論文 / 著書情報
Article / Book Information

Title	A Failure-Aware Model for Estimating and Analyzing the Efficiency of Web Services Compositions
Author	Neila BEN LAKHAL, Takashi Kobayashi, Haruo YOKOTA
Journal/Book name	Proc. of IEEE 11th Intl Symposium on Pacific Rim Dependable Computing (PRDC2005), , , pp. 114-121
Issue date	2005, 12
DOI	http://dx.doi.org/10.1109/PRDC.2005.6
URL	http://www.ieee.org/index.html
Copyright	(c)2005 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

A Failure-Aware Model for Estimating and Analyzing the Efficiency of Web Services Compositions

Neila Ben Lakhal[†], Takashi Kobayashi[‡] and Haruo Yokota[†‡]
[†]Tokyo Institute of Technology, Department of Computer Science
[‡]Tokyo Institute of Technology, Global Scientific Information and Computing Center
2-12-1 Oh-Okayama, Meguro-ku Tokyo, 152-8552 Japan
Phone: +81-3-5734-3505 , Fax: +81-3-5734-3504
Email: neila@de.cs.titech.ac.jp, {tkobaya, yokota}@cs.titech.ac.jp

Abstract

More and more within the last couple of years, there is a recognition that the Web services composition concept will constitute a major breakthrough and will revolutionize the way we deal with integrating disparate and distributed computing environments. Yet, to rise to such a position, a chief concern is to guarantee a high-dependability level of the Web services compositions, which is significantly critical, specially in view of the particularities of the Web services environment(unpredictability, heterogeneity, autonomy), if confronted with other computing environments.

Our present work falls within this context since we tackle the problem of QoS (Quality of Service) in the Web services context by verifying to what extent fault-tolerant and dynamically-executed Web services compositions are efficiently serving their purposes. And in pursuing this goal, we introduce a novel model that characterizes, estimates and analyzes several QoS properties of dynamically-executed fault-tolerant Web services compositions–namely the reliability and the execution time.

Our model allows acquiring more accurate estimations since it confers a paramount importance to the repercussions of failures. In addition, contrary to other QoS estimations models in the Web services context, which use QoS estimations published in UDDIs by the Web Services owners/providers, our model computes QoS estimations on the base of the compositions execution observations, where the observation results are collected in a history.

Finally, since Web services are stateless, tracking the failures and determining their locations is almost impossible. To overcome this limitation, we propose to attach to each of the composition's component a state. In doing so, obtained estimations can contribute in acquiring more accurate information about the failures locations and can be used later to improve the composition QoS in the future.

1. Introduction

Over the last couple of years, the Web services architecture has gained a rapid uptake as it embodies the concept of full-interoperability. Its key enablers are a set of developer-friendly standards (e.g., SOAP, UDDI, WSDL) which do not involve a whole learning curve as it is for other distributed architectures.

However, Web services (WSs), if they are individually considered, they tend to be limited in their offered capabilities. Thereby, the requirement to bring them together, in a consistent manner, to create more elaborated functionalities, in the form of Web services compositions (WSCs), has been unveiled.

To date, a significant amount of research efforts has been reported in the field of WSCs. Specifically, there are many approaches dealing with assembling basic WSs into composite ones[1][2][3]. In addition, there are already various platforms that are compliant with the WSs architecture (e.g.,.NET, J2EE, AXIS). Besides, the emergence of many popular specifications: BPEL4WS, BPML, DAMLS to name but a few, has been widely noticed as well.

Yet the WSC concept is still young; and to rise to the opportunity of becoming the de facto solution for disparate and distributed computing environments integration, defining solely the behavior of the composed services and dealing only with their coordination is insufficient. There are clearly a number of other significant challenges to consider. In particular, due to the specificity of the WSs architecture (i.e. dynamism, unpredictability, heterogeneity, autonomy) and to the obvious limitations of the Internet, any composition has to satisfy a high-dependability level that requires defining vigorous fault-tolerance mechanisms.

So far, in our previous work[3][4], we have addressed one of the aforementioned challenges that relates to the specification and execution of fault-tolerant WSCs by defining proper failure information and handling mechanisms.

This paper tackles the problem from a different angle by verifying to what extent fault-tolerant WSCs are efficiently serving their purposes, upon execution, by assessing their QoS properties. In pursuing this goal, we introduce a novel model that characterizes, estimates and analyzes several QoS properties of dynamically-executed faut-tolerant WSCs, namely the *reliability* and the *execution time*.

Contrary to most of the current approaches dealing with QoS estimations in the WSs context, which rely on the QoS information advertised by the WSs owners/providers, our model computes QoS estimations on the base of the WSCs executions observations, where the observation results are collected in a history. In doing so, more accurate estimations can be acquired since we do not rely on the providers data which might be not up to date, or subject to manipulation by the providers.

Up to now, proposed approaches that addressed the estimation of the QoS, they make use of either mathematical modeling approaches or simulation tools[5][6][7][2]: they typically provide a global idea about the variation range of the estimations of certain properties of the composition as a whole, or their estimations are only applicable for static WSCs. Yet, providing more detailed estimations, specially in the case of complex WSCs, is more and more called for. To fulfill this requirement, our model is oriented towards acquiring more practical, more detailed estimations of the QoS of each component apart, and derives the equivalent estimations for the WSC.

