Chapter 5
Parallel Adaptation of Multiple Service Composition Instances

Rafael Roque Aschoff, Andrea Zisman, and Pedro Alexandre

Abstract Existing approaches for adaptation of service compositions do not consider the fact that common services can be used in different compositions, and, therefore, a problem that may be identified in one composition could be used to predict unwanted situations in other compositions. In this paper, we propose a parallel and proactive adaptation framework that supports proactive adaptation in multiple service composition instances at the same time. In the framework, events observed for one particular service composition instance are shared between all composition instances executed in parallel in order to better predict problems and rectify them in all necessary instances, when possible. The parallel characteristic of the framework also supports balancing the load among candidate service operations, and, therefore, it considers the maximum expected service operation throughput between the compositions. A prototype tool has been implemented to illustrate and evaluate the framework in different scenarios.

5.1 Introduction

Adaptation of service compositions is considered a major research challenge for service-based systems [6, 7, 14, 19]. Several situations may trigger the need for adaptation in service compositions, including (i) changes in or emergence of new requirements, (ii) changes in the context of the composition and participating

R. R. Aschoff (✉)
Federal Institute of Pernambuco - IFPE, Pernambuco, Brazil
e-mail: rafael.roque@palmares.ifpe.edu.br

A. Zisman
The Open University, Milton Keynes, UK
e-mail: andrea.zisman@open.ac.uk

P. Alexandre
University of Sao Paulo, São Paulo, Brazil
e-mail: pedro.alexandre@usp.br

Y. Yu et al. (eds.), *Engineering Adaptive Software Systems*,
https://doi.org/10.1007/978-981-13-2185-6_5

services, (iii) changes in functional and quality aspects of services in compositions, (iv) failures in services in compositions, and (v) emergence of new services.

More recently, some approaches to support adaptation of service composition in a reactive way [1, 4, 9, 15, 18] or proactive way [5, 22] have been proposed. However, these approaches support changes in service compositions in future executions of the composition, instead of changes in compositions during their execution. Moreover, existing approaches allow changes to be performed only in a single composition and do not consider the fact that services that need to be replaced may be participating in different compositions at the same time.

In this paper we propose a framework called ParProAdapt to support *parallel and proactive adaptation* of service compositions. The work presented in this paper extends our previous work [2, 3] that supports proactive adaptation of service composition in order to allow parallel adaptation and load balancing management of service compositions. We define *parallel adaptation* of service compositions as changes in service compositions that are being executed at the same time and which share common service operations that need to be replaced. Our approach supports the situation in which common operations are being used in different compositions, and these operations may need to be replaced in the different compositions and not necessarily only in the composition where the need for changes has been identified, for example, when the service providing the operation becomes unavailable or when the operation malfunctions. The approach also supports the situation in which several execution instances of the same composition are executed concurrently, and a problem identified in one instance also needs to be rectified in the other instances of the composition.

Another novelty of the work presented in this paper is the support for *load balancing* management. More specifically, when performing changes in a service composition, the load of a particular invoke activity can be distributed over different candidate service operations in order to increase or maintain the total throughput of the composition. If a deployed service operation is unavailable, and there are no candidate operations with the same expected throughput, a combination of more than one candidate operation can be considered.

In order to illustrate, suppose a credit card service S_{CC}, with an operation to make a payment O_{Pay}, that is used in compositions C_1, C_2, and C_3. Assume that S_{CC} becomes unavailable and this is identified when trying to invoke O_{Pay} in composition C_1. Consider that O_{Pay} has not yet been invoked during the execution of C_2 and C_3. In this case, O_{Pay} should be replaced in C_1, to allow the composition to continue its execution, as well as proactively replaced in C_2 and C_3, to avoid invoking O_{Pay} in these two compositions, and only after attempting to invoke O_{Pay}, the process realises that O_{Pay} is unavailable. The situation described above is not unrealistic since it is expected that services will be used in several applications at the same time.

In the framework the *proactive adaptation* of service compositions consists of detecting the need for changes and implementation of changes in a composition, before reaching an execution point in the composition where a problem may occur, for example, the identification that the response time of a service operation in

a composition may cause violation of the composition's service-level agreement (SLA), requiring other operations in the composition to be replaced in order to maintain the SLA, or the identification that a service provider P is unavailable requiring other services in the composition from P to be replaced, before reaching the parts in the composition where services from P are invoked.

In ParProAdapt the prediction of problems that trigger the need for adaptation is based on *function approximation* and *failure spatial correlation* techniques [16]. Moreover, the need for adaptation considers a group of operations in a composition flow, instead of isolated operations, in order to avoid replacing an operation in a composition when there is a problem, and this problem can be compensated by other operations in the composition flow.

The remainder of this paper is structured as follows. In Sect. 5.2 we present an overview of the ParProAdapt framework and provide a description of the proactive and parallel adaptation approaches used in the framework. In Sect. 5.3 we describe implementation and evaluation aspects of our work. In Sect. 5.4 we give an account of related work. Finally, in Sect. 5.5 we discuss concluding remarks and future work.

5.2 Parallel and Proactive Adaptation Framework

The main goals of the ParProAdapt framework is to provide dynamic, proactive, and parallel adaptation of service compositions. It supports a parallel identification and prediction of the need for adaptation and an autonomously reconfiguration of the service compositions during their execution time. The parallel characteristic of the approach is concerned with the identification of a problem in an instance of a composition and the impact of this problem in other instances of the same composition or in different compositions that share common services when the identified problem is in any of these services.

ParProAdapt is based on an event-based strategy in which different components of the framework generate different types of events. It supports parallel and proactive adaptation of service compositions due to four different types of situations, namely, C1, events that cause the composition to stop its execution (e.g. unavailability or malfunctioning of a deployed service operation); C2, events that allow the composition to continue to be executed, but not necessarily in its best way (e.g. the network link is congested, causing delays on the response times of some operations; such fluctuations in the response time may require adaptation in order to comply with SLA parameters of the composition); C3, emergence of new requirements (e.g. messages exchanged between services need to be encrypted; the response time of the composition needs to be improved); and C4, emergence of better services (e.g. a cheaper service becomes available).

