct and will successfully complete. In practice, however, the execution of adaptation activities may fail, trigger another adaptation activity, etc. Another critical situation may occur if adaptation activities are "semantically" incorrect in a given situation, i.e., they lead to a situation that is undesirable from the application logic point of view.

The situation is made more complex by the fact that adaptation specification is defined independently from the application model using specific languages and notations. This restricts applicability of traditional techniques, such as verification or testing.

3.2 Failures due to Concurrent Changes during Adaptation

In certain application domains possible changes or events in the application environment may be as fast as the execution of adaptation activities. In such cases, events that

occur concurrently with the execution of adaptation activities may trigger another set of adaptation activities, potentially even contradictory with the initial ones.

Such interleaving of adaptation activities and contextual events executed concurrently may lead to variety of problems and inconsistencies in the application: one adaptation activity may completely "negate" the results of another one; the system may end up in an incorrect state or even deadlock; the new events continuously trigger new adaptations leading to an "adaptation stack overflow".

Configuration adaptation. In case of configuration adaptation, the problem refers to the changes in the corresponding configuration parameters of SBA. If the changes are so fast that they become comparable to the re-configuration time, the new "version" of the SBA may become incorrect or non-optimal. For instance, if services (or their QoS metrics) appear / disappear (or change) within minutes or seconds, newly discovered and bound service may become unavailable when invoked.

Composition adaptation. In case of composition adaptation, the execution flow of the application is modified and activities and tasks different from those in the original model are performed. If during the execution of these task new adaptation triggering events occur, the execution flow is modified again. Depending on how the adaptation is defined and depending on the underlying operational semantics, this may lead to a new composition implementation, where the first adaptation is not complete and therefore original requirement is not satisfied; concurrently executed processes, which may lead to incorrect state and therefore the requirements of both adaptations are violated.

Similarly to concurrent systems, the source of the problem is in the way adaptation activities are modeled, i.e., how fast, atomic, and isolated they are with respect to each other, to the system execution, and to the changes in the context.

3.3 Failure: Undesired Adaptation Loops

Another potential adaptation-specific failure refers to a situation, where the execution of adaptation activities directly or indirectly leads to a state, in which the same adaptation is triggered again. This situation corresponds to an adaptation *livelock*: the same adaptation activities are identified and repeated again and again without, however, any reasonable outcome.

The adaptation loop may be entailed by the problems identified above. Indeed, when the adaptation specification is defined, an implicit assumption may expect that the adaptation activities successfully complete. In practice, however, the actions may fail and the system remains in the same situation, triggering another loop of the same adaptation activities. In other cases, concurrent events may terminate the current activity and start another one (or execute it in parallel), which is the same as the initial one.

Configuration adaptation. Configuration adaptation loops can come in the following two forms. Firstly, re-configuration of SBA may fail and the system remains in the same state, from which the adaptation started. For example, if a service becomes unavailable, adaptation initiates discovery and binding of another service corresponding to required parameters. If these requirements can not be satisfied in the current context, adaptation fails and the activities may be started again. Secondly, if the history of adaptations (configurations) is not taken into account, it is possible that the new configuration will

be equivalent to the one, for which adaptation was triggered. Accordingly, the execution of application in this new configuration may lead to the same problems, and the same adaptation will be triggered.

Composition adaptation. Similar problems occur, when the composition adaptation is considered. That is, adaptation activities associated to a particular situation may fail to bring the application to a new state. Consider, for instance, an example from the travelling domain: with a particular application failure (ticket is not available) one can associate the adaptation specification that requires executing an alternative path (find and book ticket using train reservation service). The execution of an alternative may end up in the same failure (no train ticket is available) and the SBA enters the undesired adaptation loop.

Similarly, the loop may take place if there are adaptation specifications that are defined independently, but have implicit mutual dependencies. Consider again the travel domain. One adaptation rule (from the user preferences) may require that if the total cost estimated for the whole trip is higher than a certain amount, nearby airports should be used instead. Another adaptation rule may enforce booking a taxi, if the airport is not within certain distance from the hotel. It is easy to see that under certain conditions, the first rule may trigger the second one, which will again require changing the airport.

