QoS-driven Proactive Adaptation of Service Composition

Rafael Aschoff[1] and Andrea Zisman[1]

[1] Department of Computing, City University London,
London, EC1V 0HB, United Kingdom
{abdy961, a.zisman}@soi.city.ac.uk

Abstract. Proactive adaptation of service composition has been recognized as a major research challenge for service-based systems. In this paper we describe an approach for proactive adaptation of service composition due to changes in service operation response time; or unavailability of operations, services, and providers. The approach is based on exponentially weighted moving average (EWMA) for modelling service operation response time. The prediction of problems and the need for adaptation consider a group of services in a composition flow, instead of isolated services. The decision of the service operations to be used to replace existing operations in a composition takes into account response time and cost values. A prototype tool has been implemented to illustrate and evaluate the approach. The paper also describes the results of a set of experiments that we have conducted to evaluate the work.

Keywords: Proactive adaptation, response time, cost, spatial correlation.

1 Introduction

A major research challenge for service-based systems is the support for service compositions that need to adapt autonomously and automatically to new situations [9][10][24][28]. Some approaches for adaptation of service compositions have been proposed in [1][2][16][29]. However, these approaches support adaptation of service compositions in a reactive way, which is time-consuming and may lead to several unwanted consequences (e.g. user and business dissatisfaction, loss of money, loss of market opportunities). Therefore, it is important to provide approaches that consider adaptation of service composition in a proactive way, predicting problems in a composition before they occur. Some initial works for proactive adaptation of service composition have been proposed in [8][15][20][30]. Overall, these few approaches are fragmented, limited, and in their initial stages.

We define *proactive adaptation of service composition* as the detection of 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 the violation of the service level agreement (SLA) for the whole composition, 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 provider P to be replaced, before reaching the execution part in the composition where services from P are invoked.

Proactive adaptation of service composition includes four main steps, namely (i) prediction of problems, (ii) analysis of the problems triggered by prediction, (iii) decision of actions to be taken due to the problems, and (iv) execution of the actions. As defined in [26], the prediction of problems is concerned with the identification of the occurrence of a problem in the near future based on an assessment of the current state of the system. More specifically, in the scope of service-based systems, problem prediction is concerned with the assessment of what is the impact of a service misbehaviour, or of a group of services, in other parts of the service composition.

In this paper we describe *ProAdapt*, a framework for proactive adaptation of service composition due to changes in service operation response time; or unavailability of operations, services, or providers. ProAdapt provides adaptation of a composition during its execution time and for future executions of the composition. The framework is based on function approximation and failure spatial correlation techniques [26]. The approach uses exponentially weighted moving average (EWMA) [23] for modelling expected service operation response time, and monitors operation requests and responses to identify the availability of services and their providers.

In ProAdapt, 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 framework also identifies other operations that may be affected in a composition flow due to problems caused by a specific one. For example, when the observed response time of an operation is greater than its expected response time, the approach verifies the implication of this response time discrepancy in the service composition, instead of triggering immediate replacement of the operation. This verification considers service level agreements (SLAs) specified for the whole composition and (variable) observed values of quality aspects of the operations in the composition. When it is necessary to replace an operation, or a group of operations, the candidate operations to be used in the composition are selected based on both response time and cost constraints. This is because, in practice, there is a strong correlation between the response time and the cost for an operation.

The remainder of this paper is structured as follows. In Section 2 we present ProAdapt framework. In Section 3 we describe the proactive adaptation process for predicting and analysing problems in service composition, and for deciding and executing the adaptation actions. In Section 4 we discuss implementation and evaluation aspects of our work. In Section 5 we give an account of related work. Finally, in Section 6 we discuss concluding remarks and future work.

2 Proactive Adaptation Framework

Fig. 1 shows the overall architecture of ProAdapt framework with its main components (represented as rectangles), namely: *execution engine*, *specification*

translator, *service discovery*, *monitor*, and *adaptor*. It also shows the different types of data used as input or generated as output by the main components (represented as documents). We describe below each of these main components.

The *execution engine* receives and executes service composition specifications. We assume service composition specifications represented as BPEL4WS [5] due to its wide use and acceptance. The service composition specifications provide the flow of the application and do not have information of the exact services that need to be invoked in a composition. Instead, it contains abstract partner links information. The exact services will be instantiated by the adaptor component.

The *specification* translator is responsible to parse a service composition specification in BPEL4WS and the service level agreement (SLA) for the composition and to create a *composition model template* that will be used to generate *execution models* of the composition. An example of an execution model is described below.

The *service discovery* 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 [27][31] that has been developed by one of the authors of this paper to assist with the identification of candidate operations. This approach advocates a proactive selection of service operations that match functional, behavioural, quality, and contextual aspects. Details of this approach are out of the scope of this paper. The identified operations are used to create and adapt execution models by the adaptor component.

