A Service Selection Model to Improve Composition Reliability

Natallia Kokash [1]

Abstract. One of the most promising advantages of web service technology is the possibility of creating added-value services by combining existing ones. A key step for composing and executing services lies in the selection of the individual services to use. Much attention has been devoted to appropriate selection of service functionalities, but also the non-functional properties of the services play a key role. Due to ever changing business environment, service users are not guaranteed from unexpected service faults. Service composition can compensate the deficiencies of constituent components aggregating a redundant number of them for individual tasks. This paper proposes a service selection strategy targeting at minimizing the impact of atomic service failure on the quality of service of compositions.

1 Introduction

Service-Oriented Architecture (SOA) is an upcoming organizational model aiming at simplifying large-scale business operations by consumption of ready-to-use services. The most prominent realization of SOA is currently in the area of web services. Web services are loosely-coupled, platform-independent, self-describing software components that can be published, located and invoked via the web infrastructure using a stack of standards such as SOAP, WSDL and UDDI [12].

Composition of web services is probably the most interesting challenge spawned by this paradigm. Many efforts have been devoted to development of automatic or semi-automatic service composition mechanisms [1] [10] [16] [15]. We are striving for runtime composition, when a composer automatically searches, binds and executes available web services to satisfy some user request. Such a scenario is hardly feasible for a large spectrum of applications. Automatic service compositions are error prone. State-of-the-art techniques are not mature enough to guarantee a common semantic of the involved operations. Testing, adaptation, verification and validation processes are required. However, statically composed services, able to accomplish some generalized request classes (e.g., user trip planning), can presume manifold alternative components for each subgoal and practice dynamic switching between them. A set of services to satisfy a particular user request is selected depending on conditions like Quality of Service (QoS) parameters, service provider policy or user preferences. Several approaches for runtime selection of component services have been developed [7] [17] [18]. They tend to consider multiple quality factors and search for a solution that optimizes weighted composition of their average values under user constraints. In this paper, we propose a novel service selection strategy. Our approach differs from the existing works in two aspects. First, it is well-known that good average statistics do not prevent from unexpected service faults. We study the problem of service selection assuming their probable failures. The solution is represented by several compositions able to satisfy given constraints on quality parameters and to reduce fault recovery expenses. Second, we use a single quality measure, that, though, takes into consideration the correlation between service reliability and other parameters.

The paper is structured as follows. Section 2 discusses related work. In section 3, quality and web service composition models are outlined. Section 4 introduces a compound quality measure that characterizes reliability of redundant service compositions. Web service selection mechanism we propose is presented in Section 5. Finally, conclusions and future work are sketched in Section 6.

[1] Department of Information and Communication Technology, University of Trento, Via Sommarive, 14, 38050 Trento, Italy, email: kokash@dit.unitn.it

2 Related Work

In this section, we cover related work on QoS of atomic web services and web service compositions.

2.1 Quality of Atomic Web Services

Ran [13] describes basic non-functional QoS parameters. In Table 1, we list most common numeric factors that will be used below. *Service Level Agreement (SLA)* defines the agreed level of performance for a particular service between a service provider and a service user. The *Web Service Level Agreement (WSLA)* project [6] is targeted at defining and monitoring SLAs for web services. SLA parameters can be measured with different *metrics*, including composite ones like maximum response time or average availability. Composite metrics are specified by *functions* which are executed during the predefined time

Table 1. QoS factors of web services [13]

QoS	Description
Throughput	The number of requests served in a given time period.
Capacity	A limit of concurrent requests for guaranteed performance.
Latency	The round-trip time between client request and service response.
Response time	The time taken by a service to process its sequence of activities.
Availability	The probability that a service is available.
Reliability	Stability of a service functionality, i.e., ability of a service to perform its functions under stated conditions
Reputation	The average rate of the service reported by clients.
Execution cost	The amount of money for a single service execution.

intervals, specified by *schedules*. Sherchan et al. [14] report the relevance of recent service performance measurements. Service quality parameters depend on manifold factors which should be taken into account at the early stages of development. Menasce et al. [11] provide a methodology for planning service capacity. Knowledge about the number of potential users, frequency of service invocations and their time distributions is essential for the analysis. An accurate capacity planning can be quite problematic because of factor uncertainty or limited budget. As a consequence, a significant number of troublesome services could appear.