In addition, major part of the work done up to now considers only situations where the WSC do not fail. In doing so, the obtained estimations are very often regarded as too optimistic since they do not account for any potential failures (information, recovery) and their repercussions.

We propose in our model to account for failures and their repercussions on the effective performances of the WSCs since this is especially required in the WSs architecture, in view of the WSs inherent tendency to fail rather easily (relatively to other computing components). Typical failures causes include: incompliant WSs characteristics (e.g.,transactional supports, management policies, access rights), Internet obvious limitations (e.g., latency, timeout, security), and so forth. Moreover, since WSs are stateless, tracking the failures and determining their locations is almost impossible. To overcome this limitation, we propose to attach to each of the WSC's components a *State*[3].

By introducing the *State(S)* (e.g., active, executing, done, failed), the obtained estimations are awaited to contribute in acquiring more accurate information about the failures locations and to be used to later to improve the WSCs QoS.

In characterizing the execution time, our model introduces the *Optimistic Execution Time* and *Probable Execution Time* where the former is only limited to the correct execution situations and where the latter considers all the possible execution situations (i.e., correct execution, faulty execution, recoverable execution and none recoverable execution) of a fault-tolerant WSCs. Distinguishing between these two variants allows us to provide more accurate estimations where the failure repercussions on the delivered performances are accounted for.

As for the reliability, we introduce the *Reliability Tendency(RT)* concept, which builds mainly on the concepts of component's *State(S)*, component's *State Reliability Contribution (SRC)*, component's *Terminal State Set (TSS)* and *State Tendency Set(STS)*. *SRC*, *TSS* and *STS* are obtained on the base of historical data collected from the past executions of the WSCs.

The reminder of this paper is as follows. Section 2 introduces our novel model and describes mainly how we characterize the reliability and execution time properties. Section 3 validates our model on the base of a case study. Section 4 discusses some related work. Lastly, section 5 concludes our paper and gives a few tentative future directions.

2. Web services compositions QoS modeling and analysis

2.1. Composition specification model

We depict a WSC as an orchestration of n components C_i, with i in $[1..n]$. The different components can be assembled according to three basic aggregation patterns – namely the sequence, the parallel split and the exclusive choice. Of course to express the underpinning logic of any real process, these three patterns are far from being enough, we will introduce advanced patterns in our future work.

2.2. Composition execution model

We consider a dynamic execution model of WSCs where every component C_i, from a WSC (noted $Composition_c$) can be mapped at runtime to any available Web service (noted ws_{ip}), where $ws_{ip} \in \mathbb{S}(C_i)$ and $p \in [1..card(\mathbb{S}(C_i))]$ with:

- $\mathbb{S}(C_i)$ is the set of all the available WSs that provide functionalities satisfying what C_i requires;
- $card(\mathbb{S}(C_i)) = |\mathbb{S}_i|$ is the cardinality of $\mathbb{S}(C_i)$. It is the number of WSs that might be mapped to C_i on its execution. Those WSs are hosted by the providers;
- Any C_i can be mapped at runtime to more than one WS, at most the cardinality of $\mathbb{S}(C_i)$. We assume that C_i was invoked for execution by q WSs from $\mathbb{S}(C_i)$, $(q \leq |\mathbb{S}_i|)$. Among those invocations, we assume that $(q-1)$ executions were finished with failures. Indeed, C_i was retried $q \ times$ and the q^{th} execution delegated to the WS ws_{iq} was successful;

- We define ws_{iq}, with q a particular value of p, the WS which was mapped to C_i and terminated successfully its execution. Note the following particular cases:
  - ($q = 0$ and $|\mathbb{S}_i| \neq 0$) if C_i was not mapped to any WS (i.e., if the execution of any previous component fails, then C_i might not be attempted at all);
  - ($q = |\mathbb{S}_i| = 0$) if there is no WS that can be mapped to component C_i;
  - If the q^{th} invocation of C_i, delegated to ws_{iq}, was successful, then the $(q-1)^{th}$ previous executions of C_i with other WSs (ws_{ik}, with k in $[0..(q-1)]$) were finished with failures.

It is worth noting here that the way the WSCs are executed (i.e., how the WSs are chosen, how they are mapped to components, how the execution goes ahead, how the messages are exchanged e.g., via a central entity, in a peer to peer) and the likes issues are beyond the scope of this paper. We assume that we are dealing with WSCs for which, proper reliability enhancement mechanisms were defined. In fact, many are the approaches which dealt with this issue, for generality sake, we are not following any specific mode of execution or specification.

2.3. Web services discovery and selection model

We are dealing with dynamically-executed WSCs: the mapped WSs to the components are chosen at runtime. Many dynamic WSCs approaches were proposed (e.g., CMI[8], the eFlow Platform[9] and Self-Serv[1] framework). In what follows, we assume that the applied approach to discover the WSs does not matter. For each component C_i, the discovered WSs are ranked in a $\mathbb{S}_i$. The selection is done in the order of ranking.

