Dependable Web Service Compositions using a Semantic Replication Scheme

Daniela Barreiro Claro[1], **Raimundo José de Araújo Macêdo** [1]

[1]Laboratório de Sistemas Distribuídos - LaSiD
Departamento de Ciência da Computação
Universidade Federal da Bahia
Av. Adhemar de Barros, s/n - Campus de Ondina
Salvador, BA – Brazil CEP. 40170-110

{dclaro,macedo}@ufba.br

Abstract. *The broad acceptance of a Web service standard has led enterprises worldwide to publish their services and make businesses via the Web on the Internet. Consequently, dependable Web service executions are a new challenge. Although existing work proposes to extend the Web service structure with fault tolerant features that support such applications, most of them meet only the reliability and availability requirements of single Web service executions, not properly addressing the problem of dependable Web service compositions. This paper overviews existing work on available Web service compositions and proposes a new approach to create highly available compositions based on a semantic replication scheme. A prototype of the proposed approach was evaluated in a series of experiments where Web service failures are considered and the related performance data are presented.*

1. Introduction

Web services are autonomous applications that can be published, located, and invoked over the World Wide Web. Because their potential for heterogeneous service integration, today there is an increasing amount of companies and organizations that implement their core business and outsource other application services over the Internet. In such a scenario, it is frequent that no single Web service can suit the functionality required by the user, leading to the need to combine existing services together in order to fulfill the user request. Such a combination of Web services is called a Web service composition (WSC).

Whereas Web service specifications cover a number of issues ranging from security to transaction support, by now no specification has addressed the problem of dynamic Web service compositions. This has motivated a considerable number of research efforts on the composition of Web services both in academia and in industry [Hull and Su 2004, Hakimpour et al. 2005, Aggarwal et al. 2004, Rajasekaran et al. 2004, Andrews et al. 2003, Martin et al. 2004]. In particular, a good deal of this effort is devoted to the research of automatic compositions, for instance, by exploiting AI planning techniques [Martinez and Lesperance 2004, Ugur et al. 2004].

On the other hand, building applications from the automatic assembling of existing Web services raises another important concern: the failure of a single Web service can lead to the failure of the whole composition. Therefore, availability or continuity of service requirements must be taken into account if one would like to apply WSC in critical applications such as health systems and stock markets [Birman et al. 2004]. Hence, the construction of dependable Web services compositions (WSC) has also deserved some attention from the research community in the last years [Majithia et al. 2004, Mikalsen et al. 2002, Bhiri et al. 2005, Pires et al. 2002, Gorbenko et al. 2007]. However, as we will show in this paper, most previous related work has not properly addressed the continuity of service problem of WSC, focusing only on data integrity guarantees provided by transactional approaches. Re-executing an aborted transaction can be, however, unacceptable for some critical applications.

A commonly used technique for improving availability is to replicate services. Unfortunately, one cannot always assume or apply conventional replication techniques [Schneider 1993] for Web services published on the Internet due their degree of autonomy and heterogeneity. Consequently, we argue that such dependability mechanisms should be implemented in an upper layer into the Web compositions themselves. This paper tackles this problem by first discussing dependability requirements of WSC and by introducing a new approach that meets these requirements. The basic idea of our approach is to use ontologies to form a set of semantically alike replicas. More precisely, we propose a replication scheme where the failure of a primary service can be masked by the execution of another service semantically compatible. This replication scheme has been incorporated and implemented into the SAREK[1], a dependable web service composition framework, which is also introduced in this paper.

SAREK is made up of two main modules: the planner and the executor. The planner proposes a set of semantically similar compositions, where each proposed composition satisfies the user request (user goal). The executor is then in charge of executing a composition related to a user goal in the following way. It first randomly chooses a composition. If some Web service in this composition is unavailable, another composition can be chosen and executed. Once every Web service in the composition executes without failures, the composition reaches the given goal. To make SAREK even more dependable we propose to replicate (with a conventional primary-backup replication[Jalote 1994]) both the planner and the executor modules.

In order to validate the SAREK prototype, we have evaluated the efficiency of our replication scheme in a simulated public competition process to repair old buildings. The main motivation to apply SAREK on public competitions was the possibility to work with service compositions that would have a potential to be applied in real scenarios, since today procedures for such compositions are usually manual and require a long time period for choosing companies that fit the public work needed.

The remaining of this paper is organized as follows. Section 2 discusses and proposes a set of dependability requirements for WSC. Section 3 presents the design of SAREK and some implementation details. Section 4 shows some prototype experiments and related performance figures. Section 5 compares SAREK with related work in the light of the requirements discussed in section 2. Finally, section 6 concludes the paper and gives future directions.