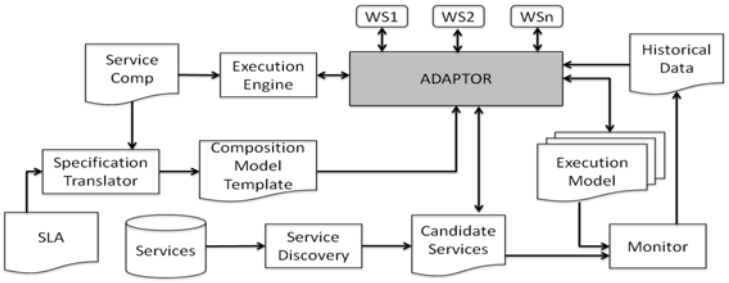


Fig. 1. ProAdapt architecture overview

The *monitor* verifies the QoS aspects of the operations used in the instantiated execution models and the replacement candidate operations, and provides historical data of these QoS aspects. The adaptor uses the historical data to predict and analyse the need for adaptation. The current implementation of the framework uses a simple monitor that we have developed that intercepts calls to the services, calculates the response time that it takes from the invocation of an operation and the receipt of its results, and accumulates the calculated response times as historical data.

The *adaptor* is the main component of our framework. It (a) receives calls from the execution engine to invoke operations in the composition and provides the results to the execution engine; (b) instantiates composition model templates and generates the execution models with real endpoint service operations to be invoked in the composition and other information; (c) predicts and analysis problems that may exist in a composition; and (d) decides on and executes actions to be taken.

Execution Model. We advocate the use of an *execution model* for each execution of a service composition. This is necessary to provide information to support prediction and analysis for adaptation, given the lack of such information in BPEL4WS [5] specifications. An execution model is a graph representation of the service composition specification with information about the (i) execution flow, (ii) deployed endpoint service operations, (iii) state of a service operation in a composition (e.g., completed, 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.

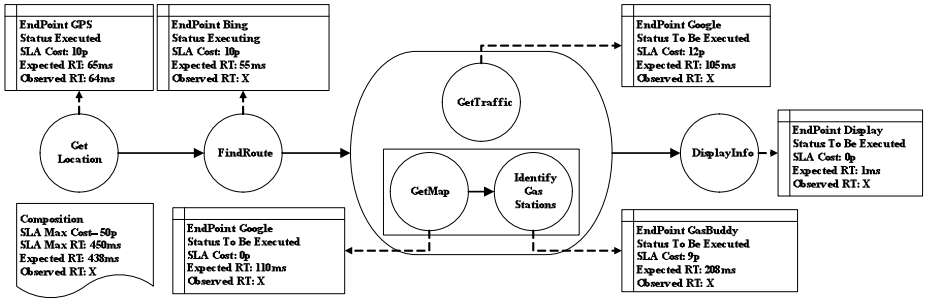


Fig. 2. Example of composition execution model

Fig. 2 shows an example of an execution model for a *RoutePlanner* service composition scenario to assist users to request information from a PDA about optimal routes to be taken when driving. As shown in the figure, the composition offers services to support the identification of a driver's current location, identification of an optional route for a certain location, display of maps of the area and route to be taken, provision of traffic information throughout the route to be taken, computation of new routes at regular intervals due to changes in traffic, and provision of information about near gas stations. In Fig. 2, for each service operation, we show its deployed endpoint, status, SLA cost values, and expected and observed response time (RT) values. We also show these values for the whole composition.

3 Proactive Adaptation Process

In ProAdapt, the adaptation process may be triggered due to (i) changes in the response times values of operations in a service composition that affects SLA values of a composition, (ii) unavailability of operations in a composition, (iii) unavailability of services, or (iv) unavailability of providers. When one of cases (i) to (iv) occurs, the adaptor verifies if the composition needs to be changed and modifies the composition during its execution time, if necessary. More specifically, the changes are performed in the execution models by trying to identify operations that can replace existing operations participating in the remaining parts of the execution model such that the aggregation of the response time and cost values of these replacement operations, together with the response time and cost values of the operations that have already been executed in the composition and the ones that are still to be executed, comply with the SLA values of response time and cost for the composition.

Response Time Modeling. In order to guarantee compliance of the SLA response time and cost values in an execution model (EM), it is necessary to consider the aggregation of the response time values of the participating operations; and the time for the adaptor to identify and analyse problems and perform changes in the execution model when necessary, as specified in the function below.

$$T(EM) = Aggreg(RT(Set(O))) + T(Adapt), \quad \text{where:} \tag{1}$$

- T(EM) is the time to complete the execution model EM;
- RT(Set(O)) is the response time of the operations in EM;
- Aggreg() is a function that returns the aggregated values of the response time of the operations depending on the execution logic of the model;
- T(Adapt) is the time required by the adaptor.

The aggregation of the response time values of the service operations in the execution model considers different execution logics in a model such as sequence, parallel, conditional selection, and repeat logics. In the example in Fig. 2, sequence execution logics are composed of operations (a) *GetLocation* and *FindRoute*, and (b) *GetMap* and *IdentifyGasStation*; while a parallel execution logic is found in operations *GetMap* and *IdentifyGasStation* with operation *GetTraffic*. In the case of sequence execution logic, the aggregated response time is calculated as the sum of the response times of the operations in the sequence; in the case of parallel execution logic, the aggregated response time is calculated as the maximum of the response time values for the operations in the parallel execution logic.