4 Adaptation Correctness Assurance

In order to ensure that the adaptation is specified and executed correctly, the adaptation approach together with the underlying adaptation toolkit and platform should provide dedicated techniques and facilities that can uncover and mitigate the problems identified above. In this section, we first classify and discuss existing techniques for correctness assurance of service-based applications in general. Based on the capabilities of these techniques, we sketch how these can be used to address the different kinds of adaptation failures discussed above. Finally, we will highlight potential evolution of existing techniques in order to address their deficits.

4.1 Existing Means for Correctness Assurance

Classification. We will classify the techniques for correctness assurance according to the following dimensions.

First, two basic *strategies* on how to ensure correctness can be differentiated, namely constructive and analytical. *Constructive techniques* provide such form of support for the system design and execution that guarantee its correctness "by construction". That is, these techniques rely on a certain formalism and a framework, which takes the application specification and a set of predefined correctness requirements, and automatically augment it with additional facilities necessary to satisfy these requirements. *Analytical techniques* provide a possibility to analyze whether the artifacts of the system (including its specification and implementation/code) have been built in such a way as to meet certain predefined correctness properties. If potential problems are identified, the root cause for these problems is identified and the artifacts are corrected accordingly.

Table 1. Classification of Correctness Assurance Techniques

Approach	*Strategy*	*Online / Offline*	*Configuration / Composition*
monitoring	analytical	$+/-$	$+/+$
testing	analytical	$+/+$	$-/+$
simulation	analytical	$-/+$	$-/+$
verification	analytical	$-/+$	$-/+$
model-driven development	constructive	$-/+$	$-/+$
automated configuration	constructive	$+/-$	$+/-$
automated composition	constructive	$-/+$	$-/+$

Second, we can distinguish the techniques with respect to when these are employed during the life-cycle of SBAs. *Offline* techniques allow one to identify the problems before the application is put into the production mode (i.e., before the application is deployed). *Online* techniques, on the contrary, are employed while the application is executed in real settings, i.e. during the actual operation of the applications.

Third, we can distinguish between techniques that address *correctness of the configuration* (i.e., availability of services, their QoS) and techniques that address *correctness of the composition* (i.e., application behavior is correct).

Techniques. Table 1 provides an overview of the existing major approaches for correctness assurance in SBAs together with the capabilities they provide:

- *Monitoring* observes the service-based application during its *current* execution (i.e., actual operation) in order to detect deviations from the expected behavior. This includes monitoring QoS properties [11], assertions [12], or complex behavioral properties of compositions [13,14].
- The goal of *testing* is to (systematically) *execute* services or service-based applications with predefined inputs and to observe the outputs in order to uncover failures; Examples for testing techniques include [15,16].
- *Simulation* corresponds to testing of a composition specification without actually executing services [17];
- During *verification*, artifacts (e.g., a service specification) are systematically *examined* (without execution) in order to ascertain that some predefined properties are met. There exist approaches that use model checking (e.g., [18,19]) or logic-based techniques [20,21].
- *Model-driven development* aims at generating low-level specifications (closer to the implementation) given high-level models and additional transformation rules that preserve certain properties [22,23].
- During *automated configuration* a predefined abstract composition model is automatically filled with the concrete services or parameters in order to satisfy some criteria (e.g., optimization of QoS metrics) [24];
- During *automated service composition*, services are composed according to the goal of the user / designer (e.g., provide a composed travel organizer from flight, hotel, train booking services) together with additional constraints, like transactionality constraints (i.e., do not book hotel if the flight booking fails); examples for automated service composition frameworks include [25,26].

4.2 Dealing with Adaptation Failures

The techniques discussed above may also be applied to tackle the failures that are specific to the adaptation. These failures, however, pose new specific requirements, which may not be addressed with the existing techniques, and would need their evolution or even new approaches. For each of the adaptation-specific failure described in Section 3 we will discuss their requirements and the applicability of the existing approaches in these regards.

Failures due to incomplete knowledge. In the case of configuration adaptation, the problem refers to the violation of the design-time assumptions about the application configuration properties (e.g., inability to find services satisfying predefined QoS levels). In case of composition adaptation, the execution of a predefined adaptation strategy leads to unexpected and incorrect behavior, since not all the possible settings and contexts are considered at design time.

To address these problems, one of the following requirements should be met:

- (1) avoid using predefined ("hard-coded") specifications but adapt them to concrete run-time situations or contexts;
- (2) validate the realizability of the specifications before the execution / deployment of an application;

For what concerns configuration adaptation, requirement (1) may be achieved with existing techniques, such as automated application configuration through dynamic negotiation of SLAs, while requirement (2) needs novel approaches as it follows from Table 1.

