Proactive Adaptation of Service Composition

Rafael R. Aschoff
Department of Computing
City University London
London. UK
abdy961@soi.city.ac.uk

Andrea Zisman
Department of Computing
City University London
London. UK
a.zisman@soi.city.ac.uk

***Abstract*—Adaptation of service compositions is considered a major research challenge for service-based systems. In this paper we describe a proactive approach to support adaptation of service compositions triggered by different types of problems. The approach allows for changes in the composition workflow by replacing a service operation, or a group of operations, by another service operation or a group of dynamically composed operations. The adaptation process is supported by the use of QoS prediction techniques, analysis of dependencies between service operations, and use of groups of service operations in a composition flow instead of isolated operations. A prototype tool has been implemented to illustrate and evaluate the framework. We also present results of experiments that we have conducted to evaluate the work.**

Keywords-component; service composition adaptation; signature dependency; spatial correlation; SLA and QoS values.

I. Introduction

The complexity of current computer-based applications, the rapid changes in market conditions and regulations, the dynamic creation of business alliances and partnerships, and the need to support the changing demands of users, require software systems to be more flexible, adaptable, and versatile. This is also the case when dealing with *service compositions* in which loosely coupled autonomous computer-based entities owned by third parties known as services are combined to realize applications and create dynamic business processes.

Autonomous and automatic adaptation of service composition has been recognized as an important research challenge [14][15][26][32]. There are several situations that require service compositions to be adapted, including: (i) changes in or emergence of requirements, (ii) changes in the context of the composition and participating services, (iii) changes in functional and quality aspects of services in compositions, (iv) failures in services in compositions, and (v) emergence of new services.

Recently, a few approaches to autonomic service composition have started to appear. These approaches can be classified as reactive [3][5][18][34] and proactive approaches [13][20][22][35]. In the reactive approaches adaptation of service compositions takes place after there are changes in the requirements, context, or services participating in a service-based application. Reactive adaptation of service composition is time-consuming and may lead to several unwanted consequences (e.g. user and business dissatisfaction, loss of money, loss of market opportunities). To alleviate these problems, the proactive approaches for service composition attempt to predict problems in a composition before they occur. Overall, the existing proactive approaches are fragmented and in their initial stages of development.

In order to complement existing works, in this paper, we describe *ProAdapt*, a framework to support proactive adaptation of service compositions during their execution time, as well as for future executions of the compositions. 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 a service provider P in a composition C is unavailable, requiring other services from provider P used in composition C to be replaced, before reaching the execution part in the composition where services from P are invoked; or the identification that a candidate replacement service operation may not provide outputs that are required by another service operation in the composition, calling for this other operation to be replaced.

The framework in this paper extends our initial work in [4] to support proactive adaptation of service composition due to changes in service operation response time or unavailability of operations, services, or providers. The work in [4] is limited in two ways. First, it only supports unavailability and response time as triggers for adaptation. Second, the approach only considers candidate replacement service operations with complete functional and structural matches with the service operations to be substituted. The framework in this paper extends the work in [4] with the following contributions:

- It supports adaptation of service composition triggered by four different classes of situations, namely *C1: problems that cause the composition to stop its execution* (e.g., unavailability or malfunctioning of a deployed service operation); *C2: problems that allow the composition to continue to be executed, but not necessarily in its best way* (e.g., the network link is congested, causing the response times of some operations to be delayed; 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 faster); and

C4: emergence of better services (e.g., a cheaper and faster service becomes available). In this paper, for cases C3 and C4, we concentrate on QoS aspects instead of general new functionalities of a service.

- It allows changes in service compositions performed by (a) replacing a single service operation in the composition by another service operation or by a group of dynamically composed service operations (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).
- It considers dependencies that may exist between the signature of service operations within a composition in order to avoid adaptation of the composition that may lead to mismatches of the signatures of the operations.

The remaining of this paper is structured as follows. In Section II we present an overview of the *ProAdapt* framework. In Section III we describe the adaptation process, including detection and analysis of the situations that trigger the need for adaptation, and decision and execution of actions to be taken when there is a need for changing the compositions. In Section IV we discuss implementation and evaluation aspects of our work. In Section V we provide an account of related work. Finally, in Section VI we discuss concluding remarks and future work.

II. PROACTIVE ADAPTATION FRAMEWORK OVERVIEW

In the case of proactive approaches, problems that trigger the need for adaptation should be predicted before they occur. As defined in [34], the prediction of problems is concerned with the identification of the occurrence of a problem in the near future based on the assessment of the current state of the system. In the scope of service compositions, the prediction of problems considers the impact of a new requirement, a service or group of services misbehaviour, or the existence of better services.

Proactive adaptation of service composition includes four main steps, namely (i) identification and 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. In order to support these steps, we have implemented *ProAdapt* framework with five main components, namely: (i) *composer*, (ii) *execution engine*, (iii) *adaptor*, (iv) *service discovery*, and (v) *monitor*. An overview of the architecture of the framework is shown in Figure 1.

The *composer* is responsible to parse business processes (service compositions) described in different languages (e.g.; BPEL4WS [9]), and their associated Service Level Agreements (SLAs), in order to generate an internal and language-free representation of the business process called *composition template*. The work in this paper concentrates in service compositions specified in BPEL4WS due to its wide adoption. The current approach is limited to compositions that use stateless services; i.e.; compositions with service operations that can be replaced at any point during its execution.