[1]In the fictional Star Trek universe, Sarek is a Vulcan ambassador, and father of Spock.

2. Required Properties for Web Service Compositions

In the literature, a composition of Web services is usually divided into two main aspects: the choreography and the orchestration [Hakimpour et al. 2005, Hull and Su 2004]. The choreography deals with how Web services interact. As Web services are autonomous and each provider can develop and publish its own Web service, communication problems, such as languages and number of parameters can make difficult the process of composing Web services. Thus, choreography deals with matching problems and interface communications in compositions [Bhiri et al. 2005, Pires et al. 2002]. Orchestration deals with the whole composition, a kind of goal-oriented approach for fulfilling a user request. The orchestration can be manual or automatic. A manual orchestration means that clients should search for and compose their own compositions, taking care about the order of web services and the parameters that should be passed. In the automatic process, the client gives a goal (the user request) and the system should automatically search for and compose the Web services to reach this goal. This paper treats only orchestration for automatic compositions.

Dependability requirements in orchestration entail the continuity of the composition execution even if a Web service fails [Birman et al. 2004]. For example, even if the company A Web service is unavailable, the whole composition tries to fulfill the user request by using another composition. Another kind of problem concerns delayed responses, i.e. a partially operational web service. In this case, the service should be replaced ensuring the continuity of the composition. Another failure can be Internet disconnections: by the moment of confirming the web service execution, an outage can disrupt Internet connections. All these kinds of faults should be treated by a fault tolerant mechanism in order to reach the goal of the composition.

Many mechanisms have been introduced by the Web service community to treat failures such as FT-SOAP [Fang et al. 2007], WS-Reliability [Evans et al. 2003], WS-ReliableMessaging [Bilorusets et al. 2005] and WS-Replication [Salas et al. 2006]. Whereas these mechanisms address several reliability requirements of Web services, they cannot ensure highly available nor dependable compositions (continuity of service). Building reliable Web service compositions is much more difficulty due to the degree of autonomy and heterogeneity of Web services [Pires et al. 2002].

Taking into account the problems cited above that can happen in a composition process, we argue that a composition of Web services should respect data consistency and computational availability in the presence of Web service failures. Furthermore, such properties should be provided without compromising the scalability of compositions and transparency, two commonly required features of distributed systems [Coulouris and Dollimore 1988].

Below we further comment on the properties we believe should be respected by automatic Web service compositions.

Data Consistency. A composition of Web services should guarantee the integrity of data in its execution process. If a service fails, a data recovery mechanism takes place guaranteeing the data consistence of the whole composition.

Computation Availability. Some Web services are published by third-part enterprises. Thus, it is not possible, from a service client perspective, to assume that all services in a composition are reliable. In this case, a composition should guarantee availability without knowing the reliability level of single Web services of a composite.

Scalability. As a composition is a combination of Web services, this property guarantees the composition ability to handle a growing amount of web services. For example, a composition should ensure that it works well either with three services or with a hundred Web services.

Transparency. The composition should guarantees that a Web service is included and removed from a composition in a transparent way, making no difference in order to achieve the composition goal. For example, if a service fails or has a degraded time execution, the composition should replace this service to reach the given composition goal in a transparent way.

In the following section we describe our approach for generating automatic Web service compositions, which respect the above properties.

3. The SAREK approach

System Model and Assumptions. A composition problem involves a set of activities $a_j \in A; j \in [1..m]$, m is the number of activities, and a set of services $s_i \in S, i \in [1..n]$, n is the number of Web services. These services can be organized according to activities as communities. Thus, each community S'_j is a subset of candidate services for a specific activity a_j, $S'_j \subset S_j$. A composition C is a sequence of activities $a_j, < a_1, a_2, ...a_m >$ performed by a sequence of selected Web services $s_i, < s_1, s_2, ...s_n >$.

Web Services are implemented by processes. Thus, we assume a distributed system made of distributed processes that communicate by exchanging messages through communication channels. Channels are assumed to be reliable; they do not lose, alter, nor duplicate messages. Such channel functionality can be achieved with mechanisms such as WS-Reliability[Evans et al. 2003] or WS-ReliableMessaging[Bilorusets et al. 2005]. Processes are assumed to fail only by crashing (prematurely halting their execution).