The aggregated response times of the operations in an execution model and candidate replacement operations is calculated based on *expected* response time values of the operations not yet executed and the *observed* response time values of the operations already executed. As outlined in [8], the response time for an operation request combines the time for executing the operation and the network time for sending the operation request and receiving its response. We observed that there are also other times that should be considered such as the time of marshaling/unmarshaling a request, and the time that a request may need to stay in a queue in both client and server sides. We define the response time of an operation as:

$$RT(O) = PT(O) + DT(O), \quad \text{where:} \tag{2}$$

- PT(0) is the processing time for an operation O, which is given by service providers.
- DT(0) is the variable delay time associated with an operation O, including the network, queue, and marshaling/unmarshaling times.

As shown above, the response time is considered a variable parameter that can be affected due to changes in the network and system resources. Therefore, in order to identify *expected* response time values, it is necessary to use techniques that predict the behaviour of random parameters. ProAdapt uses exponentially weighted moving average (EWMA) [23] technique for this prediction due to its simplicity.

The expected response time value of an operation changes with time. At time t0, an operation expected response time value is its processing time. At time ti (i>0), the expected response time is given by the EWMA function [23] below:

$$Ev(O(ti)) = Obv(O(ti\text{-}x))(1\text{-}\alpha) + Ev(O(ti\text{-}x))^{*}\alpha + \beta^{*}SD, \quad \text{where} \tag{3}$$

- Ev(O(ti)) is the expected response time value of operation O at time ti of execution;
- Obv(O(ti-x)) is the last observed response time value of O at time ti-x of execution (0<x<i; and ti-x<ti);
- Ev(O(ti-x)) is the expected response time value of O at time ti-x of execution (0<x<i; and ti-x<ti);
- α is a weight given for the past expected response time value;
- β*SD is a threshold calculated based on the standard deviation (SD) of previous observed response time values of O and β is as a constant parameter.

For each operation in an execution model, a set of candidate replacement operations is identified by the service discovery tool (see Fig. 1), based on functional and behavioural matching. The identified candidate operations are ordered by the weighted sum of the normalised response time and cost values of an operation, as per the function below. The weights are used to specify priorities in QoS values.

$$V(O) = wRT * Norm(Ev(O)) + wC * Norm(C(O)), \quad \text{where:} \tag{4}$$

- O is a candidate service operation;
- Norm(Ev(O)) is the normalised value for the expected response time of O;
- Norm(C(O)) is the normalised value for the cost of operation O;
- wRt and wC are weights used for response time and cost, with wRt+wC = 1.

Execution of Adaptation Process. For each user Ui of a service composition, an execution model EM_{Ui} is created. The initial endpoint operations used in an execution model is identified by the adaptor based on available operations, current expected response times for these operations, and SLA values of the composition.

During the execution of EM_{Ui}, the model is updated in several ways by: (a) changing the status of its operations (e.g., from "to be executed" to "executing" and "executed"), (b) calculating the observed and expected response time values of the operations as per the functions defined above, and (c) changing operations in not yet executed paths in the model, if necessary. For any of cases (i) to (iv) that may trigger the need for adaptation, the process tries to identify other parts in the execution model that may be affected by a problem. The process is based on spatial correlations of operations, services, and providers (dependencies that may exist between these elements). For example, when a service S becomes unavailable, the process considers all other operations of S in the model since these operations may not be able to be executed. Similarly, when a provider P is unavailable, the process considers all services and operations in the model from P; and when an operation O becomes unavailable, the process considers future invocations of O in the model.

We describe below the process for each case (i) to (iv). Suppose O an operation in the model being invoked by the adaptor (status "executing"); S the service associated with O; P the provider of S; Obv(O) the observed response time for O; and Ev(O) the last expected response time for O calculated using function (3).

Case (i): *Changes in the response time of O*

In this case, if the Obv(O)<=Ev(O), there is no need for adaptation and the model continues its execution. Otherwise, (Obv(O)>Ev(O)), the process verifies if the model's SLA response time is affected. This analysis is done by using function (1) above. If the SLA value is maintained, the process continues its execution. If not, the process considers operations in the model that have not yet been executed and tries to

find possible combinations of replacement operations that provide the functionality of those operations, and maintain the SLA response time and cost values. The operations in the model are considered inside the smaller possible execution logic. If a combination cannot be found, the process identifies the best possible combination.

Case (ii): *O is unavailable*
In this case, S has been changed and the adaptor receives a message from S about the unavailability of O. The process tries to identify another operation O' in the set of candidate replacement operations with the same functionality of O and acceptable expected response time and cost values. If O' exists, O is replaced by O'. The process also identifies other parts in the model not yet executed that use O and possible replacement operations for O in these parts. If these replacements operations are identified, they are use to replace O in those parts. The replacement operations are identified as in case(i) above. O is also removed from the set of candidate replacement operations.