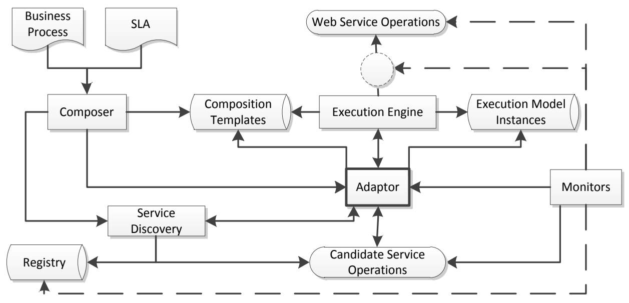


Figure 1: Architecture overview of *ProAdapt* framework.

Figure 2 presents an example of a business process model using the BPMN notation for ordering goods online. In Figure 2, the functionalities of the business process are represented as activities. As shown in the example, when a customer makes an order, the system checks the stock and in case of availability of the required goods, the system displays the interface for executing the payment. In the case of a successful process execution, the payment is received and the goods are dispatched to the customer. Otherwise, both the payment and the order are cancelled. At the end of the business process execution, the customer is notified of the status of the order by using a web interface. In the case of a successful execution, the customer receives an invoice by email or in PDF format.

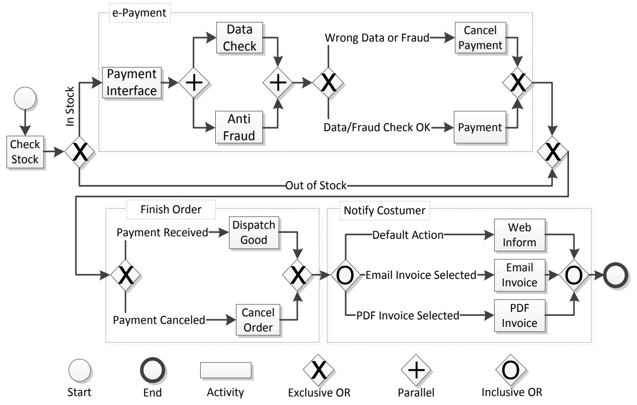
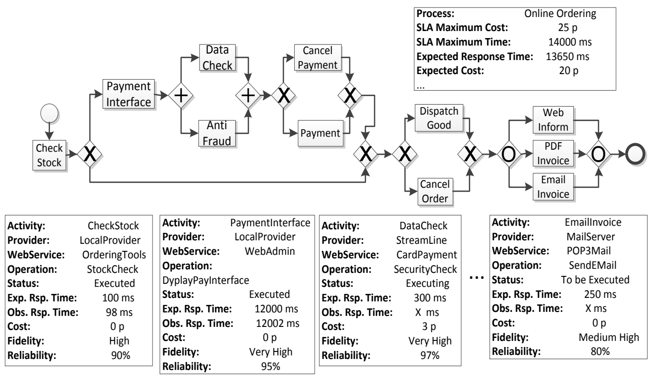


Figure 2: Example of a business process for ordering of goods.

In the approach, instances of a composition template called *execution model instances* are created for each execution of a service composition. The *execution engine* is responsible to create and run execution model instances for each user, and to invoke the service operations in the model instances. It also communicates with the adaptor component to provide information about identified problems when trying to invoke a service operation (e.g., unavailability of a service operation, or time-out of an operation response).

When a business process deployed in the *execution engine* is requested, the associated composition template is used to generate an *execution model instance* of the business process. An *execution model instance* extends a composition template 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.



Process:	Online Ordering
SLA Maximum Cost:	25 p
SLA Maximum Time:	14000 ms
Expected Response Time:	13650 ms
Expected Cost:	20 p
...	

Activity:	CheckStock
Provider:	LocalProvider
WebService:	OrderingTools
Operation:	StockCheck
Status:	Executed
Exp. Rsp. Time:	100 ms
Obs. Rsp. Time:	98 ms
Cost:	0 p
Fidelity:	High
Reliability:	90%

Activity:	PaymentInterface
Provider:	LocalProvider
WebService:	WebAdmin
Operation:	DyplayPayInterface
Status:	Executed
Exp. Rsp. Time:	12000 ms
Obs. Rsp. Time:	12002 ms
Cost:	0 p
Fidelity:	Very High
Reliability:	95%

Activity:	DataCheck
Provider:	StreamLine
WebService:	CardPayment
Operation:	SecurityCheck
Status:	Executing
Exp. Rsp. Time:	300 ms
Obs. Rsp. Time:	X ms
Cost:	3 p
Fidelity:	Very High
Reliability:	97%

...

Activity:	EmailInvoice
Provider:	MailServer
WebService:	POP3Mail
Operation:	SendEMail
Status:	To be Executed
Exp. Rsp. Time:	250 ms
Obs. Rsp. Time:	X ms
Cost:	0 p
Fidelity:	Medium High
Reliability:	80%

Figure 3: Execution model instance for the business process.

Figure 3 shows an example of part of an execution model instance for the business process presented in Figure 2. In Figure 3, we show the deployed endpoint service operations for various activities, the QoS values for the whole execution model instance and for the deployed endpoint service operations, the execution status of the service operations, and some expected and observed QoS values. As shown in Figure 3, the service operations executing the activity of *CheckStock* and *PaymentInterface* are offered by the same services provider *LocalProvider*. The *DataCheck* activity is offered by service provider *StreamLine*. The expected cost and response time values for the whole execution model instance is based on the deployed operation endpoints and their status.

The *monitor* verifies the service operations used in the instantiated execution models and the replacement candidate operations in other to identify any anomalies of types C1 to C4 described in Section I. The current implementation of the framework uses a simple monitor that we have developed that intercepts calls to the services, observes that new services become available, and checks the QoS values of the operations. It also communicates with the adaptor to inform about observed events.