2.4. Composition failure model

Fault-tolerance is the ability of a system to behave in a well-defined manner once faults occur. When considering a fault tolerant system, a first prerequisite is to specify the fault class that should be tolerated. Traditionally, with respect to the behavior of a failed system, this was done by naming one of the standard fault models. Well-known examples are the fail-stop (the system is one that, once failed, does not output any data), Byzantine (the system does not stop once it fails, and instead returns wrong information), or fail-fast (the system is one that achieves fail-stop behavior very soon after failing, it behaves in a Byzantine way for only a a short amount of time)[10][11][12].

Since the failure model is application specific, the WSCs obey originally to the Byzantine model, however, because of the WSs architecture complexity, we assume that any WSC, once invoked, it might be subject to failures which could be environment-dependent failures (at the underlying platform e.g., crashes, timeouts) or application-specific failures (e.g., erroneous input, exceptions).

We limit our study only to transient failures where the system can recover from automatically, either through performing a forward recovery or a backward recovery and this without requiring any external entity (human) intervention.

As for the backward recovery mechanisms, we assume that the rolling back the terminated components and the aborting the executing components apply. For the forward recovery, in our proposal, we assume that the component C_i is retried/reattempted by an alternative WS chosen from its $\mathbb{S}(C_i)$. Finally, we assume that any execution is subject to only one failure at a time. Regarding the failure detection, we assume that a certain period of time for each component is defined. This period of time, if it is elapsed without receiving any notification about the considered component execution progress then, a failure is deduced and a failure recovery is triggered.

2.5. Execution time property characterization

We propose to estimate the execution time of each component apart, and then to derive the equivalent estimations for the whole composition.

2.5.1. Execution time of a component: [2][5] defined the execution time taken by a single WS invocation with three constituents: *(i)* service time S(WS) necessary to perform the WS task, *(ii)*message delay time M(WS) taken by the SOAP messages in being sent/received and *(iii)* waiting time W(WS) the WS invocation delay. Yet considering only these constituents in characterizing the execution time is not enough since this definition considers only the case of one-to-one WS-component mapping. Besides, it is rather optimistic since it doesn't take into account eventual failures.

DEFINITION.1: We define the *Optimistic Execution Time* of a component C_i from a WSC $Composition_c$ (noted $T(C_i)_{opt}$) the execution time of the *dynamically-mapped* WS at runtime to C_i. This definition considers only the best case where C_i is mapped to a WS which succeeds in its execution. Inspired from [2][5], we define $T(C_i)_{opt}$ by Equation(2.1), with $T(C_i, ws_{iq})$ is the time to execute ws_{iq} ranked in position q in $\mathbb{S}(C_i)$. $T(C_i)_{opt}$ has three constituents: *Service Time* ($S(ws_{iq})$), *Message Delay Time* ($M(ws_{iq})$), and *Waiting Time* ($W(ws_{iq})$).

$$T(C_i)_{opt} = S(ws_{iq}) + M(ws_{iq}) + W(ws_{iq}) \quad \text{with: } 1 \leq q \leq |\mathbb{S}_i| \tag{2.1}$$

DEFINITION.2: We define the *Probable Execution Time*(noted $T(C_i)_{prob}$), as the estimation of the execution time of C_i (see Equation(2.2)). $T(C_i)_{prob}$ takes into account the allocated WS failure repercussions: the time to inform about the failure and to recover from it.

$$T(C_i)_{prob} = T(C_i)_{opt} + I(C_i) + R(C_i) \tag{2.2}$$

DEFINITION.3: We define the *Failure Information Time* (noted $I(C_i)$), as the time taken by the different SOAP messages in being sent/received as notifications of a certain failure. Since any component C_i might be subject to as many failures as the number of times it was reattempted, the entity $I(C_i)$), defined by Equation(2.3), is the sum of all the elapsed periods of time to notify about each of the allocated WSs failures. Depending from the mode of executing the WSCs, theses messages might be exchanged directly between peer WSs or components, or communicated to a central authority responsible for failures handling.

$$I(C_i) = \sum_{k=1}^{q-1} I(C_i, ws_{ik}) \quad (2.3)$$

DEFINITION.4: We define the *Failure Recovery Time*(noted $R(C_i)$) as the required time to recover from a particular component C_i failure (from the failure of the allocated Web service to the component C_i).

$$R(C_i) = For(C_i) \vee Back(C_i) \quad (2.4)$$

with: $For(C_i) = \sum_{k=1}^{q-1} T(C_i, ws_{ik})$

$Back(C_i) = xor(Roll(C_i), Comp(C_i), Abort(C_i))$

As defined in Equation(2.4), $R(C_i)$ can be of two kinds:

- $For(C_i)$: the total time spent in retrying C_i by other WSs. In other words, in Equation(2.5), C_i succeeds when it is invoked in the q^{th} execution when it is allocated to ws_{iq}. Thus the previously allocated $(q-1)$ WSs failed and the sum of their respective execution time is the *Forward Recovery Time* of $C_i(For(C_i))$;
- $Back(C_i)$: In a backward recovery, the mechanism to apply depends from the composition specification model, the more widely used techniques are rolling-back, aborting and compensation. Other methods which can be similarly introduced. Here, the entity $Back(C_i)$ is either equal to the *Rollback time*($Roll(C_i)$), the *Compensation time*($Comp(C_i)$), or to the *Aborting time*($Abort(C_i)$).