The above situations are mapped to different events that are analysed in terms of the need for adaptation, and, depending on the results of the analysis, the adaptation process is executed. The adaptation consists of creating a valid configuration for a composition by (a) replacing a single service operation in the composition by another service operation or by a group of dynamically composed service operations

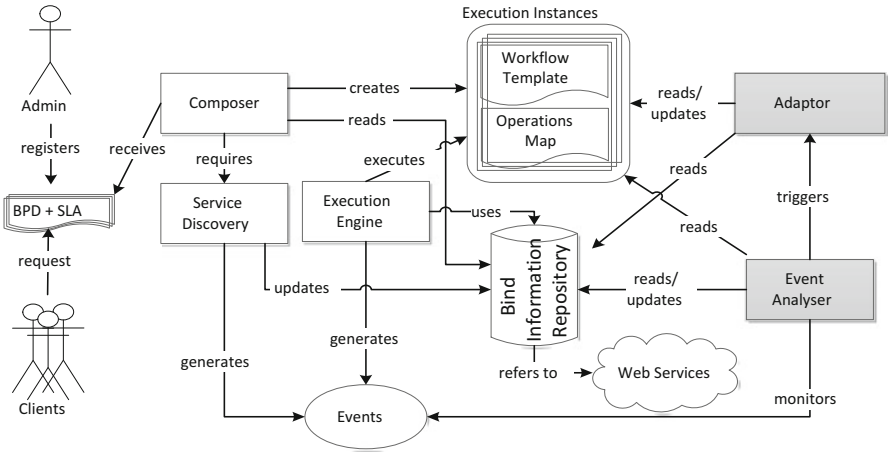

Fig. 5.1 ParProAdapt framework architecture

(replacement of types 1-1 or 1-n) or (b) replacing a group of service operations in a composition by a single operation or by a group of dynamically composed service operations (replacement of types n-1 or n-m).

The replacement of an operation may cause *signature dependency* issues with other operations in the execution instance, i.e. the situation in which the output parameter (or its part) of an operation is used as input parameter (or its part) in another operation. In the case of operation signature dependency issues, it is necessary to verify the need to replace affected operations.

The creation of a valid configuration for a composition considers the execution logic (regions) of the composition (sequence, parallel, conditional selection, and repeat) to identify a group of operations that may need to be replaced by an operation or a group of dynamically composed operations. The creation of a valid configuration is considered an optimisation problem based on the selection of appropriate combinations of candidate service operations that satisfy the SLA parameters of a composition.

Figure 5.1 shows an overview of ParProAdapt framework. The framework is composed of five mains components, namely, *composer*, *service discovery*, *execution engine*, *event analyser*, and *adaptor*, described below.

Composer This component is responsible to parse business process definitions (BPDs) (service compositions) and their associated service-level agreements (SLAs) and create an internal configuration for the service composition using the service discovery and *bind information repository*. This configuration is a *service composition execution instance*. The composer invokes the service discovery component to identify service operations that implement the logic of the service compositions and satisfy the SLA parameters of the compositions. Different configurations of a service composition execution instance may be created for the various clients requesting the composition.

Execution Instance It is composed of (i) a logic workflow of a service composition, which defines abstract operations, their order of execution and dependencies between operations and (ii) a map between the abstract operations in the workflow and the binding information for the actual services. An execution instance extends the expressiveness of a service composition with information about the (i) execution flow, (ii) deployed endpoint service operations and their locations, (iii) state of a service operation in a composition (e.g. executed, to be executed, and executing), (iv) observed QoS values of a service operation after its execution, (v) expected QoS values of a service operation, and (vii) SLA parameter values for the service operations and the composition as a whole.

Service Discovery This component identifies possible candidate service operations to be used in the composition, or to be used as replacement operations in case of problems. We assume the use of the service discovery approach that has been developed by one of the authors of this paper to assist with the identification of candidate service operations [22]. This approach advocates a proactive selection of candidate service operations based on distance measurements that match functional, behavioural, quality, and contextual aspects. The candidate service operations are identified in parallel to the execution of the compositions based on subscribed operations and are kept in a local *bind information repository*.

Bind Information Repository It keeps track of all possible service operations to be used by the service compositions, including not only the deployed operations, but also candidate service operations. The repository also contains information about the expected QoS parameters of the operations and their status (e.g. available and unavailable). The service discovery component updates the repository with information about new identified service operations or new status of already identified operations. When new operations are identified, or there are changes in the status or characteristics of existing operations, an event about the changes is generated and handled by the *event analyser* component.

Execution Engine An *execution engine* is the piece of software responsible for the execution of business processes described in the form of an executable service composition. Different service compositions can be deployed in an execution engine, and for each request of a particular composition, a *private session* must be maintained in order to individually and correctly parse input and output parameters. In the same way that a web service description (WSD) contains an abstract part for the general definitions of a web service and a concrete part for the binding information, for each service composition SC_n deployed in an execution engine, there is an abstract *composition template* T_n consisting of the workflow logic and a set of binding information for each deployed service operation S_{T_n}.

The abstract template T_n contains invoke activities pointing to abstract web service definitions. While executing a services composition SC_n, the execution engine uses the binding information S_{T_n} to identify the actual concrete operation to be invoked. Without a way to dynamically update the structural logic (T) or the binding information (S_T), compositions are bound to use the same set of

concrete operations, which results in great issues when such operations degrade their performance or present any fault.

We developed a simple execution engine that handles execution instances independently. It identifies service operations to be used and how they should be accessed. Before invoking a service operation, the execution engine requests the status of the operation and the status of the composition as a whole (e.g. when the response time for the whole composition violates the SLA parameter of the composition). In the case in which a service operation is unavailable, or there is not a match between the expected and observed QoS values of an operation, a new event is created and sent to the event analyser.

Event Analyser This component is responsible for analysing all the generated events in order to predict unwanted situations and execute parallel changes in the execution instances, when necessary. More details of the functionality of this component are discussed in the subsections below.

Adaptor This component is responsible to execute individual changes in the execution instances, based on requests received from the event analyser. In order to execute the necessary changes, the adaptor component reads information from the bind repository about available operations.

5.2.1 Proactive Adaptation Approach

The proactive adaptation approach used by the framework has been described in details in [2, 3]. In this section we provide an overview of the approach for completeness of this paper and to better understand the parallel characteristics of the approach, which is the novel aspect of the paper.