Case (iii): *S is unavailable*
In this case, P informs the adaptor that S is not available. The process detects all other operations in the model associated with S that have not yet been executed, if any; identifies replacement operations for them from the set of candidate replacement operations; changes the execution model by replacing the operations; and removes the operations of S from the set of candidate replacement operations.

Case (iv): *P is unavailable*
In this case, the component receives a "*connection exception*" message indicating that the provider is unavailable. Similar to case (iii), the process detects all operations in the execution model associated with P that have not yet been executed; identifies replacement operations from the set of candidate replacement operations; changes the execution model by replacing the operations; and removes the operations of P from the set of candidate replacement operations.

For cases (ii) to (iv) above, if the identified replacement operations match the functionality of the operations to be replaced, but do not maintain the SLA values of the composition, the process identifies the best possible combination of replacement operations to be used. When there are no replacement operations that match the functionality, the execution model cannot be adapted and the process terminates.

In order to illustrate consider case (i) above. Suppose the observed value of *FindRoute* operation in Fig. 2 as 68ms. In this case, the response time of the execution model will be 451ms (see function 1), which is higher than the composition's SLA time value (450ms). The process tries to identify replacement operations in the following parallel execution logic (*GetTraffic* and *GetMap*, and *IdentifyGasStation* operations) that can guarantee the SLA value. If no solution is available for the operations in the parallel execution logic, the process includes operation *DisplayInfo* and considers the combined execution logics for analysis. Suppose that a replacement operation for *GetMap* is identified with expected response time value of 90 ms, which is used to compensate for the high response time of *FindRoute*.

4 Implementation Aspects and Evaluation

A prototype tool of the framework has been implemented in Java (J2SE). The execution engine and the adaptor components were implemented as a single component for simplicity. The tool assumes service compositions in BPEL4WS[5] exposed as Web Services using SOAP protocol, and participating operations and user requests emulated using soapUI[11]. The service discovery tool was also implemented in Java and is exposed as a web service using Apache Axis2. The external service registry uses eXist database [12]. In order to evaluate ProAdapt we focused on three cases described below.

Case (1): Demonstration that the framework provides time reduction when compared to non-proactive approaches;
Case (2): Demonstration that the framework manages to adapt compositions ensuring the SLA values of the composition, and considering prioritization of QoS values;
Case (3): Analysis of the performance of the framework.

The evaluation was executed in a service composition with 12 operations and different types of execution logics, as shown in Fig. 3. We assumed the same syntactic and semantic characteristics for the 12 operations, with each operation having two input parameters and producing one output result. The size of each message representing an operation request (or an operation response) was around 60 bytes. Each operation has different associated costs and processing time values, as summarised in Table 1. We assumed SLA values for cost as 2800 pence and response time as 3.5 seconds for the whole composition; and the weights for the cost and response times as 0.9 and 0.1, respectively.

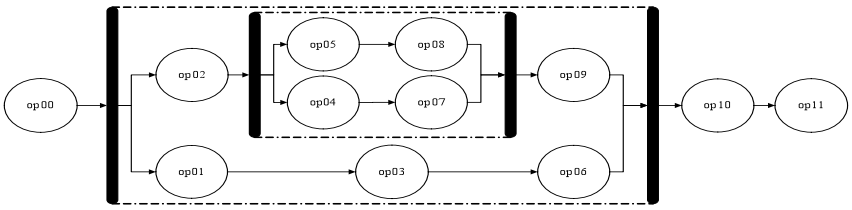


Fig. 3. Experiment service composition

We used an environment with five different machines, namely: (a) *client machine* responsible to create simultaneous requests to the service composition, simulating several concurrent users; (b) *adaptor engine machine* connected to three service providers; and (c) one machine for each *service provider* P1, P2, P3. Table 1 presents a summary of the specification of each machine and the speed of the network links between the machines. We used different speeds for the network links to emulate bottleneck situations that may occur when using an Internet environment. Each service provider contains four different services, with each service implementing three different operations in the composition in Fig. 3. The operations in the four services in provider P1 are similar in terms of their functionalities to the ones in providers P2 and P3, in order to simulate possible candidate replacement operations. We assumed different costs and processing time values for the operations in the various providers as summarised in Table 1.

We appreciate that in a real scenario the operations used in a composition may be from different providers. In the experiment, for the initial execution model we assumed all operations from the same provider (P1) to enforce a more realistic bottleneck situation. This does not invalidate our experiments since it is important to consider the network capacity between the adaptor and providers.

For the EWMA function (see Section 3), we used a threshold of 1.5 and the weight for past expected response time values of 0.6. These values were identified after executing the operations in the composition in Fig. 3 several times, and verifying that with these values the expected response times of the operations were below their observed times for 95% of the cases. We describe below how we conducted the experiments and their results for each of cases (1) to (3) above.