2.5.2. Component initial execution time estimation : It is most likely that the allocated WS failure makes the execution time tend to infinity, which is not acceptable. To avoid that, we define a *Waiting Period* (noted θ) that will avoid waiting eternally for an answer, from a particular WS, that might never come, if the WS fails to respond. After θ is elapsed and no information was received, the WS is considered as failed and a recovery needs to be performed. The question here is how to determine θ, in case the component was not yet attempted? Usually, WSs advertise their processing time or provide methods to enquire about it. This could be used here as for an initial estimation of θ. Later, when the component is invoked a number of times, θ can be estimated on the base of these past invocations(m).

In Equation(2.5)), $T(C_i)^1_{opt}$ is the *Optimistic Execution Time* of C_i when invoked for the 1^{st} time:

$$\theta_i = \max(T(C_i)^1_{opt}, T(C_i)^2_{opt}, ..., T(C_i)^m_{opt}) \quad (2.5)$$

2.5.3. Execution time dimension of a composition : To estimate the execution time of a particular WSC, its components are traversed. The components can be orchestrated in different ways to structure the WSC. We propose to use the Workflow patterns compiled in [13]. These patterns capture typical control flow dependencies encountered in Workflow modeling and they arguably apply as well for WSCs, since the situations they capture are also relevant in this domain.

PATTERN 1. SEQUENCE : A component C_j in a WSC $Composition_c$ is enabled to start its execution after the successful completion of its direct predecessor, a component C_i, having i, j in $[1..n]$ and $i < j$. The estimation of the *Probable Execution Time* is described in Equation (2.6).

$$T(\textit{Composition}_c)_{prob} = T(C_i)_{prob} + T(C_j)_{prob} \quad (2.6)$$

PATTERN 2. PARALLEL SPLIT: Multiple components are run in parallel simultaneously or in any order. If they are synchronized, after their executions termination, they merge together. Say we have a WSC $Composition_c$ composed of two components C_i and C_j which are executed in parallel, the estimation of the *Probable Execution Time* of the $Composition_c$ is described in Equation (2.7).

$$T(\textit{Composition}_c)_{prob} = \max(T(C_i)_{prob}, T(C_j)_{prob}) \quad (2.7)$$

PATTERN 3. EXCLUSIVE CHOICE: Say we have a composition $Composition_c$ composed of two components C_i and C_j which are executed exclusively, the estimation of the Probable Execution Time of $Composition_c$ is equal to the time taken by the chosen component to execute, that is either C_i or C_j (See Equation(2.8)).

$$T(\textit{Composition}_c)_{prob} = \text{xor}(T(C_i)_{prob}, T(C_j)_{prob}) \quad (2.8)$$

2.6. Reliability property characterization

We define the concept of *Reliability Tendency*(noted RT), which upholds the idea that from one state to another, the reliability contribution varies. RT builds on: the component's *State*, component's *Terminal State Set (TSS)*, component's *State Tendency Set(STS)*, and finally *State Reliability Contribution (SRC)*. These are defined on the base of a history collected by observing the WSC execution. Many are the tools to collect such a history (e.g., Jopera[14]).

DEFINITION 5. Each component C_i, after being invoked for execution, it has a *Terminal State (noted TS)* with which its invocation is terminated. After m invocations, for each component, a *Terminal States Set* (noted $TSS(C_i)$) is formed. The cardinality of $TSS(C_i)$ depends from the different possible *Terminal States* of a component. We assume that it is at least equal to one and at most equal to β. Any $TSS(C_i)$ verifies the condition of Equation(2.9):

$$1 \leq |TSS(C_i)| \leq \beta \quad (2.9)$$

DEFINITION 6. After m invocations of C_i, at least one *Terminal State*, among its different possible *Terminal States* tends to have the biggest occurrence ratio. We introduce the notion of *State Tendency Set* (noted $STS(C_i)$, as the *Terminal State(s)* that has(have) the biggest occurrence(s) ratio(s) after m invocations of that component. In other words, $STS(C_i)$ is the set of state(s) that are included within the *Terminal States Set* of C_i and which has(have) the biggest occurrence ratio(s) (noted $Occur(State)$) after m invocations of C_i (see Equation(2.10)). $STS(C_i) \subseteq TSS(C_i)$ and $occur(State)$ is the number of *times* (after *m* invocations) C_i execution was terminated with the *TS* $State$.

$$STS(C_i) = \max_{State \in TSS} \{\sum_{1}^{m} State(occur(State))\} \quad (2.10)$$

DEFINITION 7. From one *Terminal State* to another, the contribution in the reliability differs. For instance, terminating the execution of a component C_i in the `Failed` state would affect negatively the reliability, contrary to the `Committed` state which would contribute positively in increasing the reliability. We define this concept as the *State Reliability Contribution*(noted SRC) of a particular *TS*. We assume that a transition from one *Terminal State* to another makes the *SRC* stronger if is towards reaching a state denoting the execution success and it contributes negatively and makes the *SRC* weaker if it is toward a state denoting a faulty execution. The definition of the *SRC* of each state depends greatly from the considered environment characteristics (e.g., number of TS, possible states, states transitions and so forth), we will deal with this issue in our future work.