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 [31][36] that has been developed by one of the authors of this paper to assist with the identification of candidate service operations. 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 organised in descending order of best matches. Details of this approach are out of the scope of this paper, but can be found in [36]. The identified candidate operations are used to create and adapt execution models by the adaptor component.

The *adaptor* component (a) receives events from the execution engine and monitor components (situations C1, C2, and C4), and composer component (situation C3); (b) predicts and analyses problems that may exist in the composition based on these events; (c) identifies the need for adaptation of the current execution model instances and/or templates; (d) decides on the actions to be taken; (e) makes necessary changes in the execution model instances and/or templates; and (f) informs the execution engine about changes made in the execution model instances. The decision of actions to be taken and changes in the compositions are made based on the identified problem and the list of available replacement candidate service operations identified by the service discovery component.

III. Adaptation Process

In our framework, the different steps for proactive adaptation of service composition is supported by the use of techniques for the prediction of QoS aspects; the analysis of dependencies between service operations in a composition; and the consideration of groups of service operations in a composition flow instead of isolated operations. As stated above, the adaptation process may be triggered by situations of types C1 to C4. When an event from one of classes C1 to C4 occurs, the monitor, the execution engine, or the composer components notifies the adaptor. The adaptor analysis the event, verifies the need for adaptation, and enforces the necessary changes.

More specifically, as illustrated in Section II, for each user (U_i) of a service composition, an execution model instance ($I_k^{i,j}$) is created from the respective composition template (T_j) using an available list of candidate operations. This process is optimized by maintaining a default combination of candidate operations (S_{T_j}) for each deployed composition template.

An execution model instance can be updated in several ways during its execution by: (a) changing the status of its operations (e.g., from "to be executed" to "executing" and "executed"), (b) changing observed and expected QoS values of the operations, (c) changing the operation endpoints, and (d) changing the composition workflow. For cases (c) and (d), the approach supports change of an operation in the model by another operation or a group of operations, or change of a group of operations already deployed in the model by another operation or a group.

For any of the situations that may trigger the need for adaptation, the process tries to identify other parts in the execution model that may be affected by the situation. The process is based on the identification of *dependencies* between operations. More specifically, the process considers two types of dependencies, namely: (i) *spatial correlation dependency* between operations, services, and providers; and (ii) service *operation signature dependency*.

The *spatial correlation dependencies* are concerned with providers, services, and operations that become unavailable and the impact that this unavailability may have in other service or operations being used in the model. For example, consider a service S that becomes unavailable. In this case, the process considers all other operations of S in the model since these operations may not be able to be executed.

We define that an operation O_2 has a *signature dependency* with an operation O_1 (i.e., O_2 depends-by-signature on O_1), when the output parameter (or its part) of O_1 is used as input parameter (or its part) in O_2. As an example consider operations *DisplayPayInterface():Info and DataCheck (customer:String, cardNumber:String): Boolean* in Figure 3. Assume *Info* a complex data type with parameters *time:Time*, *local:Coordinate*, *name:String*, *cardNumber:String*. In this case, operation *DataCheck ()* depends on operation *DisplayPayInterface (),* since *DataCheck()* uses part of the output parameter of *DisplayPayInterface ()* as one of its input parameters.

A possible solution for the case of signature dependencies is to replace the operations with candidate operations that would satisfy the signature dependency, if they exist. This process of solving the signature dependency may be recursive. This is because candidate operations chosen to solve a signature dependency in a particular point of the execution model may cause signature dependency issues in other parts of the composition.

A. *Detection and Analysis of Problems*

The detection and analysis activities differ for each class of situations C1 to C4, as presented below.

C1: Service composition needs to stop its execution

This case occurs when an operation, a service, or a provider in the execution model instance becomes unavailable. In this situation, spatial correlation dependencies need to be identified. In the case of an operation becoming unavailable, it is necessary to identify if the operation is being used in other parts of the execution model to mark these other occurrences to be replaced as well. In the case in which a service becomes unavailable, it is necessary to identify other operations in the model that are provided by the service since they also need to be replaced. Similarly, in the case in which a provider is unavailable, it is necessary to identify all operations in the model associated with the provider.

When the unavailable operations are identified, they are marked as “out of reach” and the adaptation process needs to find a valid configuration for the composition, by replacing these operations, that respects the SLA values of the composition and signatures dependencies.

In order to illustrate, consider provider *LocalProvider* in Figure 3 as unavailable. In this case, operations *StockCheck* and *DiplayPaymentInterface* have to be marked as “out of reach”, since they are both provided by *LocalProvider*. These operations will need to be replaced.

C2: Service composition can continue its execution, but not necessarily in its best way

This case occurs when a problem does not prevent the service composition to complete its execution, but might cause violations of its SLA parameters. More specifically, this case is concerned with the situation in which one or more QoS values of a service operation O is violated. In this case, the process verifies if the violation of the QoS values for O causes a violation of the SLA parameters for the whole composition, in order to avoid replacing an operation in an execution model when the problem can be compensated by other operations in the model flow. The process also identifies other operations in the model that may be affected due to the violation of the QoS values of O.

For example, suppose that operation *CardPayment* in Figure 3 takes 10300 milliseconds to complete. In this case, the new expected response time for the composition will exceed its *SLA Maximum Time*. Consider another situation in which operation *CardPayment* takes 10014 milliseconds to complete. In this case, the observable response time is bigger than the expected response time (10000 ms), but the *SLA Maximum Time* for composition is not violated.