As described above, the adaptation process may be triggered by situations of types C1 to C4. The events generated for situations C1 to C3 may produce unwanted situations resulting in failure in the execution of service compositions. Due to the nature of situations, C3 and C4, they cannot be predicted. However, prediction techniques can be used to support situations C1 and C2. For any of the situations that may trigger the need for adaptation, the process tries to identify other parts in the execution instance that may be affected by the situation. The process is based on the use of two techniques executed by the event analyser component, namely, (a) QoS analysis and (b) spatial correlation analysis.

QoS Analysis It consists of a failure prediction technique that verifies the impact that changes of QoS values of deployed service operations may have in the SLA parameters of a composition as a whole. This analysis is used to avoid replacing an operation in an execution instance when the problem can be compensated by other operations in the execution flow. The process also identifies other operations in the instance that may be affected due to violation of QoS values.

In the framework, the process concentrates on the analysis of violations of response times and cost values of the operations. The analysis is based on the use of exponentially weighted moving average (EWMA) [13] for modelling the expected service operation QoS values. An expected QoS value (e.g. response time) of an operation is calculated based on previous observed QoS values for that operation. The new expected QoS value of an operation is updated on the bind information repository. The aggregated QoS values of an execution instance is calculated based on the expected QoS values of the operations not yet executed, and the observed QoS values of the operations are already executed. The computation of the aggregated QoS values for the whole composition depends on the type of the QoS values and the logic workflow structures of the composition (e.g. conditional, sequence, parallel, and repeat logic structures).

When there is no violation of the SLA values for the whole composition, there is no need to adapt the execution instance. If the expected values are violated, the adaptor component is invoked to identify a valid configuration for the composition. This valid configuration may be generated by replacing operations in the execution instance that have not yet been executed and by attempting to find possible combinations of replacement operations that provide the functionality of those operations and maintain the SLA values of the composition.

Spatial Correlation Analysis This technique consists of identifying spatial correlations between operations, services, and providers. It is concerned with the situation in which providers, services, and operations become unavailable and the impact that this unavailability may have in other services or operations being used in the composition. For example, consider a service S that becomes unavailable. In this case, the process considers all other operations of S in the composition since these operations may not be able to be executed. Similarly, when a provider P is unavailable, all services and operations provided by P are also marked as out of reach on the bind information repository.

During the spatial correlation analysis, the bind information repository is updated about the availability of operations. In the case in which operations deployed in the execution instances are identified as unavailable, the adaptor is invoked to identify a valid configuration for the composition. In the spatial correlation analysis, all running execution instances are aware of any issues when trying to invoke operations with a problem.

When using only the proactive adaptation approach, for the trigger situations C1 and C2, the running execution instances identify a problem with an operation only when they reach a point of execution in which they request the operation with the problem. The other parts of the execution instances that may be affected by the operation with a problem will be proactively identified based on the techniques discussed above. However, to allow running execution instances to be notified about a problem in a deployed operation, as soon as possible, for any of the trigger situations, we propose the parallel adaptation approach described below.

5.2.2 *Parallel Adaptation Approach*

The parallel adaptation approach complements the proactive adaptation approach with two new techniques, namely, (a) parallel analysis and (b) load balancing analysis. Overall, the idea of the parallel approach is to identify other running execution instances that may be affected by a problem identified in one execution instance and rectify this problem in these other instances in parallel, during their execution time. As mentioned before, those execution instances can be copies of the same service composition in which a problem was identified or different service compositions that use an operation for which there is a problem.

Parallel Analysis With this technique, it is possible to reduce the time that it is necessary to identify a problem in an operation used in a service composition, the assessment of this problem in other running service compositions that share the operation and the execution of the actions to rectify the problem.

Our approach allows instances of service compositions to be adapted in parallel independent of each other. More specifically, changes executed in one service composition instance do not necessarily interfere with other instances which are executed in parallel, even when these are instances of the same service composition.

Figure 5.2 presents a snapshot of the above characteristics of our approach. As shown in the figure, for each request m of a deployed service composition SC_n, an execution instance EI_m^n is created using the composition template T_n and its respective binding information S_{T_n}. The framework creates for each execution instance EI_m^n a private template T_m^n and binding information for this template $S_{T_m}^n$

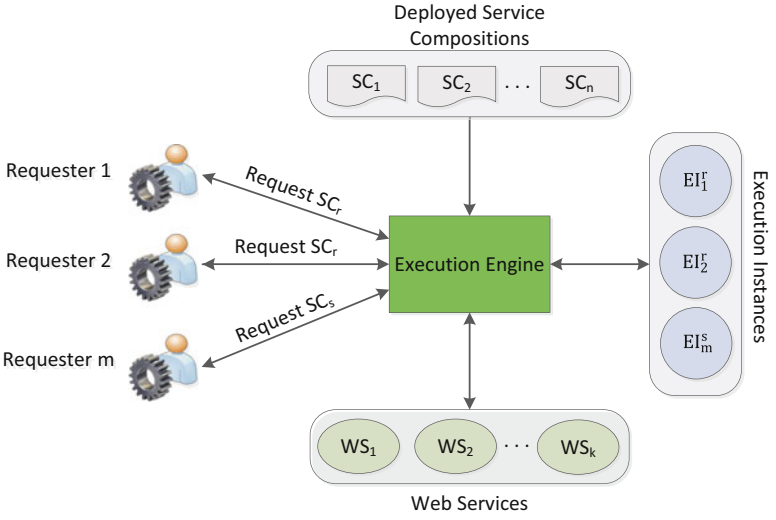

Fig. 5.2 Illustration of the execution engine accessing execution instances of service composition

in order to allow adaptations of a particular service composition instance. The approach also maintains the composition template T_n and its binding information S_{T_n} to support proactive adaptation of new instances of a service composition that may be created due to future requests. In order to illustrate, consider three execution instances EI_1^r, EI_2^r, and EI_1^s for service compositions SC_r and SC_s respectively. Changes in the private template T_1^r of EI_1^r or changes in its binding information $S_{T_1}^r$ do not create direct changes in the private templates T_2^r and T_1^s or in the binding information $S_{T_2}^r$ and $S_{T_1}^s$. However, it is possible to allow the adaptation across parallel execution instances of the same or different service compositions by accessing their private templates and binding information. Moreover, given a set of deployed service compositions $\{SC_1, SC_2, \ldots, SC_n\}$, new execution instances of these compositions can benefit from previous processed information by changing the respective composition templates $T_x, 1 \leq x \leq n$ or the respective default binding information $S_{T_x}, 1 \leq x \leq n$, which are both used to create the new execution instances.