SAREK is a modified and enhanced version of SPOC [Claro et al. 2007], where its internal architecture has been re-designed and fault-tolerant mechanisms introduced in order to attain highly available compositions. The architecture of SAREK is divided into two major modules: the Planner and the Executor. The first module, the Planner, aims to automatically determine the activities for a given composition. The Executor module executes the composition defined by the Planner activating alternative execution paths when necessary (due to failure of composite services). Both modules are replicated using a passive replication technique. If the primary module fails, a backup is voted and takes over the execution. The modules are interrelated and they communicate themselves. The output parameters of the Planner module are the input parameters for the Executor module. Figure 1 depicts SAREK with both modules and their interrelations.

The Planner execution is divided into two main phases: Planning and Optimization. At the end, the Planner module finds semantically similar compositions based on multiobjective optimizations. One of these compositions is selected by the Executor module. If, for some reasons, a composition C cannot be executed, another composition C' is selected and the Executor module tries one more time. This ensures that even if a Web service that belongs to the composition fails, SAREK does its best to execute another composition to reach the given request. The replacement of a failed Web services is done

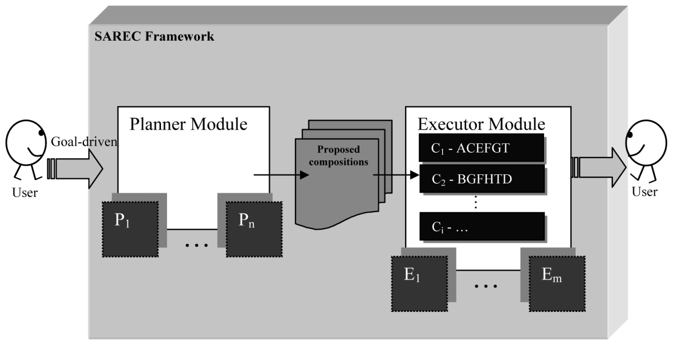

Figure 1. A general View of SAREK with the Planner and the Executor Module

transparently regarding the user request. So, such a reliability mechanism of SAREK respects the required *transparency* property.

SAREK also guarantees *data consistency* because it applies transactional approaches in each composition. Thus, if a composition is not successfully executed, the first measure is to retry the composition handling transient faults. If the problem continues and no other semantically similar composition is found the whole composition is aborted ensuring the integrity of the data.

As pointed out previously, the Planner itself should be reliable enough to ensure reliable Web service compositions. There are $n, n \geq 2$ instances of the planner module, where a $P_i, i \in [1..n]$ is voted the primary Planner module and the others will act as backups. The backup planner module (or the set of backup planner modules) starts at the same time as the primary one.

The actual execution of all backup Planners monitors P_i's execution. If P_i, for any reason, fails a new P_i is chosen and then it takes over the execution (a voting algorithm could be used in this process [Lynch 1996]). This process ensures that even if the Planner module crashes, SAREK is still able to fulfill the user request, respecting the *computational availability* property. It is important to observe that if the Planner does not accomplish its goal, the Executor module cannot even start.

The planning phase (see section 3.1) determines automatically the services that participate in a composition in runtime. The fact that our composition is only defined at runtime makes easier the addition of new Web services (this contrasts with some previous work that define the compositions in compiling time). This characteristic favors *scalability*.

The optimization phase ensures that a set of trade-off compositions are found using a multiobjective optimization algorithm (see section 3.1). The evaluation of the genetic algorithm used in the optimization process has been presented elsewhere [Claro et al. 2006]. Once P_i has executed and a set of Pareto optimal compositions[2] (semantically similar) has been proposed, the Executor module can start its execution. The Executor module also uses the primary-backup replication scheme just described to the

[2]Pareto optimal solutions or non-dominated solutions are a set of solutions where a given solution does not dominate another one and vice-versa. They are used in multiobjective optimizations where the notion of optimal solutions is based on Pareto's relation of dominance.

Planner module.

3.1. Planner Module

As previously state, the Planner module is made up of two main phases: the Planning phase and the Optimization phase. The Planning phase interacts with OPS (Ontology to Publish Services), an ontology repository to discover Web services. OPS is an OWL ontology which describes each Web service in an OWL-S format. The matching algorithm used between the planning algorithm and the concepts in OPS was a simple hierarchical method. This ontology has the domain description that SAREK deals with (i.e., public competition process). The planning phase can interact many times with this ontology so as to find new Web services for the composition. This phase aims to determine the activities that will be composed to fulfill the user request. Each activity corresponds to an action in the planning algorithm. A classical planner based on preconditions and effects was used. If a service has not ever been included in a plan, SAREK should find this service in the OPS and add it to the plan and continue planning until the goal matches the action effects. Thus, this phase determines the set of activities A that can reach the given request.