2.2 Quality of Web Service Compositions

Service compositions that embed low-quality services inherit all their drawbacks. This poses a big challenge for the software developers building new systems on the basis of available components. Cardoso et al. [4] describe the model that allows to predict quality of service for workflows based on atomic service QoS attributes. One can compensate composition deficiency if many services with compatible functionality exist. Several approaches have been proposed for quality-aware service selection. Zeng et al. [18] consider the service selection task as a global optimization problem. Linear programming is applied to find the solution that represent the service composition optimizing the target function. The latter is defined as a linear combination of five parameters: availability, successful execution rate, response time, execution cost and reputation. If the global characteristics (e.g., the total amount of money to accomplish the user goal), are not restricted, the optimal solution can be found by modified Dijkstra's algorithm searching on the graph of available compositions [8]. Martin-Diaz et al. [7] propose a constraint programming solution for procurement of web services whose demands and offers are temporal-aware. In [3] the problem is modelled as a mixed integer linear program where both local and global constraints are specified. Yu and Lin [17] modelled the service selection as a complex multi-choice multi-dimension 0-1 knapsack problem. Practice of offering different quality levels by services was taken into consideration. The above solutions depend strongly on the user weights assigned to each parameter. However, it is not trivial for a user to establish them in right way. An example in Table 2 demonstrates limitations of the algorithm in [18]. Let $w_1 = 0.6$ and $w_2 = 0.4$ reflect the user scores for response time and availability, correspondingly. After scaling and weighting phases service s_1 will be chosen. However, the services have a small difference in response time and a huge diversity in availability rate. As a fast remedy, the intuition about absolute QoS values should be involved, e.g., from 0 to 100% for availability and from 0 to $timeout$ ms for response time. The principal drawback of the approach is that the cross-impact of different quality parameters is not considered.

Table 2. QoS-aware WS selection [18]

	Response time (ms)		Availability (%)		Result
	Original	Scaled	Original	Scaled	$w = (0.6, 0.4)$
s_1	1	1	10	0	0.6
s_2	2	0	100	1	0.4

Despite the efforts aimed at insuring web service reliability service composition failures are almost inevitable. Nevertheless, they can be gently treated and do not lead to the composition breakdown. Two basic approaches for error recovery exist, namely *backward* and *forward*. The first one assumes the presence of redundant data allowing to analyze the detected fault and put the system into a correct state. The second one returns the system into a previous fault-free state without requiring detailed knowledge of the fault. Workflow systems rely intensely on backward error recovery if the resources are under the control of a single domain. Forward recovery is extensively used to handle errors in composite web services. Chafle et. al [5] propose a mechanism for fault propagation and recovery in decentralized service orchestrations. Decentralized architecture results in additional complexity requiring fault propagation between partitions executed independently.

3 Web Service Composition Model

As opposed to the discussed service selection approaches we do not consider multiple QoS factors. Service reputation is excluded due to its subjective nature. High reputation of elementary services does not imply high reputation of their compositions due to potential problems with input data and effect/precondition satisfaction that cannot be fully verified at the planning phase. We do not consider the nature of web service faults and rely solely on *probability of* service *success*. It can be defined as

$$p(s) = N_{suc}(s)/N_{total}(s),$$

where N_{suc} is the number of successful service responses and N_{total} is the total number of observed invocations. A service invocation is considered to be *successful* if the user goal is satisfied or we can proceed along with an execution of a composite service, i.e., (1) a constituent service was available, (2) the successful response message was received within an established timeout, (3) no errors were detected automatically during the output and effects checking, (4) preconditions of a subsequent service were satisfied. Along with the probability of success we often use the *probability of failure* $p(\bar{s}) = 1 - p(s)$. The other relevant parameters are *response time* $q_{time}(s)$ and *execution cost* $q_{cost}(s)$ of service s.

A composite web service can be defined in terms of the standard BPEL (stands for Business Process Execution Language) [2] or other composition languages [9]. Services are composed by the following three operators

$$C ::= (s_1; s_2) \mid (s_1|s_2) \mid (s_1 + s_2),$$

where $(s_1; s_2)$ denotes the sequential, $(s_1|s_2)$ the parallel and $(s_1 + s_2)$ the choice composition of services s_1 and s_2. If an error happens in one of the parallel services, we suppose it will be correctly forwarded to the end of partition [5]. After that we should decide whether the whole parallel composition is failed or only the erroneous branch has to be recovered. For the sake of simplicity we will not consider parallel compositions here. Since we are mostly interested in the non-functional qualities, parallel compositions can be approximately reduced to sequential ones as follows: for service $c = (s_1|s_2)$ its execution cost will be