The verification of the violation of the SLA values for the whole composition is performed by using an *aggregation function* on the *observed* QoS values of the operations that have been executed, and the *expected* QoS values of the operations that are still to be executed.

The observable QoS values used by the aggregation function are provided by the monitors. There are different techniques that can be used to identify *expected* QoS values of operations. For example, in the case of response time values for an operation request, it is necessary to consider the times to execute the operation, send the operation request and receive its response over the network, marshal/unmarshal a request, and maintain a request in the client and server queues. Therefore, to identify expected response times of operations, it is important to use techniques that predict the behaviour of random parameters affected by changes in the network and system resources. An example of such technique is the exponentially weighted moving average (EWMA) [24] that is used in the framework due to its simplicity. Due to space limitations we do not provide in this paper details of the EWMA technique.

The computation of the expected QoS parameters for the whole composition depends mainly on the type of the QoS values to be aggregated and the logic structures of the composition (e.g. conditional, sequence, parallel and repeat logic structures). For example, in Figure 3, the aggregated cost of activities *Data Check* and *Anti-Fraud* is given by the sum of the cost of the operations that perform these activities. In the case of activities *Cancel Payment* and *Payment*, the aggregated cost is given either by the maximum cost of their respective operations or the weighted average using a probability to execute each path as weights. In both cases, the result should be updated after a path is executed in the model. More details about aggregation functions are available in [30].

In the case in which there is no violation of the SLA values for the whole composition, there is no need to adapt the execution model. If one or more SLA values are violated, the adaptation 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 values of the composition.

Different from case C1, in class C2 it is possible to continue the execution of the model even when there are violations in the SLA values. The framework allows the user to decide if s/he wants the process to identify replacement operations when there are SLA violations, or to continue with the execution of the model even in the presence of violations.

C3: Emergence of new requirements

We concentrate on new requirements concerned with QoS aspects. In this case, when changes are necessary to be made, these changes are performed in the composition template T_j and/or in the combination of operations S_{T_j}. This is due to the fact that models that are being executed are following agreements (i.e., set of requirements) that were considered when the model started its execution. However, future executions of the model can consider new set of requirements.

The new requirements can be related to the whole service composition (e.g., the use of the service composition cannot cost more than 0.50 GBP), or to a specific activity in the composition (e.g., the maximum time to receive an invoice should be 2 seconds). In any of these cases, the composer component (see Figure 1) is notified about new SLAs for a certain composition and informs the adaptor about changes in the requirements. The adaptor generates a new composition template with service operation endpoints that comply with the new SLA values.

C4: Emergence of new better services

The emergence of new better services is concerned with the availability of new service operations with QoS values that improve the composition, or updates to candidate operations and their QoS values that could also improve the composition. Similar to case C3 above, in this case changes are performed in the composition template T_j and/or in the combination of operations S_{T_j}. However, in this case, all deployed composition templates will be considered.

B. Decision and Execution of Actions

For any of the cases C1-C4 described in Subsection III.A there may be a need of replacing one or more service operations in the composition template or execution model instances. In this subsection we describe the various steps in the adaptation process when a service operation needs to be replaced. The decisions and actions performed by the adaptor are concerned with the assessment of whether a new set of operation endpoints need to be identified, selected, and used to change the composition.

In order to follow the steps in the adaptation process, consider O a service operation in the execution model instance that needs to be replaced. We assume that a candidate service operation for O is an operation, or a group of dynamically composed operations, identified by the service discovery component (see Figure 1) that can provide the same functionality of O. For instance, in the example in Figure 3, a candidate replacement operation for *StockCheck()* is an operation that verifies the existence of certain good in stock, even if this operation has different signatures, quality or contextual characteristics.

We also assume that a *candidate* operation O' *fully matches* an operation O in the execution model instance if O' and O have the same structural (signature) and the QoS values of O' do not violate the SLA values for the whole composition (functional match is already a criteria for an operation to be a candidate operation).

In *ProAdapt*, when selecting a candidate replacement operation, the process gives preference for matches in the operation signatures, instead of QoS values that do not violate the SLA values of the composition. This is because it is more cumbersome to make changes in the execution model instance when there are other operations in the model that depend on the signature of the one to be replaced. Moreover, differences in QoS values may be compensated by other operations in the model to be executed, or by trying to identify other operations that could compensate the QoS aspects. Furthermore, differences in QoS do not cause the model to stop its execution.

The process considers the various execution logics (regions) in a model such as *sequence, parallel, conditional selection, and repeat logics.* The execution logics are used (i) to calculate the aggregation of *expected* QoS values of operations in the model, and (ii) to identify a group of operations in the model that may need to be replaced by an operation or a group of dynamically composed operations. During the adaptation process, the sizes of the execution logics that are considered are incrementally increased if necessary. When a replacement solution cannot be found for one execution logic El, the process considers the next larger execution logic that contains El.

For example, in Figure 2, if a solution cannot be found for the operations for activity *PaymentInterface,* the next execution logic to be considered is the parallel logic with activities *DataCheck* and *AntiFraud* together with activity *PaymentInterface.* The reason for considering execution logics (or their increments) is to allow a better performance for the adaptation process since it is faster to analyse smaller regions in a model, and find a solution for that region, instead of considering the composition as a whole.

For each operation O in the model involved in the adaptation process, a set of candidate replacement operations Set_Cand(O) is identified for O ordered in descending order of best match. The candidate replacement operations can either have (i) full match with O, (ii) match with only the signature of operation O, or (iii) no match with the signature of O.