The Optimization phase optimizes the combination of Web services s_i and activities a_j. Thus, this phase determines which Web service performs which activity. The values used to optimize the composition are based on the estimated values retrieved from each candidate Web service $s_i \in S', \forall a \in A$. This phase is like a quotation system that retrieves all estimated values of the candidate Web services and optimizes these values producing a set of trade-off (semantically similar) compositions. The set of compositions is produced because of the presence of more than one objective to optimize. For example, consider these two objectives: minimize cost and maximize service reputation, both are contradictory.

SAREK uses a genetic algorithm called NSGA-II [Claro et al. 2007] to solve this optimization problem. Despite the fact that each solution fulfills the user goal, each one has a different set of estimated values. Among these Pareto[3] solutions, the Executor module will randomly choose initially only one of them to start the execution.

3.2. Executor Module

The Executor module executes a composition of Web services based on prefixes. Each composition proposed by the Planner is represented in a OWL-S file, and, during execution, the Executor calls each Web service that appears in the running composition.

In order to provide fault tolerance both a transactional and a replication mechanisms are applied, which are described below.

Transactional level. A transaction technique is used to guarantee data consistency in case of a composition failure. Using a temporal redundancy mechanism, SAREK tries the same composition one more time to recover from a possible transient fault. If the problem continues, before rolling back the transaction, SAREK chooses another composition among those proposed by the Planner module.

[3]We use all over this paper the words: *trade-off*, *Pareto* and *semantically similar* as synonyms. In fact, all of them describe the solutions (individuals) retrieved from the NSGA-II algorithm.

Semantic replication scheme. During execution of a composition, faulty Web services can be replaced by a semantically similar services ensuring the *transparency* property. We call such a redundancy scheme semantic replication. We argue that this is a kind of spatial redundancy because there is a set of compositions that achieve the same goal. This scheme uses a prefix approach so as to increase performance when re-executing a partially failed composition. The prefix algorithm works as follows. Assume the running composition is defined as $< s_1, s_2, s_3, ..., s_n >$ and that this composition fails because of the failure of s_3, but services s_1 and s_2 were run correctly. In our example, $< s_1, s_2, s_3$, altogether form a composition. In such a composition, s_1 and s_2 must be executed successfully before the execution of s_3. Observe that the execution of s_3 depends on the results produced by s_1 and s_2. For example, if s_2 is a company that provides wood, s_3 has to be a company specialized in building wooden staircases. In other words, s_1 and s_2 are preconditions for s_3. Thus, in order to avoid re-executing s_1 and s_2, the prefix algorithm searches another composition that starts with such the prefix $< s_1, s_2 >$, saving recovery time.

If all these fault tolerant mechanisms of SAREK fail in executing a composition to termination, an error is shown to the user informing that the execution of the Web service composition was not possible.

3.3. Experimental Tests and Its Performance Evaluation

As a proof of concept, SAREK has been applied to a scenario where public competition processes are carried out for repairing public buildings. Several prototype experiments have been conducted and data performance collected for executing compositions where single Web Services were forced to fail. Before proceeding to show the experimental data, we explain in more details the application scenario.

Case Study - Public Building Competition Process. The competition process for restoring public buildings starts with a request for restoration. Based on this request, an architect with a state agent will determine the work that should be done on the buildings. This work is grouped into categories based on activities. The competition process will be then organized by work's category. An enterprise can be characterized as a general enterprise, which executes many specialized kinds of work or a specialist one, which does only one work at a time. In our first experimentation we will only consider enterprises that do only a specific work. The architect will also define an order for the activities of a work. Once the work plan is determined, the enterprises can send (also via email) their propositions with the estimated cost.

Once propositions have been received, the state agent needs to analyze them, one by one, based on their costs, duration of work, enterprise's turnover and reputation in order to find a good combination between enterprise and work. Such a multicriteria analysis will lead the state agent to decide which enterprise will execute which task and the Competition Process will be terminated. If many enterprises are candidates to execute some work, this task can be both time and effort consuming.

Applying SAREK to this case study, activities of a restoration work can be seen as activites a_j and enterprises as Web services s_i, and the whole execution is divided into two main phases: composition planning and execution, which are conducted by the Planner module and Executor module, respectively.

In the Planner module, Web services are discovered, some quotes retrieved in order to estimate values and, subsequently, an optimization step tries to optimize the values. The Executor module finds and executes a composition among those proposed by the Planner, and chooses other composition in case of composition failures, doing its best to ensure that the state agent will receive a confirmation to do the restoration.

Though SAREK produces two kinds of results (a set of trade-off compositions at the end of the Planner module, and the execution of a composition at the end of the Executor module), this paper evaluates only the actions of the Executor module. An evaluation of the planning algorithm used in the Planner module is given elsewhere[Claro et al. 2006].