$$q_{cost}(c) = q_{cost}(s_1) + q_{cost}(s_2),$$

response time

$$q_{time}(c) = max(q_{time}(s_1), q_{time}(s_2)),$$

probability of success

$$p(c) = p(s_1)p(s_2)$$

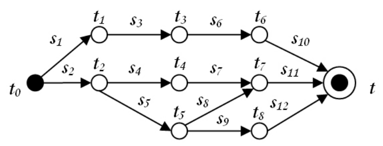

Figure 1. Composition graph

and probability of failure

$$p(\overline{c}) = p(\overline{s}_1) + p(\overline{s}_2) - p(\overline{s}_1)p(\overline{s}_2).$$

Service composition can be represented as Directed Acyclic Graph (DAG) $G = (S, T)$ with two distinguished vertices (start and end points). Nodes denote states $T = \{t_j \,|j = 1, ..., n\}$ and edges are labelled to represent available web services $S = \{s_i \,|i = 1, ..., m\}$. Generally, multiple edges between two nodes are permitted. The processing of a user request begins from the start state t_0. In the end state t the user goal is accomplished. These two states are well-defined for a given class of problems. It is assumed that all web services have deterministic behavior. A graph $G = (T, S)$ is called a *composition graph* (see Figure 1). Note that branching in the composition graph corresponds to choice operator. Further, $tail[s_i]$ will denote a tail and $head[s_i]$ a head of an edge s_i. Let also $d_{in}[t_j]$ and $d_{out}[t_j]$ refer to input and output degree of a state t_j, correspondingly.

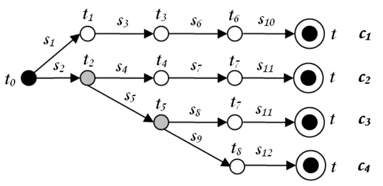

Figure 2. Configuration tree

A composition with choice operators can be separated into several sequential compositions, called *configurations*, that correspond to paths between start and end states in a composition graph. Two configurations are *independent* if they do not have common states except start and end ones, and *dependent* otherwise. All possible configurations can be shown on the *configuration tree* (see Figure 2). The start state of a composition graph corresponds to the start state of the associated configuration tree, and nodes with the same labels have the same output degree.

Definition 1 *Node t_i of a composition graph G is a* return state *if either it is the start state or its outdegree $d_{out}(t_i) > 1$.*

The above statement defines the states in which a composite service can be returned to recover a failure of its components. For the composition graph in Figure 1 return points are $\{t_0, t_2, t_5\}$. A node t_i is a return state of a configuration tree iff it is such in the corresponding composition graph.

4 Fault-tolerant Service Compositions

4.1 Fault Recovery

Existing service composition selection models do not take into account unpredictable service faults. The problem cannot be fully resolved by discarding the failed service. The time for finding a new solution from scratch increases latency for the requests arrived in the system. We propose to choose several configurations with good qualities at the selection stage. If a failure occurs in the chosen configuration, another configuration can be switched into.

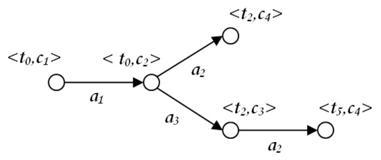

Figure 3. Execution plan

In Figure 3, a possible execution plan is shown. A tuple $\langle t_j, c_i\rangle$ with an incoming edge a_k means that configuration c_i is started from state t_j after event a_k. In this example a_1 means any failure of composition c_1, a_2 denotes a failure of service s_{11} and a_3 refers to a failure of services s_4 or s_7. A set of events and recovery actions can be established automatically based on the analysis of the dependency between configurations. In more complicated scenarios the events like *input/output is not correct/complete, preconditions/postconditions are not satisfied, message is lost, exception is raised by a service, connection error, timeout is expired, SLA is violated,* etc., can require different reactions. At this point we can start analyzing the error types in order to choose the optimal system behavior.

4.2 Failure Risk

Let $c = (s_1; s_2; ...; s_k)$ be a sequential composition with the only return point $head[s_1]$. If a service s_i fails a new configuration can be started from state $head[s_1]$. The results of services $\{s_1, s_2, ..., s_i\}$ will not be used whereas their response time and execution cost increase the total expenses to satisfy a user request. We refer to these expenses as to the *loss function* of a service s_i failure. There are no reasons to impose penalties on services $\{s_1, ..., s_{i-1}\}$ since they are not responsible for errors of service s_i. We propose a service selection strategy based on the analysis of failure risks and targeted at decreasing the expected loss in such a scenario.