Table 1. Configuration of experiment environment

Machine	Configuration	Services/ Operations	Cost (pence)	Processing Time (ms)	Network Links (Mb/s)
Client (C)	Turion 1GHz 2GB RAM				C-A = 3.0
Adaptor (A)	Core 2.33 GHz 3GB RAM				
Provider P1	Pentium 4.3 GHz 1GB RAM	S0:O00, O04, O08 S1:O01, O05, O09 S2:O02, O06, O010 S3:O03, O07, O011	100	150	A-P1 = 1.0
Provider P2	Core 1.86 GHz 2GB RAM	S0':O00, O04, O08 S1':O01, O05, O09 S2':O02, O06, O010 S3':O03, O07, O011	150	100	A-P2 = 1.5
Provider P3	Pentium 3.0 GHz 3GB RAM	S0":O00, O04, O08 S1":O01, O05, O09 S2":O02,O06, O010 S3":O03,O07, O011	300	50	A-P3 = 3.0

Case (1): In this case, we compared the execution times of the composition in Fig. 3 when there is no need for adaptation with the execution times when there are problems and adaptation is required. More specifically, we analysed the times for the situations in which the need to execute adaptation is triggered by problems with (a) operations, (b) services, and (c) providers, as described in Section 3. In all these situations, we assumed the processing times of the operations in providers P2 and P3 as the same as P1. This is necessary to create a homogeneous environment and avoid using a replacement operation with faster processing time and, therefore, diminishing the impact of the time wasted when trying to invoke an unavailable operation.

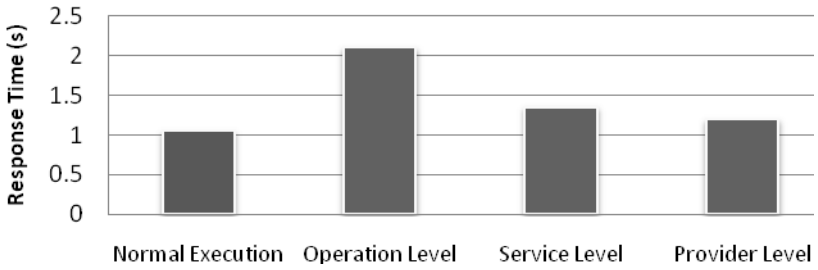


Fig. 4. Impact of spatial correlation on composition response times

For problems at the operation level (case (a)), we assumed no spatial correlation and analysed the time to execute the composition when replacement operations need to be identified for all the 12 operations in the composition. For problems at the service level (case (b)), we considered the spatial correlation between operations from the same service, and analysed the time to execute the composition when each of the four services have a problem and adaptation is required. For problems at the provider level (case (c)), we considered spatial correlation between services, and analysed the time to execute the composition when a new provider needs to be identified.

Fig. 4 presents the results of this experiment. As shown in the figure, for case (a) the time to execute the composition and change all operations without considering any spatial correlation is two times more than the time to execute the composition without any need for adaptation (normal execution). The results also show that for case (b), the time to execute the composition is improved by 36% when compared to case (a), and that for case (c) this time is improved even further by 43% when compared to case (a). We verify an improvement of using our proactive adaptation approach when compared to the situation in which a non-proactive approach is used.

Case (2): In this case, we simulated the client machine to support an increase in the number of users invoking the composition in an incremental way. More specifically, we analysed the framework in 30 intervals of ten seconds each (total of 300 seconds) with a rate of one user per second in the first interval of ten seconds, two users per second in the second interval, and an increment of one extra user every ten seconds reaching 30 users per second in the last interval. It is worth noting that for each user request performed by our client machine, 12 operations are executed and, therefore, the number of simultaneous operation executions in the experiment is greater than the number of user requests in the various intervals. Moreover, at a certain time t in the experiment, the number of users invoking the composition and consuming resources is greater than the number of new users at t, since the time to execute the composition without any problem is more than 1 second (see Fig. 4).

The above simulation was used in order to provide an environment that could on its own create problems to the composition in terms of response time and cost values due to the number of users and network resources being consumed and, therefore, allow the verification of the behaviour of ProAdapt in cases of problems.

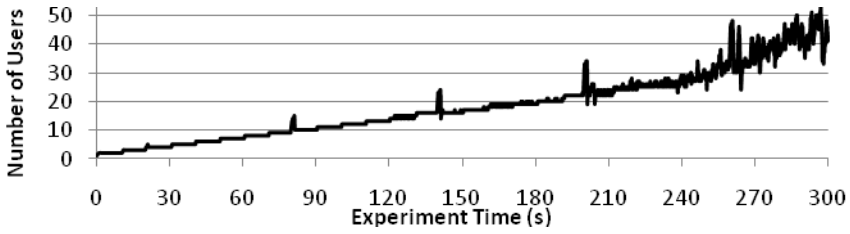


Fig. 5. Number of simultaneous users consuming resources during the experiment

Fig. 5 shows the number of concurrent users consuming resources during the different times of the experiment. As expected, the accumulated number of users is greater than the rate of new users invoking the composition. The graph in Fig. 5 shows a larger number of accumulated users towards the end of the experiment since

during this time there are a larger number of invocations for the operations in the compositions causing degradation in the response times of the operations and, therefore, delaying the execution of the compositions.