For what concerns composition adaptation, the first requirement may be achieved by extending existing automated composition or model-driven techniques, in order to accommodate for potential run-time changes and events. Analogously, existing verification and simulation techniques require further extensions in order to validate adaptation specifications. In particular, there is a need to formalize the semantics of the adaptation languages and relate it to the semantics of the underlying application models; it is necessary to define "correctness" criteria for the adaptation execution; it is necessary to model and adequately represent the dynamics of the execution context.

Failures due to concurrent changes during adaptation. When the changes and relevant events occur concurrently with the execution of adaptation activities, the adaptation may become redundant, have undesirable effects or even failures. In order to prevent these problems, the adaptation framework should take potential changes and events into account when the adaptation strategy is determined. In particular, the following requirements should be addressed:

- (1) analyze the dynamics of the execution context of the application (i.e., frequency and timing of potential changes, duration of application activities);
- (2) analyze the impact of these factors on the adaptation execution in order to guide the development of proper adaptation specifications.

Existing correctness techniques (i.e., monitoring or verification) do not provide appropriate means to address those requirements and therefore should be extended. Indeed, it

Table 2. Classification of Advanced Correctness Techniques

Approach	*Strategy*	*Online/Offline*	*Config/Compos*	*Incompl. knowl.*	*Concur. change*	*Adaptation loop*
Offline adaptation analysis	analytical	$-/+$	$-/+$	$+$	$+$	$+$
Pre-deployment monitoring and testing	analytical	$-/+$	$+/+$	$+$	$+$	$-$
Online verification and simulation	analytical	$+/-$	$-/+$	$+$	$+$	$+$
Online automated composition	constructive	$+/-$	$-/+$	$+$	$-$	$-$
Built-in adaptation	constructive	$-/+$	$-/+$	$+$	$+$	$+$
Monitoring adaptation history	analytical	$+/-$	$+/+$	$-$	$-$	$+$
Stability metrics	constructive	$+/-$	$+/-$	$-$	$+$	$-$

is necessary not only to observe the relevant events and changes, but also to monitor how frequent these changes are with respect to the application activities. This information should then be modelled and used in the analysis, and, consequently, when adaptation specifications are derived.

Failure: Undesired adaptation loops. When for some reason adaptation fails or brings the application to a situation from which it has been initiated, the same set of adaptation activities may be initiated. As a result, the application enters the "adaptation loop", thus consuming resources without any effect. In order to avoid this problem, the information about previous situations and triggered adaptations should be considered when the adaptation strategy is determined. That is,

- (1) define adaptation in such a way that the loop can not appear, or
- (2) define a special mechanisms to leave the loop when it is detected.

Neither in case of composition adaptation nor in case of configuration adaptation, existing correctness techniques are enough to address these requirements. Indeed, analytical (respectively, constructive) techniques require specific means to check the specification against such loops (resp., to construct loop-free specifications and executions).

4.3 Advanced Correctness Techniques

In order to tackle the failures that are specific to the adaptation of SBA, the existing correctness assessment techniques are not applicable in the way they are used for static applications. These techniques require specific extensions, and, moreover, in certain cases novel approaches are necessary. Here we highlight future potential techniques that could be applied in combination with the SBA adaptation approaches. Table 2 classifies advanced techniques and maps them to the adaptation-specific failures.

Offline adaptation verification and simulation. The approach to extend conventional techniques with the ability to verify application specification in combination with the adaptation specification. This requires (i) representing the latter using the same formalism as for the application; (ii) representing the dynamics of the environment (changes and events) that are relevant for the adaptation; and (iii) defining specific properties to

be checked by the analysis tool. In case of problem due to incomplete knowledge, this may include the necessity to complete the adaptation execution or an ability to reach some "stable" or "correct" state. In case of adaptation loop problem, the property may express absence of such loops. In case of concurrent changes, the model and properties may express timed constraints on the adaptation behavior [27].

The most challenging problem here is to model and represent the behavior of the environment, which is often difficult to foresee for open systems, or may generate behaviors that never happen in practice.

An initial step towards offline adaptation analysis has been presented by Fugini et al. [28]. They present an approach for testing SBAs extended with fault injection mechanisms. The goal of the approach is to check if and how the application reacts to certain faults, such as delays or data errors.