Failure risk is a characteristic considering probability that some fault will occur and the resulting impact of this fault on the composite service. For an atomic service s_i it equals

$$r(s_i) = p(\overline{s}_i)q(s_i).$$

Service s_1 is considered to be better than service s_2 if it has a smaller failure risk $r(s_1) < r(s_2)$.

Within the sequential composition $c = (s_1; ...; s_k),\ k > 1$, the risk of a service s_i failure is measured as

$$r(s_1; ...; s_i) = p(s_1; ...; s_{i-1}; \overline{s}_i)q(s_1; ...; s_i)$$

where

$$p(s_1;...;s_{i-1};\overline{s}_i) = \prod_{j=1}^{i-1} p(s_j)p(\overline{s}_i)$$

is the probability of composition failure while service s_i is being executed, and

$$q(s_1;...;s_i) = \sum_{j=1}^{i} q(s_j)$$

is the loss function of this failure. Since for a sequential composition all services should be invoked to accomplish a task, we exploit a total risk value which for a k-service long chain equals

$$R(c) = R(s_1;...;s_k) = \sum_{i=1}^{k} r(s_1;...;s_i). \quad (1)$$

Let $c = (s_1 + ... + s_l)$, $l > 1$, be a choice composition such that a service s_i is invoked if the alternatives $\{s_1, ..., s_{i-1}\}$ failed. The failure risk of an i-service long choice is defined as

$$r(s_1 + ... + s_i) = p(\overline{s}_1;...;\overline{s}_i)q(s_1;...;s_i),$$

where

$$p(\overline{s}_1;...;\overline{s}_i) = \prod_{j=1}^{i} p(\overline{s}_j).$$

The choice composition fails if all invoked services fail. Obviously, the better candidates should be tried first. The question then is how many services to consider. We propose to include a new candidate if it does not increase the total failure risk for the choice composition, i.e.

$$R(c) = R(s_1 + ... + s_l) = \min_{i=\overline{1,l}} r(s_1 + ... + s_i). \quad (2)$$

Finally, the failure risk of a composition $c = (c_1; c_2)$, where $c_1 = (s_1; ...; s_k)$ is a sequential composition and $c_2 = (s'_1 + ... + s'_l)$ is a choice one, can be computed as

$$R(c) = R(c_1; c_2) = R(c_1) + p(c_1)p(\overline{c}_2)\big(q(c_1) + q(c_2)\big).$$

Above, $q = \{q_{time}, q_{cost}\}$ refers either to response time or to execution cost. If the user preferences are given, it can incorporate both of them: $q(s) = f(q_{time}, q_{cost})$. Assuming that f is a linear combination, we get

$$q = w_1 q_{time} + w_2 q_{cost} \,|\, w_1 + w_2 = 1,\ 0 \le w_1, w_2 \le 1.$$

For simplicity we supposed that the money for the invocation of a failed service s are not payed back and the time to detect the error does not differ significantly from the usual service response time. If this is not the case, the corresponding corrections can be introduced, i.e., the loss function of a service s will be

$$q(s) = \{q_{ttd}(s), q_{cost} - q_{penalty}(s)\},$$

where q_{ttd} stands for error *time-to-detect* and $q_{penalty}$ is a *penalty* payed by the provider of a failed service.

Failure risk is a compound measure considering probability of constituent service failures, their response time and/or execution cost along with the structure of a configuration tree. Intuitively, configurations with frequent return points are more preferable. However, the composition selection will depend on the balance between all mentioned parameters. We do not strictly require that the loss function should be linear. So, we can measure the response time of a composite web service with a certain probability as follows: Let $P_1(t \le t_1)$ and $P_2(t \le t_2)$ be the probabilities that services s_1 and s_2 respond within time t_1 and t_2, respectfully. Let $P(t \le t_0)$ be the probability that a composite service $s = (s_1; s_2)$ will respond within time t_0. Letting $p_1(t)$ and $p_2(t)$ be the corresponding probability density functions, $p(t)$ can be calculated as a convolution $p(t) = \int p_1(t-\tau)p_2(\tau)d\tau$ and $P(t \le t_0)$ can be obtained by taking corresponding distribution $P(t \le t_0) = \int_0^{t_0} p(t)dt$.