DEFINITION 8. The notion of *Reliability Tendency* of a component C_i (noted $RT(C_i)$) is derived from the previous definitions as in Equation(2.11). It defines the reliability rate of each component, after m invocations.

$$RT(C_i) = \frac{\sum_{\text{TS} \in TSS(C_i)} (occur(TS).SRC(TS))}{|TSS(C_i)|} \quad (2.11)$$

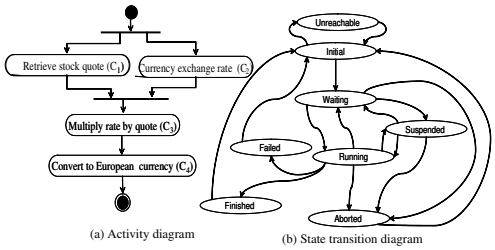
We emphasize that we introduced the appellation of *Reliability Tendency*, equivalent to *Reliability* in other work, because we are convinced that in the WSs context, precise reliability measurements are very difficult to acquire, in view of its dynamism. Estimating the rate to which the reliability will tend is more plausible.

DEFINITION 9. The *Reliability Tendency* of a WSC $Composition_c$, comprising n components, is derived from its components respective *RT* as follows:

$$RT(WSC_c) = \frac{\sum_{0 \leq i \leq n} (RT(C_i))}{n} \quad (2.12)$$

3. Validation

We describe a case study in which we make use of JOpera [14][15], to collect the history of a WSC depicting a quoting process, to show our proposal applicability.



(a) Activity diagram (b) State transition diagram

Figure 1. quoting process: UML diagrams

3.1. Case study

We consider a process that retrieves quotes in the desired currency for a user-provided stock symbol[16].(see activity diagram of Figure.1).To execute the WSC, we use JOpera: a rapid composition tool offering a visual language and an execution platform for building distributed applications out of reusable services. Our choice of JOpera was mainly guided by its practicability. The execution of the WSs follows the state diagram of (Figure.1). The process is invoked eleven times ($m = 11$). The invocations results in terms of execution time and *Terminal States* respectively for each component and for the WSC are in (Figure.2) and (Figure.3). The reasons behind the failures that have occurred were:

- The Internet connection failed during the SOAP message round trip (e.g., instance ♯9);
- The WS timed out because of a network connection failure (e.g., instance ♯4);
- The Web service returned a failure message because of data inconsistency (e.g., instance ♯8).

3.1.1. Execution time estimation and analysis : We consider the case where the execution control is delegated to a centralized authority which is responsible for all the components execution and failure recovery and for the WSs discovery and mapping to the components. In this case study, for each component, only one WS has been made available ($|\mathbb{S}_i| = 1 \; \forall \; i \in [1..4]$).

SCENARIO 1.: We consider that the mapped WSs to the components ($C_1 \mapsto ws_{11}$, $C_2 \mapsto ws_{21}$, $C_3 \mapsto ws_{31}$ and $C_4 \mapsto ws_{41}$) terminated successfully in the `Finished` *TS* (e.g., instance ♯0 in (Figure.2)). The *Probable Execution Time* of $composition_1$ is described in Equation(3.1).

$$\begin{aligned} T(Composition_1)_{prob} &= \max(T(C_1)_{prob}, T(C_2)_{prob}) + T(C_3)_{prob} \\ &\quad + T(C_4)_{prob} \\ &= T(C_2)_{opt} + T(C_3)_{opt} + T(C_4)_{opt} \\ &= T(C_2, ws_{21}) + T(C_3, ws_{31}) + T(C_4, ws_{41}) = 4{,}737 sec. \end{aligned} \quad (3.1)$$

With: $T(C_1)_{opt} = T(C_1, ws_{11})$ $\quad T(C_2)_{opt} = T(C_2, ws_{21})$

$T(C_3)_{opt} = T(C_3, ws_{31})$ $\quad T(C_4)_{opt} = T(C_4, ws_{41})$

$I(C_1) + R(C_1) = I(C_2) + R(C_2) = I(C_3) + R(C_3) = I(C_4) + R(C_4) = 0$

Execution time of the different web services and the composition after every invocation (11 instances)

Web Services	Instance No. 0	1	2	3	4	5	6	7	8	9	10
getQuote (WS_{11})	1.102	0.981	2.544	1.282	1.031	0.160	0.902	1.022	1.001	5.859	31.425
getRate(WS_{21})	1.202	1.151	2.564	0.991	1.182	0.160	1.172	1.181	9.704	5.879	31.395
Multiply(WS_{31})	1.222	2.343	2.104	0.010	20.900	0.000	0.010	0.010	0.000	0.000	0.000
FromEuro(WS_{41})	2.313	1.161	6.309	0.000	0.000	0.000	2.592	1.062	0.000	0.000	0.000
composition	4.737	4.655	10.977	1.292	22.082	0.160	3.774	2.253	9.704	5.879	31.425