Experiments. A prototype implementation of SAREK was developed using Java version 1.5, and other technologies such as MySQL Database 4.1, Apache Tomcat 5.0, Axis 1.3, Jena API 2.3, OWL-S API 1.1.0.

In order to evaluate SAREK the public competition process just described has been simulated in a series of experiments. The experiments were carried out in a computer with Intel motherboard Core Duo, processor T2300 1.66 Ghz and 1Gb of RAM. In the experiments, we do not make network connections because we would like to evaluate the semantic replication scheme without the overhead of connections on the Internet. Thus, all the Web services are located in the same machine. Our simulated evaluation scenario has four activities: *supply wood*, *supply concrete*, *supply iron* and *build staircase*. Each activity can be performed by two candidate companies. We assumed that the preconditions to build a staircase were to *supply concrete* and other material such as *wood* or *iron*. The graph in Figure 2 is an example of a possible activity arrangement.

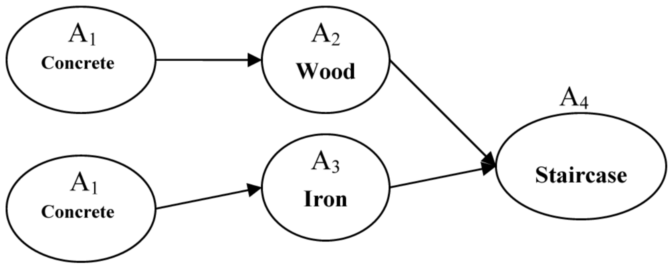

Figure 2. An example of the building repair scenario

The number of proposed compositions was limited by the total amount of money a client wants to spend in the whole composition process. The Web service WSDL operation (*String executeWS()*) is responsible to execute the Web service. If the execution runs correctly (without failures), the operation returns an 'OK' signal. We opted to make this method as simple as possible in order to measure the framework overhead. Below are presented the results produced by two runs of SAREK for the repair building scenario; the first one without failures and the second one with failures.

The first two lines of Figure 3 present the output of the Planner, indicating that the companies 4, 2 and 6 were selected (first line) to execute the activities in the order indicated in line 2 (*SupplyConcrete*, *supplyIron*, *buildStaircase*). As shown in the figure, the three activities executed without failures.

```
**********************************IDProcess:200704
CompanyOrderExecution: 4;2;6;
WorkOrderExecution: supplyConcrete;supplyIron;buildStaircase;
WorkExecution*OK*: supplyConcrete;CompanyExecutionOK: 4
WorkExecution*OK*: supplyIron;CompanyExecutionOK: 2
WorkExecution*OK*: buildStaircase;CompanyExecutionOK: 6
*********************************
```

Figure 3. A composition example without failures

In figure 4 is presented an example with failures, and the output shows which service failed and which new Web services were chosen to be executed the next time. The example shows the recovery actions caused by three faulty services, illustrating the application of the prefix algorithm of the semantic replication scheme.

```
********************************** IDProcess:200704
CompanyOrderExecution: 5;3;1;
WorkOrderExecution: supplyConcrete;supplyIron;buildStaircase;
*FAULT* WorkExecution: supplyConcrete;*FAULT*CompanyExecution: 5
CompanyOrderExecution: 4;3;6;
WorkOrderExecution: supplyConcrete;supplyIron;buildStaircase;
WorkExecution*OK*: supplyConcrete;CompanyExecutionOK: 4
*FAULT* WorkExecution: supplyIron;*FAULT*CompanyExecution: 3
CompanyOrderExecution: 4;2;6;
WorkOrderExecution: supplyConcrete;supplyIron;buildStaircase;
WorkExecution*OK*: supplyIron;CompanyExecutionOK: 2
*FAULT* WorkExecution:buildStaircase;*FAULT*CompanyExecution: 6
CompanyOrderExecution: 4;2;1;
WorkOrderExecution: supplyConcrete;supplyIron;buildStaircase;
WorkExecution*OK*: buildStaircase;CompanyExecutionOK: 1
**********************************
```

Figure 4. A composition example with 3 faulty Web services

We forced three failures by crash: service 5, service 3 and service 6. The first composition chosen was $5;3;1;$. However, as the service number 5 failed, SAREK tried to find among the semantically similar compositions another composition that performs the same task. Thus, the composition $4;3;6;$ was chosen. The service 4 that performs *supplyConcrete* was correctly executed but the next Web service (number 3) failed. As a consequence, another composition with the same prefix 4 should be chosen and another Web service should perform the *supplyIron* task. Now the composition $4;2;6;$ should be executed, but service number 6 also fails and another composition with the prefix $4;2;$ should be found. Finally, the successfully executed composition was $4;2;1$.