5 Web Service Selection Methodology

5.1 Objectives

It is reasonable to assume that an SLA with the end user is established in such a way that available service configurations can satisfy the constraints on response time and execution cost provided the normal conditions, i.e., that the services are not overloaded and no failures befall. However, the unexpected faults of component services lead to resource loss and may cause the violation of negotiated parameters. If there is reserve of the resources (the maximum budget for a task is not reached and there is time left before task execution deadline), a user task can be completed by another configuration. Our objective is to choose an execution plan (see Figure 3) that maximizes composition reliability, which is actually defined by the probability to accomplish a user request within established time and cost boundaries.

The problem of increasing service reliability differs from the search of a single configuration with optimal quality parameters. For a single-objective function the latter one can be formalized as selection of a path $(s_1; ...; s_k)$ between the start and end states that maximizes the following target function:

$$f(c) = p(c)(q^{max} - q(c)) = p(s_1;...;s_k)(q^{max} - q(s_1;...;s_k)) = \\ = \prod_{i=1}^{k} p(s_i)(q^{max} - \sum_{i=1}^{k} q(s_i)),$$

where q^{max} defines the resource limit, taken from an SLA (or chosen big enough to guarantee the positive value of $f(c)$). Such a configuration can be found in time $O(m)$, where m is a number of available web services, by searching of a path in a configuration graph optimizing the formula above.

However, practical problems rarely have only one objective. The simplest multi-objective problems focus on two criteria. This subclass of problems is known as *dual criteria optimization*. The basic approach is to take the less important parameter as objective function provided that the most important criterion meets some requirements. We identify two basic dimensions, namely, *response time* and *execution cost*. Although it is hard to predict which of them is more important we suppose that for a provider of composite web services focus should be put on response time. Usually, the internal structure of services is hidden from the end-user, so (s)he expects to pay the fixed price for a single service execution (provided the same quality level). At the same time, service delays can be indemnified by penalties. On conditions that response time constraint is satisfied, a provider can optimize its own expenses.

5.2 Failure Risk Evaluation Algorithm

In this section we present our risk evaluation model for composite web services.

To simplify the explanation of the proposed algorithm we will use the notion of graph contraction. The *contraction of an edge* (v_i, v_j) of a directed graph is the directed graph obtained by replacing two nodes v_i and v_j with a single node v such that v is a head of the edges incoming to the original vertices and a tail of the edges outgoing from them. The *contraction of a graph* is a process of transforming the graph by applying the operation of edge contraction. A sequential composition $(s_1; ...; s_k),\ k > 0$, can be seen as an elementary service with the failure risk obtained by formula (1) (see Figure 4).

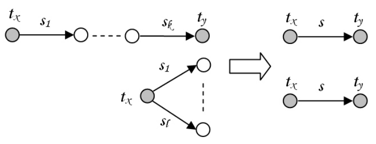

Figure 4. Contraction operations in a composition graph

To show this graphically we can contract the path $(s_1; ...; s_k),\ k > 0$, in the configuration tree to a single edge s. After replacing all the paths we will get a reduced tree, where each node is either a return point or the end state, still unambiguously representing all possible configurations (see Figure 5). Such a tree is called a *contracted configuration tree*.

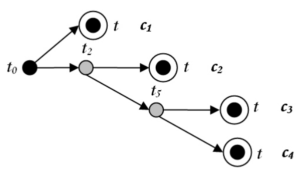

Figure 5. Contracted configuration tree

Any choice composition $(s_1 + ... + s_l),\ l > 0$, can be seen as an elementary service with the failure risk obtained by formula (2). However, we can calculate failure risks of composite services with alternative components only if the failure risks of the latter ones are known, which, in their turn, can be composite as well.

The above observations define our failure risk evaluation algorithm (see Algorithm 1). Given a composition graph, it computes the failure risks of composite services defined by the subtrees of any return point. Given these values, a weighted contracted configuration tree can be formed where edges with smaller weights represent the most promising directions.