Pre-deployment monitoring and testing. Here the idea is to evaluate some properties and metrics of the application and its environment before the system is ready for deployment. This may include, in particular, monitoring the QoS of potential services, evaluating characteristics of the context, estimating duration of application activities, etc. Furthermore, this kind of analysis may be applied in order to evaluate the dynamics of the application context, e.g., how often relevant changes happen, what are the ranges, etc. This information may be used to restrict the models of the environment exploited by the previous approach and, therefore, to make this models more compact and realistic.

Online verification and simulation. This technique consist of verifying or simulating the adaptation specification at run-time before its actual execution. Similarly to the offline adaptation analysis, this technique permits verifying correctness of the specification. On the positive side, it is applied in a concrete situation, and therefore is simpler to model and represent. On the downside, such analysis may slow down the application adaptation.

A sort of first attempt towards online verification of adaptation has been presented in [29]. It aims to verify correctness constraints in the scope of process migration problem. When the running process instance is dynamically changed, the proposed technique is used to check that the changes are compatible with certain constraints.

Online automated composition. This approach aims to bring the automated techniques applied at design-time to the execution phase. In this way, the adaptation specification is defined dynamically, taking into account concrete situation. However, as in the case of online analysis, such technique may considerably slow down the adaptation.

Built-in adaptation. This approach combines model-driven techniques with the automated composition approach. The adaptation specification defined at design-time is automatically composed with (i.e., built-in) the application specification. As a result, an integrated executable specification is generated that takes into account possible runtime changes and events. As in the case of offline adaptation analysis, the need to model possible events increases the complexity of the technique. On the positive side, the resulting specification is efficient with respect to online composition.

Monitoring adaptation history. One way to deal with the adaptation loop problem is to monitor and store the information about previous adaptations (adaptation history).

In particular, this information includes the event/situation, in which the adaptation was triggered, and the outcome of its execution (i.e., positive/negative, state reached, etc.). When the need for adaptation is identified, the system will compare the situation with previous histories. If the situation was already registered and the previous adaptation failed, then in the current situation some other strategy should be applied.

Stability metrics. In order to deal with highly dynamic environments, in which the changes may happen during adaptation, one can use special stability metrics. This metrics may be used in order to estimate and keep information on how stable a certain property is over time, for example, the frequency of changes in certain QoS parameter of the service involved in the composition. Such metrics would allow one to discover and bind only "stable" services, such that the adaptation is not triggered too often. Indeed, this requires specific monitoring techniques, as well as the corresponding capability of, e.g., the discovery framework.

5 Conclusions

Adaptive service-based application are often subject to specific failures and problems that are not exposed in "static" systems. In order to address these problems, novel approaches that extend both adaptation and traditional correctness assessment means are necessary. In this paper we have identified and classified adaptation-specific failures; we have also demonstrated how these failures show themselves in different adaptation approaches. Through a review and a classification of the existing correctness assurance techniques we have demonstrated that these techniques are not enough to deal with adaptation specific failures. Based on the identified gaps and requirements, we have revealed future directions and approaches that would improve correctness in the adaptive service-based applications. As a future work we would like to instantiate the proposed approaches and to integrate them within concrete adaptation frameworks.

References

1. Al-Ali, R.J., Hafid, A., Rana, O.F., Walker, D.W.: QoS Adaptation in Service-Oriented Grids. In: Middleware Workshops, pp. 200–210 (2003)
2. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: An Approach for QoS-aware Service Composition Based on Genetic Algorithms. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 1069–1075 (2005)
3. Chafle, G., Dasgupta, K., Kumar, A., Mittal, S., Srivastava, B.: Adaptation in Web Service Composition and Execution. In: Int. Conference on Web Services - ICWS, pp. 549–557 (2006)
4. Bianculli, D., Jurca, R., Binder, W., Ghezzi, C., Faltings, B.: Automated Dynamic Maintenance of Composite Services based on Service Reputation. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 449–455. Springer, Heidelberg (2007)
5. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive process management with adept2. In: ICDE, pp. 1113–1114 (2005)