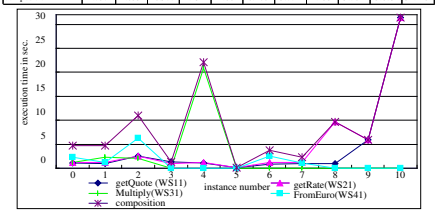


Figure 2. Quoting process: execution results

SCENARIO 2.: In this scenario, we take the case of one of the instances in which one or more components fail. The execution of the component retrial is not possible since there is no other available wss to reattempt it. As a result, a backward recovery needs to be launched. In Equation(3.2), we follow the case of instance ♯4 in which the Web service ws_{31}, allocated to C_3, failed. Equation(3.1) becomes:

$$T(Composition_1)_{prob}=\max(T(C_1)_{prob}+T(C_2)_{prob})+T(C_3)_{prob} \quad (3.2)$$
$$+T(C_4)_{prob}$$
$$T(C_1)_{prob}=T(C_1)_{opt}+R(C_1) \quad T(C_2)_{prob}=T(C_2)_{opt}+R(C_2)$$
$$T(C_3)_{prob}=I(C_3) \quad T(C_4)_{prob}=0 \quad R(C_1)=Back(C_1)=Roll(C_1)$$
$$R(C_2)=Back(C_2)=Roll(C_2) \quad I(C_3)=T(C_3,ws_{31})$$

This case study shows that estimating the *Execution Time* at the component level, and considering all the possible execution situations (going from faulty executions to successful executions)can help the designers to acquire more easily detailed data, without having to go through any complex modeling formalisms. The obtained data is more practical since no simulation systems are needed to analyze the results. On the base of such data, the system designers can locate more smoothly error-prone component(s), their failures reasons might be more easily investigated and eventually, the composition overall structure can be altered, if required, to improve the performances.

3.1.2. Reliability estimation and analysis : From the results of the invocations of the quoting process (Figure.2), we determine the TSS, STS, and RT of the different components of the composition (Figure.3). Typical interpretations of these results are:

- Both C_3 and C_4 tend to not succeed in their executions in 9.1% of the cases because of their own failures (i.e., they tend to terminate in 9.1% of the cases their executions in the `Failed` state). For example, instance ♯4 and instance ♯3 failed respectively because of failures occurred at C_3 (ws_{31} failed to send back its response, a timeout occurred) and at C_4 (a network failure was behind not allowing ws_{41} to receive its input);

Instance	0	1	2	3	4	5	6	7	8	9	10	
Terminal State (TS)	Finished	Finished	Finished	Finished	Finished	Failed	Finished	Finished	Finished	Failed	Failed	(C_1,WS_{11})
	Finished	Finished	Finished	Finished	Finished	Failed	Finished	Finished	Failed	Failed	Failed	(C_2,WS_{21})
	Finished	Finished	Finished	Finished	Failed	Unreachable	Finished	Finished	Unreachable	Unreachable	Unreachable	(C_3,WS_{31})
	Finished	Finished	Finished	Failed	Unreachable	Unreachable	Finished	Finished	Unreachable	Unreachable	Unreachable	(C_4,WS_{41})
	Finished	Finished	Finished	Failed	Failed	Failed	Finished	Finished	Failed	Failed	Failed	Composition$_1$

	Terminal States Set (TSS)	State Tendency Set (STS)	Reliability Tendency (RT)
(C_1,WS_{11})	{Finished(72.7%), Failed(27.3%), Unreachable(0%)}	{Finished(72.7%)}	(72.7%*1+27.3%*(-1)+0%*0.5)/3=15.1%
(C_2,WS_{21})	{Finished(63.6%), Failed(36.4%), Unreachable(0%)}	{Finished(63.6%)}	9.1%
(C_3,WS_{31})	{Finished(54.5%), Failed(9.1%), Unreachable(36.4%)}	{Finished(54.5%)}	21.2%
(C_4,WS_{41})	{Finished(45.5%), Failed(9.1%), Unreachable(45.5%)}	{Finished(45.5%), Unreachable(45.5%)}	19.7%
		Composition$_1$ Reliability tendency(RT)	*16.3%*

Figure 3. Quoting process: TSS, STS, and RT

- C_3 and C_4 tend to not start their executions and to terminate in the `Unreachable` state respectively in 36.4% and 45.5% of the total invocations. A component state is set to the `Unreachable` state when the condition associated with it evaluates to false. In such a case, its execution is skipped[15]. In the case of C_3 and C_4, their conditions were not fired because their predecessors failed (e.g., in instance ♯8, C_2 failed). C_1 and C_2 tendency to finish in the `Failed` is rather high: up to 27.3% for C_1 and 36.4% for C_2. By their failures, they cause the overall composition failure. The reasons that lay behind the frequent failures of C_1 and C_2 need to be investigated. Other WSs bearing the same functionalities as C_1 and C_2 need to be searched. Lastly, revising the WSC structure (i.e., component order, invocation conditions) is to be envisaged;
- The estimations of the *Reliability Tendency*(RT) according to the Equation(2.11) is in (Figure.3). In determining these estimations, defining the different *State Reliability Contributions*(SRC) of each *Terminal State* was required. The estimation of the different SRC is done on the base of collected history. In this case study, as initial values, we affected to the *Terminal States* `Finished`, `Failed` and `Unreachable` the SRC $+1.0$, -1.0 and $+0.5$, respectively. Our motivation behind allocating such values is that, we advocate that the `Finished` state is the best contributing in the reliability, the the `Failed` state is the worst contributing in the reliability, and the `unreachable` state is in between: the component execution was about to start but it did not because its activation condition was not fired.