During the selection process, the operations in Set_Cand(O) that are considered as replacement options for O are operations that match at least the input parameters of O (as explained above, the operations in Set_Cand(O) provide at least the same functionality of O). An operation O' in Set_Cand(O) is considered as a replacement operation if the data types of the input parameters of O' are subsumed under the input parameters of O. This is because we need to guarantee that the input parameters of O' (yet to be executed) will be able to be defined by values that are available from the parts of the model that have been executed. In the case of O being the first operation in the execution model instance, these values will be defined by inputs to the initial execution of the composition.

For situations in class C1 (service or provider unavailability), O is one of the operations that need to be replaced given that it is possible to have other operations used in the model associated with S or P, different from O. In this case, the process temporarily replaces these other unavailable operations by one of their candidate replacement operations, so that they can be considered when executing the adaptation process. As described below, the process relies on expected QoS values for all the operations in the model to be executed to verify violations of the SLA values of the whole composition. These temporary replacement operations may be changed during the execution of the adaptation process.

The overall idea of the process is to replace O with an operation in Set_Cand(O), by traversing the set of candidate operations in order, and choosing an operation in the set that has the best match with O and does not cause problems to the rest of the execution model, if possible. Examples of these problems are violations of SLA values for the whole composition, or operation signature dependencies that cannot be resolved. The process terminates when either a replacement operation for O is identified in Set_Cand(O), or all the operations in the set are considered without success. When analysing if it is possible to replace O by an operation in Set_Cand(O), other operations not yet executed in the model may also need to be replaced to resolve SLA violations or operation signature dependencies. The adaptation process is recursive; it tries to identify replacement candidate operations for those operations.

In the situation in which operations that match the signature of O are identified, the process verifies if any of these operations can be used to replace O by analysing if there are violations of the SLA values for the whole composition. If the SLA values are not violated, the adaptation process uses the replacement operation to change the execution model, and terminates.

If the SLA values are violated, the adaptation process identifies the list of operations in the model that have not yet been executed. For each of these operations, the process tries to identify combinations of candidate replacement operations that could maintain the SLA values of the whole composition, recursively. The order in which the operations in the list are considered takes into account execution logics (regions) in the model. When necessary, the execution logics are incrementally enlarged, until a combination of replacement operations is identified, or all remaining operations in the model are considered. If a combination of replacement operations is found, the execution model is changed, and the adaptation process terminates.

In the situation in which no operation that matches the signature of O can be used, or they did not exist in the set of candidate operations for O, the adaptation process identifies within the list of operations in the execution model that have not yet been executed, the operations that have a signature dependency with the replacement operation for O. For each of these *dependent* operations, the process verifies if the replacement operation can be used and accepted by the dependent operation, or if replacement operations can be found for a dependent operation.

In the case in which signature dependencies between operations could not be resolved (i.e., no replacement operations can be found for a dependent operation that accepts the replacement of operation O), the process verifies if operations in the execution model can be replaced as a group by a candidate replacement operation, or a set of dynamically composed operations. A group of operations is considered based on execution logics, by adding one operation to the group at each time. If candidate replacement operations are found, the execution model is changed. If not, or there are no more operations to be added to compose an execution logic, the model cannot be adapted, and the adaptation process terminates.

The adaptation of the composition templates and execution model instances is considered an optimization problem based on the selection of appropriate combinations of candidate service operations that satisfy the SLA parameters of a composition. In order to avoid considering all possible combinations which is time consuming, in *ProAdapt* we do not try to identify the best combination, but look for a valid combination given current constraints (e.g.; the candidate operations, the QoS parameters of the operations that have already been executed, and the SLA parameter values of the whole composition). In addition, during the selection of appropriate combinations, the approach ignores those combinations that are identified as invalid in the middle of the selection process. This can happen when candidate operations used in a combination cause the combination to be invalid, independent of other candidate operations in the combination.

Figure 4 presents an example of a possible combination of operation endpoints for a simplified version of the execution model instance shown in Figure 3. More specifically, suppose that the model in Figure 3 has already been modified with activities *e-Payment*, *NotifyCustomer*, and *FinishOrder*, represented as boxes in Figure 3. Each activity in Figure 4 has a set of possible candidate operation endpoints in order of best match. As shown in Figure 4, activity *CheckStock* has three candidate operations while activity *FinishOrder* has only one candidate operation. The numbers on the top of the activities represent the order in which these activities are executed in the composition. In the example, suppose operation *CardPayment* becomes unavailable. Consider that for activity *CheckStock* operation *StockCheck* has already been executed. In the first time of the selection process, the approach identifies combination *e-Payment2, ProcessOrder, GeneralNotification,* , and *StockCheck*. However, suppose this combination does not satisfy the required SLA values for the composition. In this case, the selection process marks the combination as invalid and continues to try for another combination. Suppose the second selected combination satisfies the required SLA values and does not generate signature mismatches. In this case, the selection process identifies *e-Payment3, ProcessOrder, GeneralNotification,* and *StockCheck* as the new valid combination.

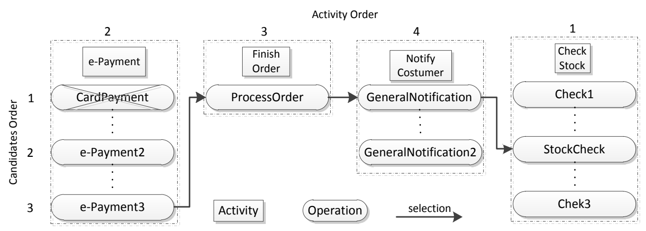


Figure 4: Example of a scenario for the selection of operations.