6. Hallerbach, A., Bauer, T., Reichert, M.: Managing Process Variants in the Process Life Cycle. Technical report, University of Twente, TR-CTIT-07-87 (2007)
7. 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)
8. Baresi, L., Guinea, S., Pasquale, L.: Self-healing BPEL processes with Dynamo and the JBoss rule engine. In: ESSPE 2007: Int. workshop on Engineering of software services for pervasive environments, pp. 11–20 (2007)
9. Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z.: The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. Technical report (2005)
10. Lazovik, A., Aiello, M., Papazoglou, M.P.: Associating Assertions with Business Processes and Monitoring their Execution. In: Service-Oriented Computing - ICSOC 2004, Second Int. Conference, pp. 94–104 (2004)
11. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A.P.A., Casati, F.: Automated SLA Monitoring for Web Services. In: Feridun, M., Kropf, P.G., Babin, G. (eds.) DSOM 2002. LNCS, vol. 2506, pp. 28–41. Springer, Heidelberg (2002)
12. Baresi, L., Guinea, S.: Towards Dynamic Monitoring of WS-BPEL Processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 269–282. Springer, Heidelberg (2005)
13. Mahbub, K., Spanoudakis, G.: Monitoring WS-Agreements: An Event Calculus-Based Approach. In: Baresi, L., Nitto, E.D. (eds.) Test and Analysis of Web Services, pp. 265–306. Springer, Heidelberg (2007)
14. Barbon, F., Traverso, P., Pistore, M., Trainotti, M.: Run-Time Monitoring of Instances and Classes of Web Service Compositions. In: Int. Conference on Web Services (ICWS), pp. 63–71 (2006)
15. Bai, X., Chen, Y., Shao, Z.: Adaptive Web Services Testing. In: 31st Annual Int. Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 233–236 (2007)
16. Canfora, G., di Penta, M.: SOA: Testing and Self-checking. In: Proceedings of Int. Workshop on Web Services - Modeling and Testing - WS-MaTE, pp. 3–12 (2006)
17. Mayer, P., Luebke, D.: Towards a BPEL Unit Testing Framework. In: Workshop on Testing, Analysis, and Verification of Web Services and Applications, TAV WEB 2006, pp. 33–42 (2006)
18. Fu, X., Bultan, T., Su, J.: Analysis of Interacting BPEL Web Services. In: Proceedings of the 13th Int. World Wide Web Conference (WWW 2004) (2004)
19. Sharygina, N., Krning, D.: Model Checking with Abstraction for Web Services. In: Test and Analysis of Web Services, pp. 121–145 (2007)
20. Davulcu, H., Kifer, M., Ramakrishnan, I.V.: CTR-S: A Logic for Specifying Contracts in Semantic Web Services. In: Proc. WWW, pp. 144–153 (2004)
21. Grüninger, M.: Applications of PSL to Semantic Web Services. In: Proc. 1st Int. Workshop on Semantic Web and Databases, pp. 217–230 (2003)
22. Skogan, D., Gronmo, R., Solheim, I.: Web Service Composition in UML. In: Proceedings of the Enterprise Distributed Object Computing Conference (EDOC), pp. 47–57 (2004)
23. Castro, V.D., Marcos, E., Sanz, M.L.: A Model-Driven Method for Service Composition Modelling: a Case Study. Int. J. Web Eng. Technol. 2, 335–353 (2006)
24. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality Driven Web Services Composition. In: WWW 2003 (2003)
25. Marconi, A., Pistore, M., Poccianti, P., Traverso, P.: Automated Web Service Composition at Work: the Amazon/MPS Case Study. In: Int. Conference on Web Services (ICWS), pp. 767–774 (2007)

26. Berardi, D., Calvanese, D., De Giacomo, G., Mecella, M.: Composition of Services with Nondeterministic Observable Behavior. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826. Springer, Heidelberg (2005)
27. Kazhamiakin, R., Pandya, P.K., Pistore, M.: Representation, Verification, and Computation of Timed Properties in Web Service Compositions. In: Proceeding of the Int. Conference on Web Services (ICWS), pp. 497–504 (2006)
28. Fugini, M.G., Pernici, B., Ramoni, F.: Quality Analysis of Composed Services through Fault Injection. In: Business Process Management Workshops, pp. 245–256 (2007)
29. Ly, L.T., Rinderle, S., Dadam, P.: Integration and Verification of Semantic Constraints in Adaptive Process Management Systems. Data Knowl. Eng. 64, 3–23 (2008)