4. Related work

4.1. Conventional composite systems context

In addressing the QoS of conventional composite systems, a large variety of techniques has been proposed[17][12]. Those techniques are supported by underlying modeling formalisms (e.g., block diagrams, Markov chains, Petri-nets, logics, etc.).

Similarly, in the context of software engineering, many mathematical techniques have been developed. The mostly-related models to our paper are the structural models of reliability[18] and the Markov reward models[19], which form the basis of all performability models. In the former, a state diagram which depicts the system behavior is used. Based on Markov chain properties, the transition between states is assumed as a Markov process. In the later, the system is assumed to be modeled as a Markov process with finite state space and a reward rate (performance measure) is associated with each state. The main shortcoming here is that building good models requires lots of expertise and efforts. And very often, the designers are not keen of building such models because of their inherent complexity. Moreover, the obtained models are not straightforwardly interpretable, further simulations need to be performed.

4.2. Web services context

The work found in the literature that relate to the estimation and analysis of QoS properties of WSCs, is very limited. One category is oriented towards rating the QoS by the WSs users. In[20], an approach for selecting services based on their semantics as well as their quality as judged by users is proposed. To this end, a query language based on DAML that accommodated several essential query and manipulation templates is developed. The users/providers estimations of the QoS might be incorrect and/or biased by the users impartiality. In our work, we do not rely on the users/providers QoS rating, instead, designers observe the WSCs execution and collect the execution results in a history to use later as a base to estimate the QoS properties.

In[21], the authors discussed a model with Service Level Agreement(SLA) which is used as a bridge between providers and consumers. However, when considering WSCs, and in particular, dynamic WSCs, dealing with SLAs gets very complex. Hierarchical QoS Markup Language (HQML), Web Ontology Language (OWL-S), Web service Level Agreement language (WSLA), are examples of specifications that have addressed the need for a QoS model. The common point between these specifications is that they have described the QoS of WSs. For instance, DAML-S, has included constructs which specify several QoS parameters, namely, the quality rating and the degree of quality. However, these specifications have not supplied any precise characterization of the different parameters and it is only for WSs.

[6] has presented a model to evaluate the QoS of both elementary and composite services. In [6], the potential failure repercussions on the global execution time has not been considered. The reliability was mapped directly to each of theWS individual reliability. It was defined as the probability that a request to a particular WS is correctly responded within a maximum expected time frame.

This method of characterizing the reliability is not extensible to dynamically assembled WSCs.

4.3. Workflow Management System(WFMS)context

In view of the similarities that exist with the area of WFMS(i.e., both deal with aggregating components following a process underpinning logic), we consider some related work in this area. The Crossflow project[22][23] and the METEOR project[5][2][24][25][26] have made major contributions on QoS. Specifically, the METEOR project has investigated four QoS dimensions, namely the time, the cost, the reliability, and the fidelity. However, it has not considered in any way dynamic WSCs. It has focused on analyzing, predicting, and monitoring the QoS of workflows. Their proposed model is actually derived from a more general work[7], in which to describe tasks reliability in workflow context, the discrete-time stable reliability model proposed in[27] is followed(see Equation(4.1)). In this WFMS, task structure[28] has a prefixed number of states and all of the states contribute equally in the reliability. This way of modeling the tasks is limitative. It is worth if the model is easily extensible with other state. Both of the above models are for static WSCs.

$$R(t) = 1-(\text{system failure rate}+\text{process failure rate}) \quad (4.1)$$

5. Conclusions and future directions

We presented a novel model for characterizing, estimating, and analyzing the efficiency of dynamically-executed fault-tolerant Web services compositions.

Our work main contributions resided in, first, proposing an innovative approach for estimating the QoS in terms of execution time and reliability, in the context of Web services compositions. While most of the current approaches dealing with QoS estimations in the WSs context rely on the QoS information advertised by the WSs providers, our model estimates QoS properties on the base of the WSCs executions observations, where the observation results are collected in a history. In doing so, more accurate estimations can be acquired since we do not rely on the providers data which might be not up to date, or which might be subject to manipulation.

Second, our work conferred a paramount importance to the failure repercussions on the WSCs efficiency. In fact, not only correct execution instances were examined to estimate the efficiency and later analyze it, but our model was oriented towards considering the system in all of its possible states: correct, faulty, recoverable executions. In doing so, we intended to make our model capable to reflect the real state of any system in general, and to the typical case of WSCs, with their inherent tendency to fail rather easily compared with others.

Third, since WSs are stateless, tracking failures and determining their location is very difficult.

To overcome this problem, we attached to each component a state. In doing so, the more error-prone components can be more easily found. The reasons that lay behind the failures can be investigated and approximated and such information can be used to improve the composition quality in the future. Finally, who says a model applicable to the Web services context says also it is applicable to any business process modeling application or classic distributed composite systems (e.g., workflow), in view of the similarities . Indeed a high level of applicability to a wide range of systems is foreseen for our model.

Our ongoing work comprises mainly conducting experiments using our implemented simulation system of our previous work WS-SAGAS, a transaction model for WSCs reliable specification, and THROWS, an architecture for highly available execution of WSCs [3] [29][30][4] to evaluate our proposed approach. We intend to introduce a new module dedicated to collecting in a history, the past executions. Finally, we intend to enrich our model with other efficiency estimation dimensions.