As described in Sect. 5.2.1, in the proactive approach, the verification of the status of deployed operations is only executed when a running execution instance reaches a point of execution in which a deployed operation is requested. With the parallel analysis, the event analyser component triggers parallel adaptation of all affected execution instances. This allows parallel execution instances to reconfigure themselves earlier in the running process (before reaching the operation with issues) and, therefore, augment the probability of success in the adaptation since there will have potentially more options for changes in the composition, for example, in the case in which it is necessary to change a group of operations in the composition that have not yet been executed, in order to conform to the SLA values of the composition.

The parallel analysis is executed by verifying if each running execution instance has a valid configuration, before the execution of each deployed operation in the instances. The verification of a valid configuration consists of analysing if there is any operation not yet invoked that may have become unavailable and if there are any SLA violations due to QoS discrepancies. This verification is executed by the event analyser based on the information in the execution instances and the bind information repository (see Fig. 5.1). During the above verifications, an execution instance cannot proceed with its execution until either an adaptation is performed or it is concluded that there is no need for adaptation.

Load Balancing This technique is used to verify if the throughput of the service compositions are maintained as initially specified for the compositions. The throughput specified for a service composition is reflected in the activities and their deployed operations in the composition. The throughput of each service operation in an execution instance is calculated and compared with the maximum accepted throughput value of the composition, in order to avoid overloading the use of the deployed operations. This is done by using a throughput counter for each deployed operation. When an execution instance is created, the counters associated with the operations are incremented; when the operations are invoked during the execution

of an instance, their associated counters are decremented. The maximum accepted throughput value of an operation is maintained in the bind information repository to allow the composer and adaptor components know which operations can be used in an execution instance, without causing operation overload.

In the case in which a deployed operation O needs to be replaced, the approach supports the use of one or more operations to replace O when these operations provide the same functionality of O and the sum of the throughput values of these operations are equal to the throughput value specified for the activity associated with O. The above is possible due to the parallel approach being described in the paper since the framework keeps track of the parallel use of all operations in the execution instances that are running at the same time.

5.3 Implementation and Evaluation

In order to demonstrate and evaluate the work, we have implemented a prototype tool of the framework in Java. The tool assumes service compositions in WS-BPEL [20] exposed as web services using SOAP protocol, and participating operations and user requests emulated using SoapUI. The service discovery tool was also implemented in Java and is exposed as a web service.

In our previous paper [3], we showed how computationally inexpensive and scalable are the various activities concerned with the proactive adaptation aspect of the framework for a single service composition. In particular, we analysed the time to identify and resolve SLA violations, the time to identify and resolve signature dependencies, the time to identify spatial correlations, and the time to adapt a composition by changing groups of operations in a composition. In the current parallel approach, the activity that generates additional computational effort is concerned with the reconfiguration algorithm of a service composition and its additional analysis of the load of operations. Therefore, we consider that the parallel extension is aligned with our previous results with respect to the computation of these activities. In this paper, our focus is to demonstrate if there are improvements in the adaptation process when considering the parallel adaptation, in terms of the number of service compositions that can adapt successfully. In other words, we are trying to verify if our approach is able to improve the dependability of service compositions by dealing with two specific types of problems, namely, (a) a problem that can only be solved by changing the identified faulty operation and a set of other operations which are logically presented in the compositions prior to the faulty one and (b) a problem where there is no single operation that can support the number of requests being generated.

As previously discussed, our approach is able to adapt parallel execution instances of single or multiple service compositions. The adaptation process itself, however, makes no difference if the execution instances are of the same or distinct compositions. Such distinction exists only during the initial phase when the execution instance must be created based on the template of a deployed composition.

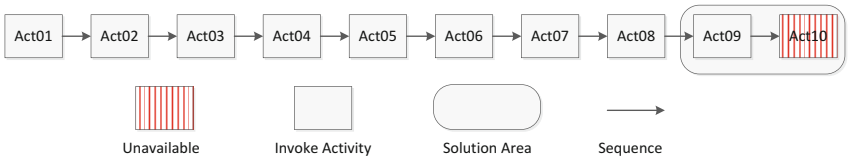

Fig. 5.3 Service composition workflow for evaluation of Scenario 1

In order to make the evaluation and discussion of the results more clear, we decided to conduct our experiments with multiple instances of the same service composition.

In the experiments, we assume that each of the various execution instances starts its execution in different time steps. We also consider that the number of running execution instances at different points of their execution flow is approximately the same. More specifically, the number of running instances executing the initial part of their flows is similar to the number of those in the middle or in the end of their execution flows. The work has been evaluated for three main cases with different scenarios. In the first two cases, we compare the use of the parallel and proactive approach with the proactive approach only for a service composition with a linear structure (Case 1) and a complex service composition (Case 2). For both cases (Case 1 and Case 2), we assume that one or more operations in the composition become unavailable. However, the approach supports a similar process for the other types of problems (e.g. violation of QoS values of an operation). Finally, in Case 3 we evaluate the load balancing technique.

Case 1 – Scenario 1: In this scenario, we use a service composition with a sequential workflow formed by ten *invoke activities*, as shown in Fig. 5.3. We assume that at a certain time in the experiment, the service operation assigned to the last invoke activity (Act10) becomes unavailable. Consider the existence of a set of candidate service operations for each invoke activity (Act1–Act10) presented in Fig. 5.3, and the use of any of the available candidate service operations for Act10, along with the current assigned operations for (Act1–Act09), would cause a violation of the SLA value of the whole composition. Consider the existence of a valid configuration for the service composition when replacing both the operations assigned for activities Act9 and Act10.

We compared a number of execution instances that (a) were able to adapt successfully (successful), (b) were not able to adapt (unsuccessful) and (c) did not require adaptation because they were not affected by the problem (not required), for the case in which we used the parallel and proactive approach with the case in which we used only the proactive approach. We considered 50, 100, 150, and 200 execution instances of the composition shown in Fig. 5.3.

Figure 5.4 presents the results of this experiment. For each different number of execution instances considered in the experiment, the first column represents the results when using the parallel and proactive approaches (specified as *parallel* for simplicity), while the second column represents the results when using only the proactive approach (specified as *proactive* for simplicity).