Fig. 6 shows the time to execute each execution of the composition (represented as squares) during the whole experiment, and the compositions that were able to adapt themselves and finish within the SLA response time values (dotted line in the graph). As shown, the majority of the executions managed to adapt themselves and finish within the SLA response time value. The graph also shows a stable response time for the executions in the beginning of the experiment and oscillations in the response times starting at 200 seconds of the experiment. This is because between 20 and 25 service composition requests per second provider P1 reaches its full capacity, causing degradation in the operations response times, and eventually, the need to adapt the composition executions so that the SLA values are maintained.

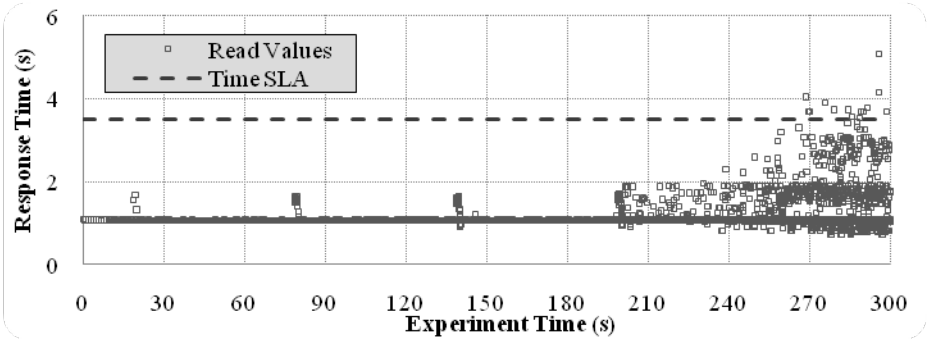


Fig. 6. Variation of composition response times during the experiment

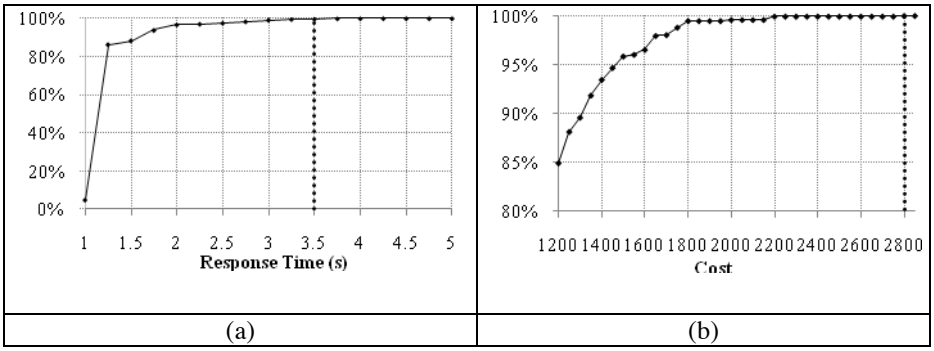
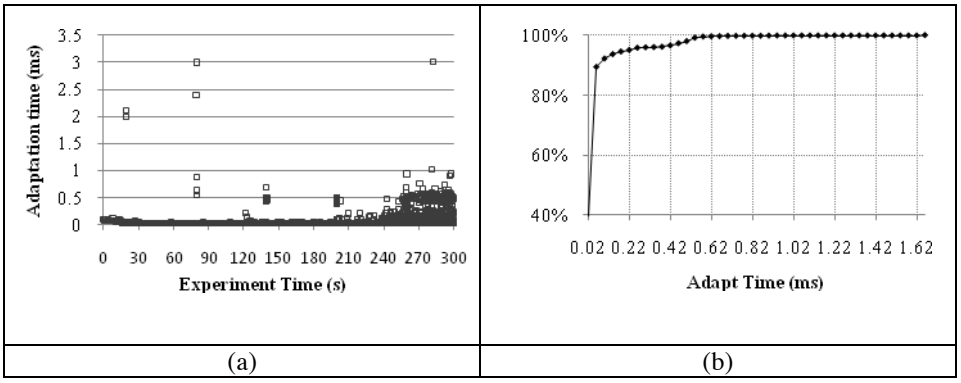


Fig. 7. Cumulative frequency distribution of (a) response times and (b) costs

The cases in which the executions did not finish before the SLA value (cases above dotted line in Fig. 6), were due to bottlenecks in the providers and lack of available operations that could be executed faster and with the cost values specified in the experiment. In order to verify the impact of these cases, we present the cumulative frequency distribution for the response times of the executions in Fig. 7(a). As shown, the response times of the executions where below 1.5 seconds for 80.65% of the cases, and in 99.69% of the cases the executions respected the SLA response time

value. Therefore, from a total of 4650 user requests performed during 300 seconds of experiments, only 14 user requests could not be executed for the given SLA value.

Similarly, Fig. 7(b) presents the cumulative frequency distribution for the costs of the executions. As shown, the cheapest executions (1200 pennies) occurred in 85% of the cases. The graph also shows that the SLA cost value was respected in all 4650 requests, which was expected since the framework identifies only operations that respect the cost values when adapting the composition.