Evaluating the Performance of the Semantic Replication. Two kinds of experiments were carried out to evaluate the performance of the semantic replications scheme. Each experiment was run 400 times for calculating the average time and related standard deviation (SD). In the first kind, a composition is fixed and re-executed 400 times. In the second kind, a new randomly chosen composition is executed for each of the 400 iterations. In both kinds of experiments, the compositions were run with and without failures. In table 1 the collected figures are summarized, and explained in the following.

400 runs	Without Faults	SD	With Faults	SD
Fixed composition	118	48	185	59
Random composition	117	49	155	73

Table 1. Performance Evaluation of Executor Module (in miliseconds)

In the first experiment, for fixed composition without failures, we obtained an average of 118 milliseconds to execute the composition. In the next experiment, we ran the same fixed composition but we forced a Web service to fail by crash. Thus, the prefix algorithm in this case should find another composition with the same prefix to execute. The average time in order to recover from this failure and execute the whole composition was 185 milliseconds.

A second set of experiments was done using random choice of the composition. Without failures, there is almost no overhead on randomly choosing a composition (117 milliseconds). However, considering that a faulty service was forced inside a composition, the composition randomly chosen undertakes about 155 milliseconds. In this case, in some times, the random choice does not lead to a faulty composition (since the failed Web service was not present in such a composition), thus the execution time of the composition decreases, so increasing the performance.

Figure 5 depicts the overhead caused by faults for an increasing number of forced Web service failures. As expected, time increases as the number of faulty services increases because other compositions should be found in order to correctly terminate a composition

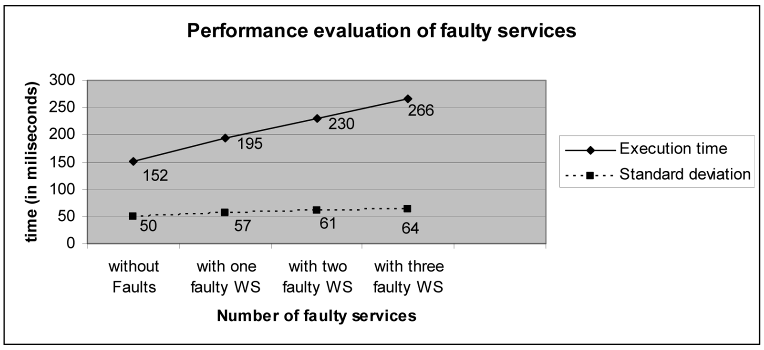

Figure 5. The Evaluation of Executor Module in Face of Multiple Web Service Failures

The Planner module in SAREK has some interactions with the user. Some dialog boxes are shown to the user to type input values (e.g., stair dimensions and maximum cost for a composition). Likewise, in the Optimization phase, output values are written into files. Due to such I/O interactions, in our experiments, the whole framework takes about 26 seconds to execute. However, the time shown in table 1, related to the execution of the Executor Module, is not affected by the above mentioned I/O interactions.

The following section discusses the existing fault tolerant mechanisms dealing with Web services and their limitations, regarding the proposed properties.

4. Comparison with Related Work

Several researchers have proposed extensions to Web services architecture in order to enhance dependable aspects in Web service compositions. These researches are discussed below in the light of required properties presented in section 2. As far as we know, none of the published papers has put together these 4 properties in an evaluation framework. However, a careful reading of the literature reveals that each of the properties appears isolated or combined with one or two other properties. So, our contribution in this matter is much on compiling the set we believe is the most significant to dependable Web service compositions.

The authors in [Pires et al. 2002] propose a multilayered architecture called WebTransact. The WebTransact treats each service that belongs to a composition as a transactional unit. The authors propose a WSTL (Web Service Transaction Language) located upon WSDL to model compositions as composite tasks (activities). The WebTransact lower level provides mediator services that integrate semantic similar remote services (Web services). Concerning the set of properties proposed, WebTransact deals with *data consistency*. It does not address high availability (*computational availability*) since no redundancy mechanism is proposed. Finally, it does not handle *Transparency* property because each Web service included into the composition should have the WSTL features described in the WSDL file. Since the number of mediators can increase, the *scalability* property can be guaranteed.