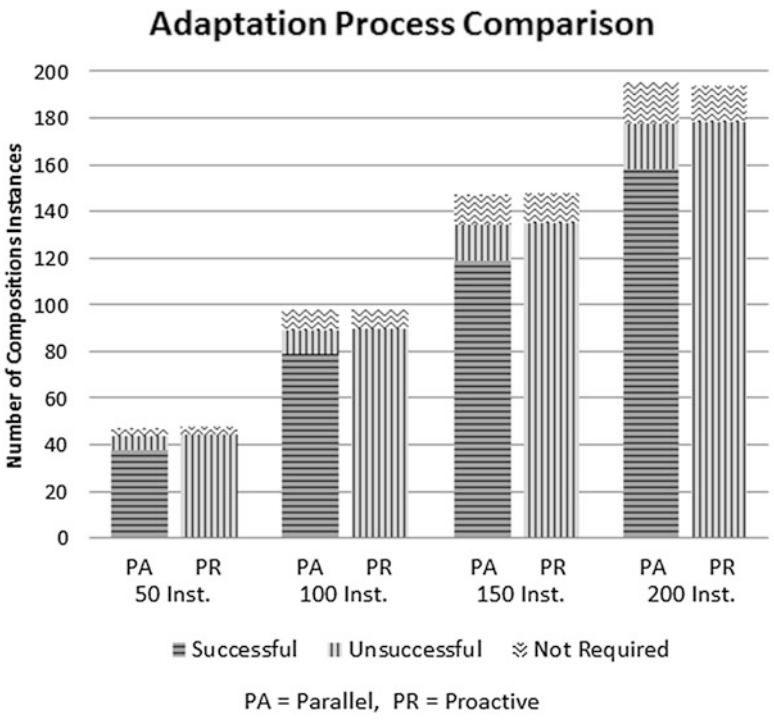

Fig. 5.4 Comparison of the adaptation process for Case 1 – Scenario 1.

As shown in Fig. 5.4, when using the combined parallel and proactive approaches, there are many more instances that are adapted and finished successfully. This is because in the parallel approach, several execution instances that are still in operation are notified about the unavailability of the operation associated with Act10 and have not yet executed the operation associated with Act09. Contrary, in the case when only the proactive approach is used, the adaptation process is attempted when the execution process tries to invoke the operation associated with Act10 and realises that this operation is unavailable. In this scenario, the process requires the replacement of the operation associated with Act09 as well. However, when attempting to invoke the operation associated with Act10, the operation for Act09 has already been executed and cannot be replaced.

Figure 5.4 also shows that even when using the proactive approach only, some instances are able to finish successfully for all the different numbers of execution instances used in the experiment. These instances are the ones that managed to invoke the operation associated with Act10 before this operation became unavailable and, therefore, were able to finish their execution successfully.

Case 1 – Scenario 2: In this scenario, we use the same service composition of Scenario 1, but we consider different positions in the composition where the operation associated with an activity becomes unavailable. We consider the

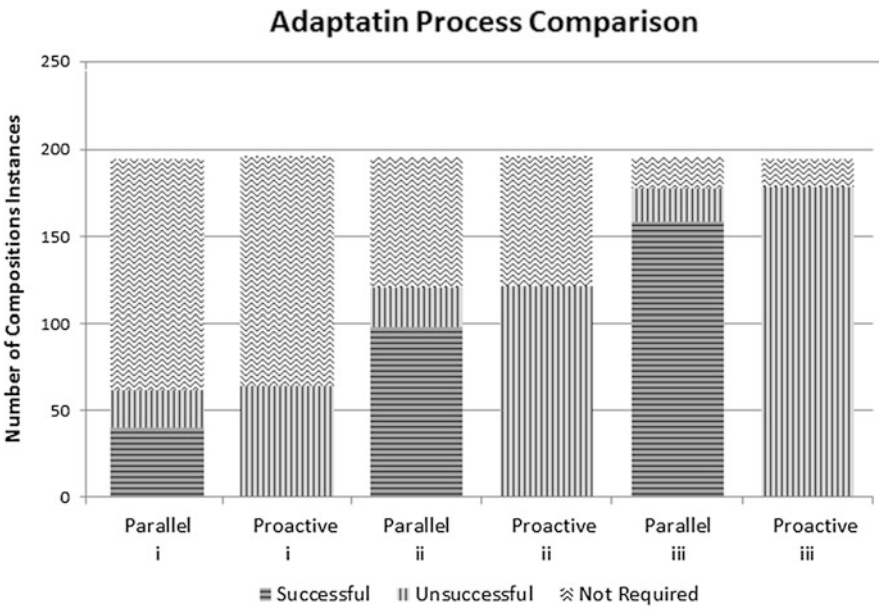

Fig. 5.5 Comparison of the adaptation process for Case 1 – Scenario 2

situations in which the operations associated with activities Act04, Act07, and Act10 become unavailable at the same time. In all three cases, we assume that a valid configuration exists when replacing both the operation that becomes unavailable and the ones associated with the previous activity of the unavailable operations, i.e. operations associated with (i) Act03 and Act04, (ii) Act06 and Act07 and (iii) Act09 and Act10. We assume 200 instances of the service composition executed at the same time.

Figure 5.5 shows the results of the experiments for situations (i) to (iii) above. As shown in the figure, when using only the proactive approach, in any of situations (i) to (iii), none of the execution instances could be successfully adapted. This is because the execution instances have already invoked the operations associated with the activities that occur before the activities that become unavailable (activities Act03, Act06, and Act09).

The results also show that the number of execution instances that do not require adaptation decreases when the problem occurs at a position closer to the end of the composition.

The number of unsuccessful adaptation instances, however, increases. This is due to the number of running execution instances that are at a point *before*, *the same*, or *after* the point in which the problem is identified, during their execution. This also explains the reason for having similar numbers of execution instances that do not require adaptation, for the parallel and proactive approach and the proactive approach only, in situations (i) to (iii).