(a) (b)

Fig. 8. (a) Adaptation time and (b) cumulative adaptation time over the experiment

Case (3): In this case, we analysed the time spent by ProAdapt to adapt the compositions. Fig. 8(a) shows the times of the adaptor component in milliseconds for all executions in our experiment, while Fig. 8(b) shows the cumulative frequency distribution. As shown, in the majority of the cases the overall adaptation time is very small and does not cause a significant increase on the overall response time of the composition execution. The experiment shows that only in few cases the time for the adaption process was 3 milliseconds at most, which is very low when compared with the time to execute the composition when there is no need for adaptation. This is due to the way the process is implemented in which compositions are analysed based on execution logics and without looking for the optimal combination of operations, but for combinations that meet given SLA values.

Overall, the results of our experiments are very positive and demonstrate that the framework can support proactive adaptation of service composition during execution time, due to different QoS characteristics. The experiments also show that the performance of the adaptation process is good and that the process does not cause penalties when changes in the composition are necessary.

5 Related Work

Some approaches have focussed on dynamic service composition, in which services are identified and aggregated during runtime in support of certain functional and quality characteristics of the desired systems [1][3][6][7][13][19][25].

Approaches for reactive adaptation of service composition were proposed in [1][2][16][18][29]. These approaches propose changes in service composition based

on pre-defined policies [2], self-healing of compositions based on detection of exceptions and repair using handlers [29], context-based adaptation of compositions using negotiation and repair actions [1]; and key performance indicator (KPI) analysis and the use of adaptation strategies related to the KPI fulfilment [16].

Exceptions to the reactive approaches are found in the works in [8][14][15][17][20][22][28]. As in the case of ProAdapt, the work in [8] is based on prediction of performance failures to support self-healing of compositions. The work uses semi-Markov models for performance predictions, service reliability model, and minimization in the number of service re-selection in case of changes. The decision to adapt is based on the performance of a single service, while our framework considers a group of related service operations in a composition, avoiding unnecessary changes to the composition. Moreover, the work in [8] does not support unavailability and malfunctioning of operations, services, and providers, as well as spatial correlations between these elements in a composition.

In [17] the PREvent approach is described to support prediction and prevention of SLA violations in service compositions based on event monitoring and machine learning techniques. The prediction of violations is calculated only at defined checkpoints in a composition based on regression classifiers prediction models.

The works in [14][20][28] advocate the use of testing to anticipate problems in service compositions and trigger adaptation requests. The approach in [28] supports identification of nine types of mismatches between services to be used in a composition and their requests based on pre-defined test cases. In [14][20] test cases are created during the deployment of service compositions and used to identify violations after a service is invoked for the first time. However, the creation of test cases is not an easy task and the work does not specify how to generate new test cases for a modified composition.

In [15] the authors describe a two-stage adaptation approach due to dependability requirements of service-oriented systems. The work combines proactive adaptation to support self-protection of the system and reactive adaptation to support self-healing of the system. The paper does not describe the advantages of combining both approaches and lacks details of the proactive approach.

Similar to our framework, some works have been proposed to support prediction of response time on the web [22][30]. In [22] the authors describe a probability function for web-access response time that uses file-size cumulative function and delay probability density function. The work in [30] presents a framework for performance evaluation of web services based on queuing networks and fork-and-join. As in the case of our framework, this work considers the execution of a service and, therefore its response time, from the moment a service request leaves a client machine to the moment service results return to the client. Our framework complements the above works and applies prediction of operation response time.

Similar to our approach, in [4] the authors advocate that the management of service compositions during runtime needs to consider the structure of a composition and the dependencies between the participating services, and propose an approach that determines the impact of each service in a composition on its overall performance. The reactive region-based reconfiguration approach presented in [18] also has similarities with our work since it considers services that are in certain regions in a

composition. However, in [18] the reconfiguration approach is used only for future executions of the composition, instead of current executions as in our work.

Our framework complements existing works for service composition adaptation and contributes to the challenge of supporting adaptation in a proactive way during execution time, taking into consideration SLA values for the whole composition.

6 Conclusion and Final Remarks

In this paper we presented ProAdapt, a framework for proactive adaptation of service composition due to changes in service operation response times; or unavailability of operations, services and providers. The framework uses function approximation and failure spatial correlation of operations, services, and providers to predict problems. It also uses exponentially weighted moving average (EWMA) to model response times of operations. The adaptation process is performed during the execution of a composition and considers a group of operations in a composition to verify if a problem can be compensated by other operations in a composition flow. Replacement operations are selected based on their response times and cost values. A prototype implementation of the framework has been developed and used to evaluate the framework with positive results

Currently, we are extending the framework to support proactive adaptation due to other types of QoS aspects and other circumstances. Examples of these circumstances are availability of new (better) service operations than the ones used in a composition, and changes in requirements or emergence of new requirements for the system. We are also investigating other ways of adapting compositions including changes in the structure of the composition's workflow (e.g., replacement of one service by a group of services, or vice-versa). We are expanding our prototype tool to support analysis of the impact of a problem in conditional and repetition execution logics in service compositions, and integrating the adaptor component with our proactive service discovery framework for identification of candidate replacement operations.