For each composite service s_i let $stack[s_i]$ be a structure where QoS parameters of constituent services are accumulated. Starting from the end state we gradually contract edge chains and partitions assigning failure risks to the corresponding services as has been discussed in Section 4.2. The end state t is the only point satisfying the condition $d_{out}[t] = 0$ at the initial step of the algorithm. Since a composition graph is acyclic, after removing of an edge s_i we again will have the states without outgoing edges. When we reach a return point, the chain in the stack of s_i is contracted and its failure risk is measured by (1). Provided that no more sequences can be processed, the algorithm switches into choice compositions. Each choice composition is replaced by a service with the failure risk computed by (2), and this information is transmitted to the precedent edges. Then, we repeat sequential composition replacement again, and so on, terminating if no more composite services are found in graph G'.

Algorithm 1. Failure Risk Evaluation (*FRE*)

```
1   $G' \leftarrow G,\ G' = (T', S')$
2   while $|S'| \geq 1$ do
3     Sequential($G'$)
4     Choice($G'$)

Sequential($G'$)
1   for all $s_i \mid d_{out}[head[s_i]] = 0$ do
2     if $(d_{out}[tail[s_i]] = 1)$ and $(tail[s_i] \neq t_0)$ then
3       for all $s_j \mid head[s_j] = tail[s_i]$ do
4         $stack[s_j] \leftarrow stack[s_i] \leftarrow s_j$
5       $S' = S' \backslash s_i$
6     else //$tail[s_i]$ is a return point
7       $s \leftarrow$ sequential composition of services $stack[s_i]$
8       remember $R(s)$ as a weight of the edge $s_i$ in graph $G$
9       $S' \leftarrow S' \backslash s_i \cup s$

Choice($G'$)
1   for all $t_i \mid (d_{out}[t_i] > 1) \cup (\forall t_k \in adj[t_i],\ d_{out}[t_k] = 0)$ do
2     $s \leftarrow$ choice composition of services $\{s_j\} = \{(t_i, t_k), \mid \forall t_k\}$
3     for all $s_j \mid head[s_j] = t_i$ do
4       $stack[s_j] \leftarrow s$
5     $S' \leftarrow S' \backslash \{s_i\}$
```

Proposition 1 *Given a composition graph $G = (T, S)$, the algorithm FRE measures the failure risks of the composite services defined by the subtrees of return points in polynomial time from number of available web services.*

A proof follows from the next observations:

***Sequential(G')*:** The information about each service $s_i, \mid d_{out}[tail[s_i]] = 1$, is pushed into $d_{in}[tail[s_i]]$ stacks and can be processed in time $O(1)$ later. So, for any state $tail[s_i]$, a time $O(d_{in}[tail[s_i]])$ is required and this state will not be considered again by this function. Summing up by all states we get the process time of all sequential compositions $O(\sum_{i=1}^{n} d_{in}[tail[s_i]]) = O(m)$.

***Choice(G')*:** For each state t_i time $md_{out}[t_i]\ln(d_{out}[t_i])$ is required to calculate the failure risk and $d_{in}[t_i]$ to transmit this information for the precedent services. After that its outgoing edges are deleted and it will not be considered again. Hence, the processing of all choice compositions can be accomplished in time $O\big(\sum_{i=1}^{n}(d_{in}[t_i] + md_{out}[t_i]\ln(d_{out}[t_i])\big) = O(m^2\ln(m))$.

A composition configuration can be chosen by a simple greedy heuristic that selects a service with the least failure risk in any return state. In case of a service failure, gradual change is preferable to sudden, large-scale switching into other configuration since we maximally reuse the results of already invoked services, that is, a composite service should go back to the nearest return point and complete the job by ”attaching” a new configuration. The alternative choices with poor quality (i.e., ones that increase the total failure risk) are excluded from the calculation of failure risk of choice composition. However, such services still can be invoked to recover a failure if there are no better alternatives and the loss of rollback to the next return point is significant.

6 Concluding Remarks and Future Work

Risk analysis is an important process accompanying the development of any significant software system. Being business-oriented applications, constructed from uncontrolled components and used via the Internet, web service compositions imply a long list of potential risks, varying from security threats to economic unprofitableness. In such conditions self-adaptivity and redundancy are desirable composition qualities traded against development cost and performance efficiency.

We proposed a model for selecting web services that aims at increasing the reliability of service compositions. We introduced the notion of failure risk for web services and proposed a composition failure risk evaluation methodology. Further, we briefly described the selection algorithm based on local analysis of promising configurations.