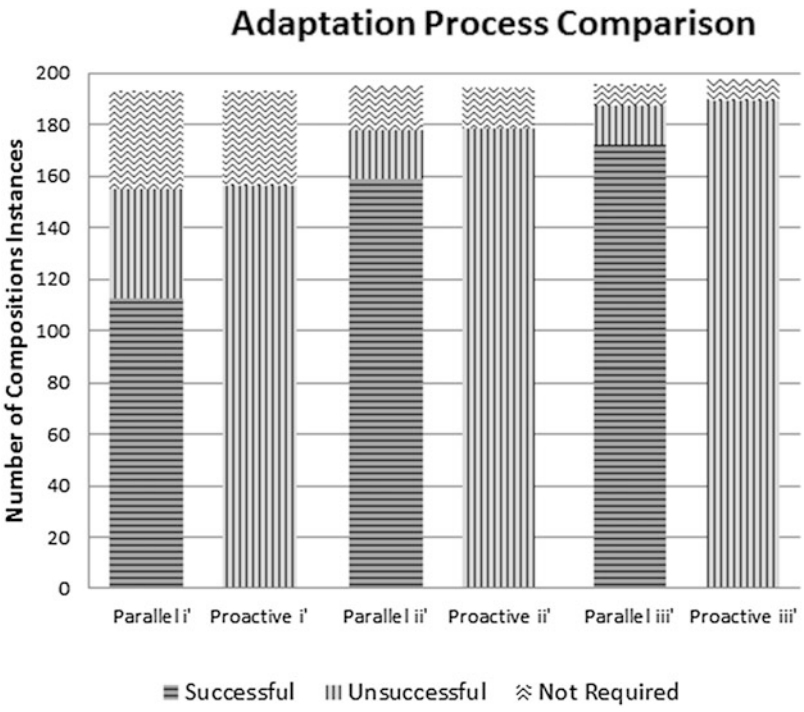

Fig. 5.6 Comparison of the adaptation process for Case 1 – Scenario 3

From Fig. 5.5 we observe that when using the parallel and proactive approaches, the number of successful adaptation instances increases, as the problem occurs at a position closer to the end of the composition. This is because the number of execution instances that can be adapted increases, since there are more instances at execution points before the operation becomes unavailable.

Case 1 – Scenario 3: In this scenario, we compare the approaches when using service compositions with a sequential structure, as in Scenarios 1 and 2, but of different sizes. We considered compositions with (i') 5, (ii') 10 and (iii') 15 activities. Similar to the above scenarios, we assume that in each of the three compositions, the operation associated with the last activity becomes unavailable and that a valid configuration exists when replacing both the operation that becomes unavailable and the operation associated with the previous activity. We assume 200 instances of the service composition executed at the same time.

Figure 5.6 shows the results of the experiments for compositions (i') to (iii'). The results in the figure show an increase in the number of execution instances that required adaptation as the size of the compositions increase. As in the case of Scenario 2, this is due to the number of running execution instances that are at a point before, the same, or after the point in which the problem is identified, during their executions. Similarly, the results show an increase in the number of successful

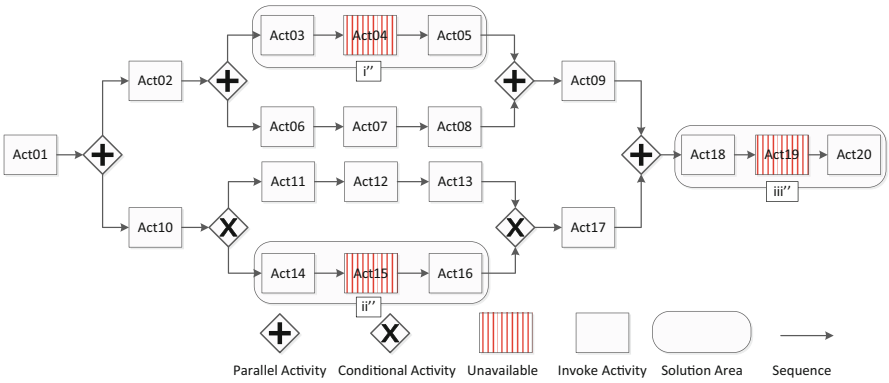

Fig. 5.7 Complex service composition workflow

adaptations for bigger compositions. Similar to Scenarios 1 and 2, the results show that when using only the proactive approach, in any of situations (i') to (iii'), none of the execution instances could be successfully adapted.

Case 2: In this case we use the service composition shown in Fig. 5.7. We consider that the operations associated with activities Act04, Act15, and Act19 become unavailable. We also assume that for each unavailable operation, the solution of a valid configuration exists when replacing the operations associated with the previous and next activities of the unavailable operation and the operation that becomes unavailable. We assume 100 instances of the service composition executed at the same time.

The example in Case 2 differs from the scenarios in Case 1 since (a) the service composition is more complex with more activities organised in different execution logics (conditional and parallel), (b) a valid configuration for the composition includes the replacement of operations associated with activities before and after the operation that becomes unavailable and (c) the operations that become unavailable are associated with activities in different execution logics.

The results of this experiment are shown in Fig. 5.8 for the unavailability of (i") Act04, (ii") Act15, and (iii") Act19. As it was expected, when the operation that becomes unavailable is at the end of the composition (situation (iii")), a larger number of execution instances require adaptation since there are more running instances at execution points before the operation becomes unavailable (as in the previous scenarios). The results show that for situation (i"), half of the execution instances did not require adaptation. For those execution instances that required adaptation, half of them were successfully adapted.

We also observe that situation (i") required more instances to be adapted than situation (ii"). This is due to the fact that situation (ii") is a conditional execution logic, and, therefore, not necessarily all the execution instances will execute this path in the composition. This is not the case in situation (i") in which all the instances need to execute the respective path in the composition.