References

1. D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. PAWS: A Framework for Executing Adaptive Web-Service Processes. IEEE Software, 24 (6), 2007.
2. L.Baresi, E. Di Nitto, C. Ghezzi, and S. Guinea. A Framework for the Deployment of Adaptable Web Service Compositions. Service Oriented Computing and Applications Journal (to appear).
3. R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz. Heuristics for QoS-aware Web Service Composition. IEEE International Conference on Web Services, 2006.
4. L. Bodenstaff, A. Wombacher, M. Reichert, M.C. Jaeger. Analyzing Impact Factors on Composite Services, IEEE Int. Conf. on Services Computing, September, 2009
5. BPEL4WS. download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf
6. G. Canfora, M. Di Penta, R. Esposito, M. L. Villani. QoS-Aware Replanning of Composite Web Services, IEEE Int. Conf. on Web Services, 2005

7. M. Colombo, E. Di Nitto, and M. Muri. SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In Proc. of the 4th Int. Conf. on Service Oriented Computing, 2006.
8. Y. Dai, L. Yang, B. Zhang. QoS-Driven Self-Healing Web Service Composition Based on Performance Prediction. Journal of Computer Science and Technology, 24(2), March 2009.
9. E. Di Nitto, C, Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A Journey to Highly Dynamic, Self-Adaptive, Service-based Applications. Automated Software Engineering Journal, 15, 2008 pp. 313-341.
10. S. Dustdar and M.P. Papazoglou. Services and Service Composition – An Introduction. IT Information Technology, 2(2008), pp. 86-92.
11. Eviware. soapUI; the Web Services Testing tool. www.soapui.org.
12. eXist. http://exist.sourceforge.net.
13. K. Fujii and T. Suda. Semantics-based Dynamic Web Service Composition. Int. Journal of Cooperative Inf. Systems, 15(3), pp293-324, 2006.
14. J. Hielscher, R. Kazhamiakin, A. Metzger, M. Pistore. A Framework for Proactive Self-Adaptation of Service-based Applications Based on Online Testing. 1st European Conference Towards a Service-Based Internet, LNCS vol. 5377, Spain, 2008.
15. N. Jun, Z. Bin, Z. Xiamgyu, Z. Zhiliang, L. Dancheng. Two-Stage Adaptation for Dependable Service-Oriented System. International Conference on Service Sciences, 2010.
16. R. Kazhamiakin, B. Wetztein, D. Karastoyanova, M. Pistore, and F. Leymann. Adaptation of Service-based Applications Based on Process Quality Factor Analysis. ICSOC/ServiceWave 2009.
17. P. Leitner, A. Michlmayr, F. Rosenber, and S. Dustdar. Monitoring, Prediction and Prevention of SLA Violations in Composite Services. Int. Conf. on Web Services, 2010.
18. K.J Lin, J. Zhang, Y. Zhai, and B. Xu. The Design and Implementation of Service Process Reconfiguration with End-to-end QoS Constraints in SOA. Journal of Service Oriented Computing and Applications, vol 4, 2010.
19. MAIS Project. Mobile Information Systems – Infrastructure and Design for Flexibility and Adaptability, B. Pernici, ed. Springer, 2006.
20. A. Metzer, O. Sammodi, K. Pohl, M. Rzepka. Towards Pro-active Adaptation with Confidence Augumenting Service Monitoring with Online Testing, Software Engineering for Adaptive and Self-managing Systems, SEMAS, South Africa, May 2010.
21. T.M. Mitchell. Machine Learning, McGraw-Hill International Editions, 1997.
22. M. Miyagi, K. Ohkubo, M. Kataoka, and S. Yoshizawa. Performance Prediction Method for Web-Access response Time Distribution Using Formula, Network Operations and Management Symposium, 2004.
23. NIST/SEMATECH eHandbook of Statistical Methods, www.itl.nist.gov/div898/handbook
24. M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leyman, and B. Kramer. Service-Oriented Computing Research Roadmap. http://tinyurl.com/6jhvd44.
25. M. Pistore, A. Marconi, P. Bertolini, and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level, Int'l Joint Conf. Artificial Intelligence, 2005.
26. F. Salfner, M. Lenk, and M. Malek. A Survey of Online Failure Prediction Methods. ACM Computing Surveys, 42(3), 2010.
27. G. Spanoudakis and A. Zisman. Discovering Services during Service-based System Design using UML. IEEE Transactions of Software Engineering, 36(3): 371-389, 2010.
28. D. Tosi and G. Denaro and M. Pezzè. Towards Autonomic Service-Oriented Applications. International Journal of Autonomic Computing (IJAC), 2009, pp. 58—80
29. WSDiamond.http://wsdiamond.di.unito.it/status.html.
30. S. Youcef, M.U. Bhatti, L. Mokdad, V. Monfort. Simulation-based Response-time Analysis of Composite Web Services, 10th IEEE International Multitopic Conference.
31. A. Zisman, J. Dooley, G. Spanoudakis. A Framework for Dynamic Service Discovery, Int. Conf. on Automated Software Engineering, Italy, 2008.