The authors in [Mikalsen et al. 2002] propose a solution for introducing transactional reliability into Web services. As Web services are autonomous, incompatible transaction models may be involved in the same composition. Thus, they propose transactional attitudes to providers so as to explicitly describe their specific transactional semantics and to clients for describing their expectations. A middleware acts as an intermediary between the client and the provider. However, this framework does not ensure certain properties such as: *transparency* because transactional attitudes are included into both the client and the provider; *computational availability* because no replication mechanism is supported; and no *scalability* because the middleware (a web service) receives all incoming request and should complete the transaction in case of failure. However, as all the previous transactional approaches, this work handles *data consistency*.

In [Gorbenko et al. 2007] a mechanism for forwarding error recovery by using exception handling in a composition is proposed. They propose a transactional solution in the composition level in order to handle undependable Web services (*transparency*). The *data consistency* property is ensured because if a single service inside the composition fails and cannot be recovered, the whole composition aborts its execution. *Scalability* property is achieved because each composite Web service is aggregated into dedicated servers, thus WS compositions (WS components in their terminology) are distributed over a network. This approach is similar to the transactional level of SAREK as explained in section 3.2. However, differently from SAREK, no *computation availability* mechanism is proposed.

If only transactional approaches are used, even with a compensable technique that reduces recovery time, they might spend more time than the acceptable to fulfill the user goal. Few researchers propose replication mechanisms for treating dependable Web service composition requirements. A replication mechanism can shorten the recovery time and can increase the system availability (*computation availability*)[Chan et al. 2006].

The work in [Majithia et al. 2004] is similar to our proposition in the sense that they also use a goal-oriented framework, using composition graphs, where in case of service unavailability, other composition graphs are built to take over the execution. This framework deals with almost all proposed properties except data consistency because it does not have any transactional mechanism. Thus, if some faults occur in the *abstract* module or in the *concrete* module, no recovery mechanism is used. In contrast, our approach has a redundancy mechanism in both modules: the Planner and the Execution. Moreover, SAREK can be more efficient in terms of response time due to its prefix mechanism in case of failures, whereas this framework builds new composition graphs in case of service failures, not reusing the services that were executed correctly.

We argue that in order to have a highly available composition, the four properties previously enumerated should be respected, and our approach meets such requirements by combining a transactional mechanism with a semantic replication scheme. A further advantage of our approach is due to the fact that by applying a goal-oriented replication, we are able to tolerate faults not only originated from computing environment (such as energy outage), but also faults originated by design[Jalote 1994]. The table 2 summarizes these approaches based on the minimal set of properties previously explained.

Related Work	**DC**	**CA**	**S**	**T**
WebTransact[Pires et al. 2002]	Yes	No	Yes	No
WSTx Framework[Mikalsen et al. 2002]	Yes	No	No	No
DeWs [Gorbenko et al. 2007]	Yes	No	Yes	Yes
Semantic Grid Framework[Majithia et al. 2004]	No	Yes	Yes	Yes
SAREK Framework	Yes	Yes	Yes	Yes

Table 2. Comparison of WSC approaches. DC=Data Consistency; CA=Computation Availability; S=Scalability; T=Transparency;NA=Not Applied

5. Conclusion

This paper discussed the challenges for achieving Dependable Web service compositions, suggesting a set of required dependability properties (data consistency, computation availability, scalability and transparency) to be fulfilled by compositions. In the light of such properties, existing work is discussed and it is concluded that most of the proposals lack the adequate support for availability. Moreover, in this paper, we propose a new framework called SAREK that satisfies the pointed properties by combining together a transactional and a semantic replication scheme. SAREK is divided into two modules: the Planner and the Executor. In the Executor module, SAREK uses a technique based on prefix to shorten recovery time, and both modules are replicated with a primary-backup scheme. To the best of our knowledge, SAREK is the first framework that provides such fault-tolerant guarantees in service compositions. In future work we will evaluate the fault-tolerant mechanisms presented by measurement analysis in real scenarios.

6. Acknowledgments

Dr. Daniela Barreiro Claro is supported by FAPESB(BOL2071/2006) and Prof. Raimundo José de Araújo Macêdo is supported by FAPESB and CNPQ (Edital Univer-

sal).

References

Aggarwal, R., Verma, K., Miller, J., and Milnor, W. (2004). Constraint driven web service composition in meteor-s. In *IEEE SCC 2004*, pages 23–30.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S. (2003). Business process execution language for web services version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

Bhiri, S., Godart, C., and Perrin, O. (2005). Reliable web services composition using a transactional approach. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service*, Hong Kong.