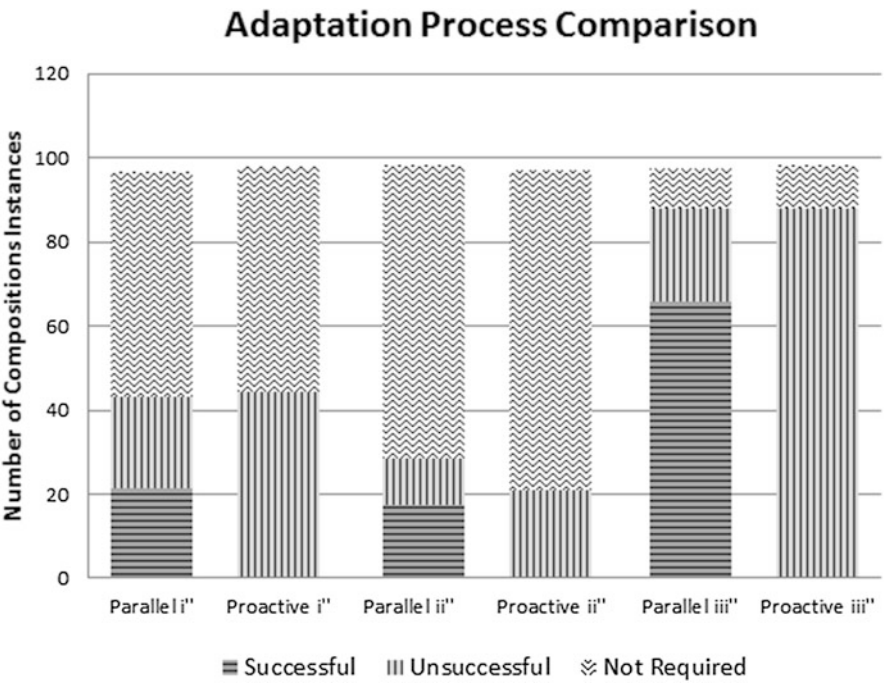

Fig. 5.8 Comparison of the adaptation process for Case 2

Case 3: In our approach, the efficiency of the proposed technique to dynamically distribute the load of service operation requests among different service providers, and in parallel with the execution of service composition instances, depends on the number of requests for a particular service operation and the capacity of the service operation to fulfil its requests. The size, complexity and logic of a service composition do not cause impact to the load balancing technique. Therefore, in order to evaluate the load balancing technique, we used a simple service composition. More specifically, the evaluation was executed in a scenario with a single invoke activity (IA) deployed in two operations given by two different providers P_1 as OP_1 and P_2 as OP_2. We assumed both OP_1 and OP_2 configured with a processing time of 1 s. Moreover, in the experiment we used a maximum of 20 concurrent service composition instances and configured OP_1 and OP_2 to be able to handle up to ten concurrent requests.

In order to introduce some random behaviour in the income rate of operation requests, we simulated the compositions requests and assumed that each request respects a uniform distribution with minimum zero and maximum one. In other words, each of the 20 parallel processes generating concurrent requests sleeps for a specific amount of time and generates a new request. After that, the process starts again if the experiment is not over. This behaviour is depicted in Fig. 5.9.

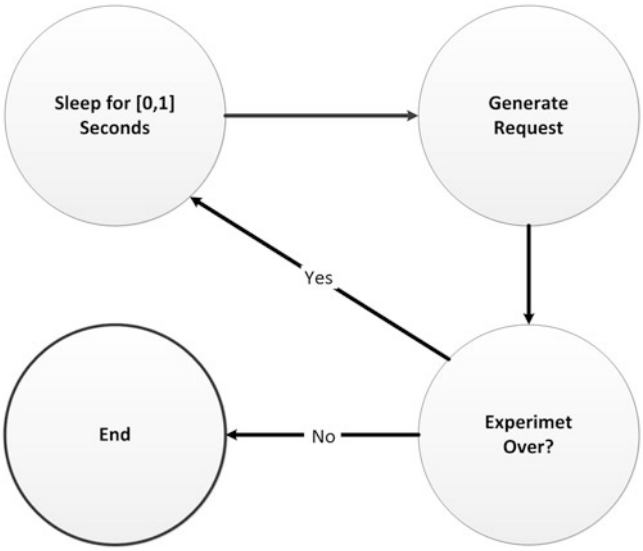

Fig. 5.9 State machine of the concurrent requests generator process

In the above-described experiment, we expected to observe an improvement in the overall performance of the execution engine in terms of the number of successfully concluded composition requests. The basic idea is that if no distribution of the load is in place, the best thing that an approach can do is to jump from one operation to another as soon as it is detected as unavailable (e.g. an operation that is not responding due to high traffic). Moreover, considering that no single operation is suitable to answer all concurrent instances, it is almost mandatory to employ some form of online testing to discover if the operation becomes available again. Without a way to assess the availability of an operation, the composition instances would just fail to be created since the system would indicate that no operation is available to perform the required tasks. We implemented a basic online testing procedure that periodically checks if the previously failed operation is available and marks it in the local repository as available again.

In our experiments, using the parallel adaptation with load balancing techniques, the average time to finish an execution instance was about 1 s. This was expected since the processing time of both OP_1 and OP_2 is configured as 1 s. Moreover, there was only at most 20 concurrent requests, and the combined throughput for OP_1 and OP_2 was 20. Therefore, no extra issues were introduced. We noted that in the case in which the ability to distribute the load between different operations was turned off, the average time to conclude an execution instance rose to about 3 s. This was due to the fact that now the adaptor component had to constantly face an error due to the high load of requests in either OP_1 or OP_2. Figure 5.10 presents a snapshot of the distribution of the requests made to OP_1 in both experiments. As we can see, when the load balancing technique is in place, the load distribution is much more

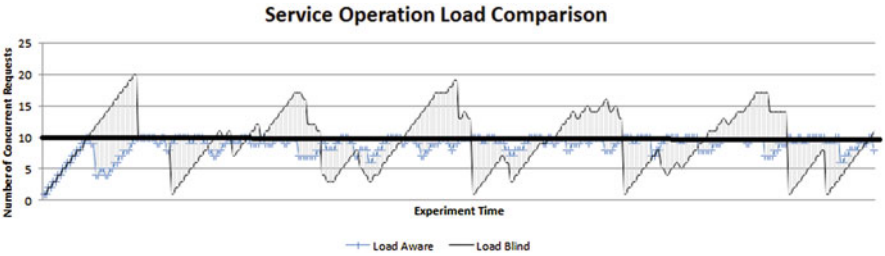

Fig. 5.10 Comparisons of the distribution of operation request for a single provider between the approach with and without load balancing

homogeneous. The black line at ten concurrent requests indicates the threshold for the capability of OP_1. Given that the approach with load balancing respects the threshold for individual operations by identifying its maximum capability, no issues are introduced. However, when such awareness is removed, and there is no way to alleviate the load, the threshold is not respected. This causes errors and requires adaptations to be executed.

5.4 Related Work

The work presented in this paper is concerned with approaches that support dynamic adaptation of service compositions, which is considered a major research challenge for service-based systems [6, 7, 14]. Initial approaches were proposed to support adaptation of service compositions in a reactive way [1, 4, 9, 15]. These approaches support adaptation of service composition based on predefined policies [4], self-healing of compositions based on detection of exceptions and repair using handlers [15], context-based adaptation of compositions using negotiation and repair actions [1] and key performance indicator analysis [9].