IV. IMPLEMENTATION AND EVALUATION

A prototype tool of the framework has been implemented. All components of the framework except the service discovery component have been implemented in C++. The service discovery component has been implemented in Java and is exposed as a web service using Apaxe Axis 2 [2]. The tool assumes service compositions in BPEL4WS exposed as Web Services using SOAP protocol. The external service registry uses eXist database [16]. We have also developed a simulator in C++ to represent the infrastructure environment such as providers, services, and operations. The simulator was built on top of the Network Simulator 3 (NS3).

The *ProAdapt* framework was evaluated in terms of the performance of the adaptation process using a Pentium 4 3GHz with 3GB RAM. More specifically, we analysed: the performance of the main activities in the process (case 1); and the performance of the whole adaptation process for problems in different parts of the composition (case 2).

The evaluation was executed in a service composition with 13 activities and different types of execution logics, as shown in Figure 5. The 13 activities in the composition are executed by 13 operations from ten distinct operation endpoints. We considered four candidate replacement operations for each service operation Opi in the composition, which gives a total of 50 different operation endpoints. The operations used in the experiments have different numbers and types of input and output parameters. We also assumed that each operation has at least a signature dependency with another operation.

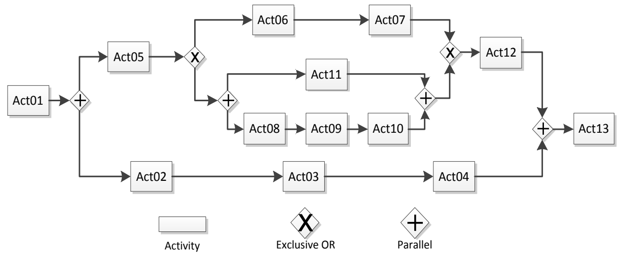


Figure 5: Service composition for evaluation.

In the evaluation, we considered two types of QoS aspects, namely response time and costs, for each service operation and the SLA values of the whole composition. We used these two QoS aspects because of their importance for service-based systems, the strong correlation that exists between response time and cost aspects (there is normally an increase in the cost when providing faster services), and the possibility of demonstrating the prediction of response time values for the composition and their deployed operations. The combinations of the response time and cost values for the operations used in the experiments are (5ms, 25p), (10ms, 20p), (15ms, 15p), (20ms, 10p), (25ms, 5p). We used the same weights for response time and cost values. The 50 endpoint operations in the experiment are associated with 25 different services and five providers.

Case (1): In this case we measured the (a) time to identify if a replacement operation causes violation of the SLA values for the whole composition, (b) time to resolve SLA violations, (c) time to identify signature dependency problems, (d) time to attempt to resolve signature dependency problems, (e) time to adapt by changing groups of operations in a composition, and (f) time to identify spatial correlations due to operation, service, and provider unavailability. Table 1 shows the results of these times in nanoseconds for 100,000 executions with mean, standard deviation (SD), median, minimum, and maximum values.

In activity (a) we measured the time that it takes to compute the aggregation of the observed and expected QoS values of operations and to verify if there is a violation of the SLA values of the whole composition. Note that this time is independent of where a problem occurs in the composition since all the operations need to be considered.

In order to analyse the time for activity (b), we used candidate replacement operations that do not create signature dependency problems since we only want to measure the time that it takes to resolve SLA violations. The result shown in Table 1 is for the worst case scenario in which the combination of replacement operations that satisfies the composition uses all the last candidates for each operation. This case could lead to a situation in which it is necessary to verify a total of 5^{13} possible combinations. As shown in Table 1, the small time achieved by our approach is due to the use of the optimization technique.

TABLE I. PERFORMANCE PER ACTIVITY IN ADAPTATION PROCESS

Act	Cases	Results				
		Mean (ns)	***SD (ns)***	***Med. (ns)***	***Min (ns)***	***Max (ns)***
(a)		593	2	593	589	606
(b)		84704	866	84802	82527	87018
(c)	1 dep	1119	11	1117	1110	1214
	3 dep	3418	18	3419	3410	3498
	5 dep	5891	25	5881	5865	5993
(d)	1 dep	8902	255	8775	8736	9646
	5 dep	25406	333	25259	24965	26215
(e)	G 1	37346	439	37098	36669	38854
	G 2	48653	788	48294	47561	51078
	G 3	56394	514	56528	55575	57987
	G 4	65278	541	65381	64372	67069
(f)	Oper.	885	13	881	871	979
	Serv.	7031	95	1026	6765	7447
	Prov.	18033	333	17960	17368	19276

For analysis of case (c) we considered the situation in which an operation O needs to be replaced and O has a dependent operation O'. We analysed situations in which O' depends only on O, O' depends on three operations including O, and O' depends on five operations including O. The results in Table 1 show the time to verify if O' has dependency problems with one, three, and five operations. The results show that the time to execute this situation increases linearly with the number of dependencies.

In case (d), we considered similar scenarios as in (c), but we measured the times to find a solution. In any of the considered cases, the solution was identified for the last candidate replacement operation. For the case in which there are five dependencies, we used a scenario in which the current operation can only satisfy one dependency; the first candidate can satisfy two dependencies; the third candidate can satisfy three dependencies, and so on until reaching the fifth candidate that satisfies all five dependencies, which creates a total of 19 verifications.

In the case of one dependency, one would expect a ratio of one to five between the time to find a solution (d) and the time to identify the problem (c) in a dependent operation, given that we are using five candidates in the experiment. The additional time shown in the results for the case of one dependency is due to extra activities executed by the approach (e.g., selection of candidate operations and execution of changes in the model). A similar situation is found for the case of five dependencies given that 19 verifications where performed.