Several important issues have been left out of the scope of this paper. Careful study of parallel service compositions and dependability between different configurations is needed. Comprehensive experiments should be carried out in order to characterize the effectiveness of our model. Here we pursued the goal of failure risk minimization provided that a request should be executed within some constraints on time and cost, but without intention to minimize these expenses. The problem of finding an execution plan that maximizes the probability to fulfill the task and spend minimal resources can be tackled in the same manner. In some cases we can distinguish *permanent faults* and *temporal faults* (e.g., intermittent server crashes). The latter are characterized by *time to repair*. It can be reasonable to repeat the invocation of the same service instead of switching into an alternative configuration.

We intend to deeper investigate the optimization problems related to usage of web service compositions. As a future work we are planning to look into the perspectives of using advanced scheduling policies. Real-life conditions such as limited service capacity, failures, execution deadlines, provider vs. client interests, etc., transform development of reliable composite services into a challenging task. Static selection models can be useful if the chosen services require significant adaptation efforts and involvement of alternative ones is not feasible or expensive. In dynamic scenarios the end-user would prefer a fast service despite of its low average availability if it is available at the moment of invocation, or a slower service provided that the faster one is overloaded.

Acknowledgments

We are grateful to Marco Aiello for his useful comments on the paper.

REFERENCES

[1] Aggarwal, R., et al.: ”Constraint-driven Web Service Composition in METEOR-S”, *IEEE Conference on Service Computing*, 2004.

[2] Andrews, T., Curbera, T. et al.: ”Business Process Execution Language for Web Services”, 2003, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

[3] Ardagna, D., Pernici, B.: ”Global and Local QoS Constraints Guarantee in Web Service Selection,” *IEEE International Conference on Web Services*, 2005, pp. 805–806.

[4] Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: ”Quality of service for workflows and web service processes”, *Journal of Web Semantics*, Vol. 1, No. 3, 2004, pp. 281–308.

[5] Chafl, G., Chandra, S., Kankar, P., Mann, V.: ”Handling Faults in Decentralized Orchestration of Composite Web Services”, *International Conference on Service-Oriented Computing*, 2005, pp. 410–423.

[6] Dan, A., Davis, D., Kearney, R., et al.: ”Web services on demand: WSLA-driven automated management”, *IBM Systems Journal*, Vol. 43, No. 1, 2004, pp. 136–158.

[7] Martin-Diaz, O., Ruize-Cortes, A., Duran, A., Muller, C.: ”An Approach to Temporal-Aware Procurement of Web Services”, *International Conference on Service-Oriented Computing*, 2005, pp. 170–184.

[8] Gu, X., Chang, R.: ”OoS-Assured Service Composition in managed Service Overlay Networks”, *IEEE International Conference on Distributed Computing Systems*, 2003.

[9] Kokash, N., D'Andrea, V.: ”Service Oriented Computing and Coordination Models”, *Proceedings of Challenges in Collaborative Engineering Workshop*, 2005, pp. 95–103.

[10] Lazovik, A., Aiello, M., Papazoglou, M.: ”Planning and monitoring the execution of web service requests”, *International Conference on Service-Oriented Computing*, 2003, pp. 335-350.

[11] Menasce, D. and Almeida, V.: Capacity Planning for Web services, Prentice Hall, Upper Saddle River, NJ, 2002.

[12] Papazoglou, M. P., Georgakopoulos, D.: ”Service-oriented computing”, *Communications of the ACM*, Vol. 46, No. 10, 2003, pp. 25–28.

[13] Ran, Sh.: ”A Model for Web Services Discovery With QoS”, *ACM SIGecom Exchanges*, Vol. 4, No. 1, 2003, pp. 1–10.

[14] Sherchan, W., Krishnaswamy, Sh., Loke, S-W.: ”Relevant Past Performance for Selecting Web Services”, *International Conference on Quality Software*, 2005, pp. 439–445.

[15] Srivastava, B., Koehler, J., ”Web Service Composition - Current Solutions and Open Problems”, *Proceedings of ICAPS Workshop on Planning for Web Services*, 2003.

[16] Sirin, E., Hendler, J., Parsia, B.: ”Semi-automatic Composition of Web Services Using Semantic Descriptions”, *In Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS*, 2003.

[17] Yu, T., Lin, K.J.: ”Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints”, *International Conference on Service-Oriented Computing*, 2005, pp. 130–143.

[18] Zeng, L., Benatallah, B., et al.: ”QoS-aware Middleware for Web Services Composition”, *IEEE Transactions on Software Engineering*, Vol. 30, No. 5, 2004, pp. 311–327.