Other approaches have recently been proposed to support adaptation of service compositions in a proactive way [2, 3, 5, 10, 11, 19]. The work by Dai et al. [5] uses semi-Markov models for performance predictions, service reliability model, and minimization in the number of service reselection in case of changes. The decision to adapt is based on the performance of a single service. One of the first works to use a proactive approach is PREvent [10], which was designed to support prediction and prevention of SLA violations in service compositions based on event monitoring and machine learning techniques. The works by Metzer et al. [11] and Tosi et al. [19] advocate the use of testing to anticipate problems in service compositions and trigger adaptation requests. However, the creation of test cases is not an easy task.

Approaches to support multilayered monitoring and adaptation of service compositions have been proposed [8, 17, 21]. Some of these approaches use the concepts of adaptation taxonomy and templates (patterns) created during design time to

represent possible solutions for adaptation problems [17]. Other approaches rely on dynamic identification of cross-layered adaptation strategies for software and infrastructure layers [8, 21] or on the use of aspect-oriented techniques to support adaptation of compositions due to QoS aspects [12].

Our framework differs from the above approaches since it supports parallel adaptation of running execution instances. In addition, it allows for parallel and proactive adaptation of service compositions due to different types of problems and provides different ways of adapting the compositions.

5.5 Conclusions and Future Work

In this paper we described ParProAdapt framework, a parallel and proactive adaptation framework that supports parallel identification and prediction of the need for adaptation and reconfiguration of the service compositions during their execution time. The framework supports the identification of a problem in an instance of a composition and the impact of this problem in other instances of the same composition or in different compositions that share common operations when the identified problem is in any of these operations. When a problem is identified in an instance of a composition, other affected parts of the composition are proactively identified, in order to rectify the various composition instances. A prototype tool has been implemented, and the approach has been evaluated in several scenarios. The results of the evaluation demonstrate that the use of a proactive approach combined with a parallel approach outperforms the use of only a proactive approach in terms of the number of composition instances that are successfully adapted.

Currently, we are extending the framework to support service compositions that provide interactions with users and how the proactive and parallel adaptation can deal with these interactions and delays that may be caused by them. We are also investigating the use of the congestion control algorithm used in the TCP protocol to dynamically adjust the expected throughput of service operations. Another future work consists of considering different types of constraints when attempting to adapt a service composition (e.g. stateful services and operations that need to be used by certain service providers).

References

1. Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., Plebani, P.: PAWS: A framework for executing adaptive web-service processes. IEEE Softw. **24**(6), 39–46 (2007)
2. Aschoff, R., Zisman, A.: QoS-driven proactive adaptation of service composition. In: ICSOC'11, pp. 421–435 (2011)
3. Aschoff, R., Zisman, A.: Proactive adaptation of service composition. In: SEAMS'12, pp. 1–10 (2012)

4. Baresi, L., Di Nitto, E., Ghezzi, C., Guinea, S.: A framework for the deployment of adaptable web service compositions. SOCA **1**(1), 75–91 (2007)
5. Dai, Y., Yang, L., Zhang, B.: QoS-driven self-healing web service composition based on performance prediction. J. Comput. Sci. Technol. **24**(2), 250–261 (2009)
6. Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K.: A journey to highly dynamic, self-adaptive service-based applications. ASE **15**(3), 313–341 (2008)
7. Dustdar, S., Papazoglou, M.P.: Services and service composition – an introduction (services und service komposition – eine einführung). Inf. Technol. **50**(2), 86–92 (2009)
8. Guinea, S., Kecskemeti, G., Marconi, A., Wetzstein, B.: Multi-layered monitoring and adaptation. In: ICSOC'11 (2011). https://doi.org/10.1007/978-3-642-25535-9_24
9. Kazhamiakin, R., Wetzstein, B., Karastoyanova, D., Pistore, M., Leymann, F.: Adaptation of service-based applications based on process quality factor analysis. In: LNCS'09 (2009)
10. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: Monitoring, prediction and prevention of SLA violations in composite services. In: ICWS'10 (2010)
11. Metzger, A., Sammodi, O., Pohl, K., Rzepka, M.: Towards pro-active adaptation with confidence: augmenting service monitoring with online testing. In: SEAMS'10 (2010). http://doi.acm.org/10.1145/1808984.1808987
12. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for WS-BPEL. In: WWW'08 (2008). https://doi.org/10.1145/1367497.1367607
13. Natrella, M.: e-Handbook of Statistical Methods. Nist/Sematech (2010). http://www.itl.nist.gov/div898/handbook/
14. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: a research roadmap. Int. J. Coop. Inf. Syst. **17**(2), 223–255 (2008)
15. Pernici, B.: Self-healing systems and web services: the WS-DIAMOND approach. In: LNBIP'09 (2009)
16. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated composition of web services by planning at the knowledge level. In: IJCAI'05 (2005)
17. Popescu, R., Staikopoulos, A., Liu, P., Brogi, A., Clarke, S.: Taxonomy-driven adaptation of multi-layer applications using templates. In: SASO'10 (2010). https://doi.org/10.1109/SASO.2010.23
18. Saboohi, H., Amini, A., Herawan, T., Kareem, S.: Failure recovery of composite semantic services using expiration times. In: Herawan, T., Deris, M.M., Abawajy, J. (eds.) Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013), Lecture Notes in Electrical Engineering, vol. 285, pp. 683–690. Springer, Singapore (2014). https://doi.org/10.1007/978-981-4585-18-7_77
19. Tosi, D., Denaro, G., Pezze, M.: Towards autonomic service-oriented applications. Int. J. Autom. Comput. **1**, 58–80 (2009). https://doi.org/10.1504/IJAC.2009.024500
20. Web Services Business Process Execution Language (WS-BPEL) Version 2.0.: Organization for the Advancement of Structured Information Standards (OASIS) (2007). http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
21. Zengin, A., Kazhamiakin, R., Pistore, M.: Clam: cross-layer management of adaptation decisions for service-based applications. In: ICWS'11 (2011). https://doi.org/10.1109/ICWS.2011.76
22. Zisman, A., Spanoudakis, G., Dooley, J., Siveroni, I.: Proactive and reactive runtime service discovery: A framework and its evaluation. IEEE Trans. Softw. Eng. **39**(7), 954–974 (2013)