The measurements for activity (e) considered a problem in operation Op8 that can only be resolved by trying to replace a group of operations in the composition. We analysed the time to resolve the situation in four different groups of operations. More specifically, we considered the following groups: G1 with operations Op8, Op9; G2 with operations Op8, Op9, Op10, Op11; G3 with operations Op5, Op6, Op7, Op8, Op9, Op10, Op11, Op12; and G4 with all operations in the composition (Op1-Op13). The results in Table 1 show a linear increase with the number of execution logics used in the evaluation, and not with the number of operations being grouped. This is due to the incremental increase of the execution logics by the approach when trying to find a solution.

In case (f) we show the times to identify spatial correlations when an operation, service, and provider become unavailable. We considered these cases as initial steps for resolving problems in class C1, which are followed by the steps in cases (a) to (e), explained above. The time spent in case of provider unavailability is higher than the time for service unavailability, which is higher than the time for operation unavailability, due to the number of operations associated with a service and a provider.

The results in Table 1 demonstrate that case (b) has the higher time of execution due to the large number of combinations of replacement operations that this case needs to consider. It is also observed from the results, that the activity to group operations requires more time to be executed when compared to the other activities.

Case 2: In order to measure the performance of the whole adaptation process, we considered extra signature dependencies between operations in the composition, namely Op3 depends on Op1; Op9 depends on Op5; Op12 depends on Op5; and Op13 depends on Op12 and Op4.

This case was analysed for problems in different parts of the composition. More specifically, we considered problems in operations (i) Op1, (ii) Op8, and (iii) Op11. For case (ii), the problem in Op8 occurred after operations Op1, Op5, and Op2 have been executed. For case (iii), the problem in Op11 occurred after operation Op1, Op5, Op2, Op6, Op7, and Op8 have been executed.

Differently from case 1(b), where the only combination that satisfies the SLA values for the whole composition is the one that uses all the last candidates for each operation, in this case there are other combinations that satisfy the SLA values. In the experiment for case 2, the only combination that satisfies both SLA values of the composition and signature dependencies of operations is the one that uses the last candidate operations.

Table 2 shows the results of these experiments in milliseconds for 100,000 executions with their mean, standard deviation (SD), median, minimum, and maximum values. These results involve all the activities discussed in Table 1. As shown in Table 2, the time for problems in an operation in the beginning of the execution of the composition (Op1) is higher than the time for an operation in the middle of the execution (Op8), which is higher than the time for an operation in the end of the execution (Op11). This is because, there is a need to verify more steps in the adaptation process and consider more combinations of replacement operations when a problem occurs in the beginning of the composition execution instead of towards the end. The probability of finding a solution when a problem occurs in the beginning of the execution is also higher than when the problem occurs towards the end due to the number of available replacement candidates.

TABLE II. PERFORMANCE FOR THE WHOLE ADAPTATION PROCESS

	Problem Location		
Metric	***Op1***	***Op8***	***Op11***
Mean (ms)	1.015005	0.954445	0.206295
SD (ms)	0.004789	0.003574	0.017664
Min (ms)	1.0106	0.950112	0.202703
Median (ms)	1.01336	0.953046	0.203881
Max (ms)	1.04865	0.967082	0.698439

Overall, the initial results of our experiments show that the time for our adaptation process is small when we consider the time that it takes to execute an operation, to send the operation request and receive its response over the network, along with other times involved in the process. A service-based system is composed of several operations, which increases even more the time to execute the operations in the system. The adaptation process can be used without causing considerable performance penalties to the execution of the composition.

V. RELATED WORK

Initial approaches have focused on the generation of service composition during system design, based on anticipated interactions of services and matches between input and output functionality [7][25][26][33]. Other approaches have focused on dynamic service composition, in which services are identified and aggregated during runtime [1][3][10][11][12][21][27].

More recently, some approaches have been proposed to support adaptation of service compositions in a reactive [3][5][18][34] or proactive [13][20][22][32] way. These approaches advocate changes in service composition based on pre-defined policies [5], self-healing of compositions based on detection of exceptions and repair using handlers [34], context-based adaptation of compositions using negotiation and repair actions [3], and KPI analysis and use of adaptation strategies [18].

The work in [13] is based on prediction of performance failures to support self-healing of compositions. It 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 due to several issues and not only performance, as well as dependencies between service operations in a composition.

In [20] 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. Our approach supports changes in compositions due to problems that may occur in any part of the composition.

The works in [22][32] advocate the use of testing to anticipate problems in service compositions and trigger adaptation requests. The approach in [32] 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 [22] test cases are created during the deployment of service compositions and used to identify violations after a service is invoked for the first time. The creation of test cases is not easy. The paper does not specify how test cases are created for modified compositions.

Similar to our approach, in [8] 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.

Some approaches have been proposed to support multi-layered monitoring and adaptation of service compositions [17][28][35]. The work in [28] uses adaptation taxonomy and templates (patterns) created during design time to represent possible solutions for adaptation problems. The work in [17][35] dynamically identifies cross-layered adaptation strategies for software and infrastructure layers. In our work, we also consider multi-layered adaptation aspects. In the above approaches the strategies for adaptation are concerned with the replacement of a service in the composition by another service or a group of services, while our framework also supports the replacement of a group of services in the composition by a single service or a group of services. Our framework supports changes in service compositions during their execution.