Bilorusets, R., Box, D., Cabrera, L. F., Davis, D., Ferguson, D., Ferris, C., Freund, T., Hondo, M., Ibbotson, J., Jin, L., Kaler, C., Langworthy, D., Lewis, A., Limprecht, R., Lucco, S., Mullen, D., Nadalin, A., Nottingham, M., Orchard, D., Roots, J., Samdarshi, S., Shewchuk, J., and Storey, T. (2005). *Web Services Reliable Messaging Protocol (WS-ReliableMessaging).*

Birman, K., van Renesse, R., and Vogels, W. (2004). Adding high availability and automatic behavior to web services. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pages 17–26.

Chan, P. P. W., Lyu, M. R., and Malek, M. (2006). Making services fault tolerant. In *the Third International Service Availability Symposium (ISAS 2006)*, volume 4328, pages 43–61, Berlin, Germany. Springer-Verlag(LNCS).

Claro, D. B., Albers, P., and Hao, J. (2006). *Semantic Web Services, Processes and Applications*, chapter 8, pages 205–234. Springer Publisher.

Claro, D. B., Albers, P., and Hao, J. (2007). A framework for automatic composition of rfq web services. In *IEEE Proceedings of the First Workshop on Web Service Composition and Adaptation (WSCA) held in conjunction with International Conference of Web Services (ICWS'07)*, pages 221–228, Salt Lake City, USA. IEEE SCW 2007.

Coulouris, G. F. and Dollimore, J. (1988). *Distributed systems: concepts and design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Evans, C., Chappell, D., Bunting, D., Tharakan, G., Shimamura, H., Durant, J., Mischkinsky, J., Nihei, K., Iwasa, K., Chapman, M., Shimamura, M., Kassem, N., Yamamoto, N., Kunisetty, S., Hashimoto, T., Rutt, T., and Nomura, Y. (2003). *Web Services Reliabiliy (WS-Reliability).*

Fang, C.-L., Liang, D., Lin, F., and Lin, C.-C. (2007). Fault tolerant web services. *J. Syst. Archit.*, 53(1):21–38.

Gorbenko, A., Kharchenko, V., and Romanovsky, A. (2007). On composing dependable web services using undependable web components. *Int. J. Simulation and Process Modelling*, 3(1/2):45–54.

Hakimpour, F., Sell, D., Cabral, L., Domingue, J., and Motta, E. (2005). Semantic web service composition in irs-iii: The structured approach. In *CEC '05: Proceedings* *of the Seventh IEEE International Conference on E-Commerce Technology (CEC'05)*, pages 484–487, Washington, DC, USA. IEEE Computer Society.

Hull, R. and Su, J. (2004). Tools for design of composite web services. In *SIGMOD 04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 958–961, New York, NY, USA. ACM Press.

Jalote, P. (1994). *Fault Tolerance in Distributed Systems*. Prentice Hall PTR; US Ed edition, USA.

Lynch, N. A. (1996). *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA.

Majithia, S., Walker, D. W., and Gray, W. A. (2004). Automated composition of semantic grid services. In *AHM - e-Science All Hands Meeting*.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Parsia, M. P. B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004). Owl-s: Semantic markup for web services. http://www.daml.org/services/owl-s/1.0/owl-s.html.

Martinez, E. and Lesperance, Y. (2004). Web service composition as a planning task: Experiments using knowledge-based planning. In *Workshop on Planning and Scheduling for Web and Grid Services held in Conjuction with the 14th ICAPS*, British Columbia,Canada.

Mikalsen, T., Tai, S., and Rouvellou, I. (2002). Transactional attitudes: Reliable composition of autonomous web services. In *Proceedings of the Workshop on Dependable Middleware-Based Systems in conjunction with IEEE International Conference on Dependable Systems and Networks (DSN'02)*.

Pires, P., Benevides, M., and Mattoso, M. (2002). Building reliable web service composition. In *NODE'2002 Web and Database - related Workshops on Web, Web-Services and Databases Systems*, volume 2593, pages 59–72, London,UK. LNCS.

Rajasekaran, P., Miller, J., Verma, K., and Sheth, A. (2004). Enhancing web services description and discovery to facilitate composition. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC) held in conjunction with ICWS'2004*, pages 55–68.

Salas, J., Pérez-Sorrosal, F., Patino-Martinez, M., and Jimenez-Peris, R. (2006). Ws-replication: A framework for highly available web services. In *WWW'2006 - International World Wide Web Conference*, pages 357–366, Edinburgh, Scotland. ACM.

Schneider, F. B. (1993). *Replication management using the state-machine approach*, chapter 7, pages 169–197. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Ugur, K., Sirin, E., Nau, D., Parsia, B., and Hendler, J. (2004). Information gathering during planning for web service composition. In *Workshop on Planning and Scheduling for Web and Grid Services held in Conjuction with the 14th ICAPS*, British Columbia,Canada.