Acknowledgment

Part of this research was supported by CREST of JST (Japan Science and Technology Agency), a Grant-in-Aid for Scientific Research on Priority Areas from MEXT of the Japanese Government (#16016232), and the 21st Century COE Program "Framework for Systematization and Application of Large-scale Knowledge Resources."

References

[1] Q.Sheng B.Benatallah M.Dumas and E.Y.Mak. Self-serv: A platform for rapid composition of web services in a peer-to-peer environment. In *VLDB*, pages 1051–1054, 2002.

[2] J.Cardoso and A.Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems*, 2003.

[3] N. Benlakhal T. Kobayashi and H. Yokota. Throws: An architecture for highly available distributed execution of web services compositions. In *RIDE WS-ECEG'2004, the 14th Int.Workshop on Research Issues on Data Eng.: Web Services for E-Commerce and E-Government Applications*, pages pp.103–110, Boston, USA, March 2004. IEEE.

[4] N. Benlakhal T. Kobayashi and H. Yokota. Distributed architecture for reliable execution of web services. Technical report, IEICE, DBWS2003 2B, 2003.

[5] S.Chadrasekaran J. A. Miller G. Silver I. B. Arpinar and A. Sheth. Composition, performance analysis and simulation of web services. *Electronic Markets: The International Journal of Electronic Commerce and Business Media*, 2003.

[6] L.Zeng B.Benatallah M.Dumas J.Kalagnanam and Q.Z.Sheng. Quality driven web services composition. In *the 12th inter. conf. on World Wide Web*, pages 411 – 421, Budapest, Hungary, 2003. ACM Press.

[7] J.Cardoso A.Sheth J.Miller J.Arnold and K.Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 2004.

[8] H.Schuster, D.Baker, A.Cichocki, D.Georgakopoulos, and M.Rusinkiewicz. The collaboration management infrastructure. In *ICDE*, pages 677–678, 2000.

[9] F.Casati and M.Shan. Event-based interaction management for composite e-services in eflow. *Information Systems Frontiers*, 4(1):19–31, 2002.

[10] L.Lamport R.Shostak and M.Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3), 1982.

[11] F.C.Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26, 1999.

[12] J-C.Laprie. Dependable computing and faut tolerance: concepts and terminology. In *Proc. of the* 15^{th} *int. Sym. on Fault-tolerant Computing (FTCS-15)*, pages 2–11, 1985.

[13] P.Wohed, Wil M.P. v.Aalst, M.Dumas, and A.H.M.Hofstede. Pattern based analysis of bpel4ws. Technical report, Queensland Uni.of Tech., Brisbane, 2002.

[14] C.Pautasso T.Heinis and G.Alonso. Autonomic execution of service compositions. In 3^{rd} *IEEE Int. Conf. on Web Services (ICWS'05)*, Orlando, USA, July 2005.

[15] C.Pautasso. *A Flexible System for Visual Service Composition*. PhD thesis, ETH, July 2004.

[16] xmethods. www.xmethods.net, 2004.

[17] H.Kobayashi. *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*. Addisonwesley edition, 1978.

[18] R.C.Cheung. A user-oriented software reliability model. *IEEE Transactions On Software Eng.*, 6(2):118, March 1980.

[19] M.A.Qureshi and W.H.Sanders. Reward model solution methods with impulse and rate rewards:an algorihm and numerical results. *Performance evaluation*, 1994.

[20] A. Soydan Bilgin and Munindar P. Singh. A daml-based repository for qos-aware semantic web service selection. In *ICWS*, pages 368–375, 2004.

[21] Li.Jin, V.Machirajuand, and A.Sahai. Analysis on service level agreement of web services. Tech. Report HPL-2002-180, Software Technology Laboratories, HP Lab., June 2002.

[22] P.Grefen K.Aberer H.Ludwig and Y.Hoffner. Crossflow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. *IEEE Data Eng.Bulletin*, 24(1):52–57, 2001.

[23] J.Klingemann J.Wasch and K.Aberer. Deriving service models in crossorganizational workflows. In *RIDE-Information Tech.for Virtual Enterprises*, Australia, March 1999.

[24] J.Cardoso J.Miller A.Sheth and J.Arnold. Modeling quality of service for workflows and web service processes. *technical report*, 2002.

[25] A.Sheth J.Cardoso J.Miller and K.kochut. Service-oriented middleware. In 6^th *World Multiconf. on Systemics, Cybernetics and Informatics*, Orlando, FL, 2002.

[26] J.Cardoso. *Quality of Service and Semantic Composition of Workflows*. Ph.d. dissertation, Dep. of Computer Science, University of Georgia, Athens, GA., 2002.

[27] E.C.Nelson. A statistical basis for software reliability assessment. Technical report, TRW Systems Report, March 1973.

[28] N.Krishnakumar and A.Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2), 1995.

[29] N. Benlakhal T. Kobayashi and H. Yokota. A simulation system of throws architecture for ws-sagas. Technical Report 7-B-4, 14th IEICE Data Eng. Workshop, March 2004.

[30] N. Benlakhal T. Kobayashi and H. Yokota. Ws-sagas: transaction model for reliable web-services-composition specification and execution. *DBSJ letters*, 2(2):17–20, Oct. 2003.

[31] A.Ankolekar. Daml-s: Web service description for the semantic web, 2002.