In [19][23] the authors propose approaches based on aspect-oriented to support adaptation of service compositions. The work in [23] tackles some QoS aspects and makes use of alternative services for a service that needs to be replaced. The work in [19] suggests two ways for dealing with mismatches in service compositions, namely the creation of stand-alone adapters and the use of aspect-oriented adaptation. Similar to our framework, the work in [19] also tackles the issue of operation signature mismatches. However, in [19] the approach transforms the data types of the operation signatures, instead of identifying replacement operations that can accept the dependency mismatch as in our work.

The *ProAdapt* framework complements existing works for adaptation of service compositions and contributes to the challenge of supporting proactive adaptation during execution time of the service composition, due to different classes of problems. The framework supports distinct ways of adapting the compositions.

VI. Conclusion and Future Work

In this paper we presented *ProAdapt*, a framework to support proactive adaptation of service compositions. The framework supports adaptation triggered by four different classes of problems. The adaptation process allows for the replacement of a single operation in the composition by another operation or a group of dynamically composed operations; or the replacement of a group of operations in the composition by one operation or a group of dynamically composed operations. In the case of QoS violations, the adaptation process verifies if the problem can be compensated by other operations in the composition yet to be executed. A prototype tool has been implemented and used to evaluate the framework in terms of performance from various perspectives.

Currently, we are extending the framework to support service compositions that provide interactions with users and how the adaptation can deal with these interactions and delays that may be caused by them. We are also studying the use of different prediction techniques in the adaptation process to identify if certain techniques can provide prediction earlier than others. We are expanding the prototype to allow for analysis and adaptation of instances of the same composition template, at the same time, and performing other experiments of the approach.

References

[1] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint Driven Web Service Composition in METEOR-S, Int. Conf. on Services Comp. 2004.

[2] Apache.http://ws.apache.org/axis2/.

[3] 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.

[4] R. R. Aschoff and A. Zisman. QoS-driven Proactive Adaptation of Service Composition. In Proc. ICSOC, 2011, pp.421-435

[5] 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).

[6] 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.

[7] BizTalk. http://www.microsoft.com/biztalk.

[8] L. Bodenstaff, A. Wombacher, M. Reichert, M.C. Jaeger. Analyzing Impact Factors on Composite Services, IEEE Int. Conf. on Services Computing, September, 2009.

[9] BPEL4WS. ibm.com/developerworks/library/specification/ws-bpel/

[10] G. Canfora, M. Di Penta, R. Esposito, M. L. Villani. QoS-Aware Replanning of Composite Web Services, IEEE Int. Conf. on Web Services, 2005.

[11] F. Casati, S. Ilnicki, L.J. Jin, V. Krishnamoorthy, and M.C. Shan. Adaptive and Dynamic Service Composition in eFlow. www.hpl.hp.com/techreports/2000/HPL-2000-39.pdf

[12] 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.

[13] 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.

[14] 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.

[15] S. Dustdar and M.P. Papazoglou. Services and Service Composition – An Introduction. IT Information Technology, 2(2008), pp. 86-92.

[16] eXist. http://exist.sourceforge.net.

[17] S. Guinea, G. Kecskemeti, A. Marconi, and B. Wetzstein. Multi-layered Monitoring and Adaptation. In Proc. of the 9th Int. Conf. on Service Oriented Computing, December, 2011, Cyprus.

[18] 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.

[19] W. Kongdenfha, H. R. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul. Mismatch patterns and Adaptation Aspects: A Foundation for Rapid Development of Web Service Adapters. IEEE Transactions on Services Computing, V. 2, N. 2, April-June 2009.

[20] 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.

[21] MAIS Project. Mobile Information Systems – Infrastructure and Design for Flexibility and Adaptability, B. Pernici, Springer, 2006.

[22] 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, SEAMS, South Africa, May 2010.

[23] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive Monitoring and Service Adaptation for WS-BPEL. In 17th International World Wide Web Conference, China, 2008.

[24] NIST/SEMATECH eHandbook of Statistical Methods, www.itl.nist.gov/div898/handbook

[25] OpenStorm. http://www.openstorm.com.

[26] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leyman, and B. Kramer. Service-Oriented Computing Research Roadmap. http://tinyurl.com/6jhvd44.

[27] 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.

[28] R. Popescu, A. Staikopoulos, P. Liu, A. Brogi, and S. Clarke. Taxonomy-driven Adaptation of Multi-layer Applications Using Templates, In 4th IEEE International Conference on Self-Adaptive and Self- Organizing Systems. Hungary, September/October, 2010.

[29] F. Salfner, M. Lenk, and M. Malek. A Survey of Online Failure Prediction Methods. ACM Computing Surveys, 42(3), 2010.

[30] D. Schuller, A. Polyvyanyy, L. García-Bañuelos, and S. Schulte. Optimization of Complex QoS-Aware Service Compositions. In Proc. ICSOC, 2011, pp.452-466.

[31] G. Spanoudakis and A. Zisman. Discovering Services during Service-based System Design using UML. IEEE Transactions of Software Engineering, 36(3): 371-389, 2010.

[32] D. Tosi and G. Denaro and M. Pezzè. Towards Autonomic Service-Oriented Applications. International Journal of Autonomic Computing (IJAC), 2009, pp. 58—80

[33] WebSphere. http://www.ibm.com/software/info1/

[34] WSDiamond.http://wsdiamond.di.unito.it/status.html.

[35] A. Zengin, R. Kazhamiakin, and M. Pistore. CLAM: Cross-Layer Management of Adaptation Decisions for Service-Based Applications. In Proc. ICWS, 2011.

[36] A. Zisman, J. Dooley, G. Spanoudakis. A Framework for Dynamic Service Discovery, Int. Conf. on Automated Software Engineering, Italy, 2008.