TQoS: Transactional and QoS-aware selection algorithm for automatic Web service composition

Joyce El Haddad, Maude Manouvrier, and Marta Rukoz

Abstract—Web Services are the most famous implementation of service oriented architectures that has brought some challenging research issues. One of these is the composition, i.e. the capability to recursively construct a composite Web service as a workflow of other existing Web services, which are developed by different organizations and offer diverse functionalities (e.g. ticket purchase, payment), transactional properties (e.g. compensatable or not) and Quality of Service (QoS) values (e.g. execution price, success rate). The selection of a Web service, for each activity of the workflow, meeting the user's requirements, is still an important challenge. Indeed, the selection of one Web service among a set of them that fulfill some functionalities is a critical task, generally depending on a combined evaluation of QoS. However, the conventional QoS-aware composition approaches do not consider the transactional constraints during the composition process. This paper addresses the issue of selecting and composing Web services not only according to their functional requirements but also to their transactional properties and QoS characteristics. We propose a selection algorithm that satisfies user's preferences, expressed as weights over QoS criteria and as risk levels defining semantically the transactional requirements. Proofs and experimental results are presented.

Index Terms—Web Service Selection, Automatic Composition, Transactional Web Service, Local Optimization of Quality of Service, Workflow Patterns.

1 Introduction

Web Services (WSs) are the most famous implementation of service oriented architectures allowing the construction and the sharing of independent and autonomous softwares. Web service composition consists in combining Web services, developed by different organizations and offering diverse functional (e.g. ticket purchase, payment), behavioral (e.g. compensatable or not) and non-functional properties (i.e. Quality of Service values – e.g. execution price, success rate), to offer more complex services.

The Web service composition can be view as a three-steps process: (1) composite Web service specification, (2) selection of the component Web services and (3) execution of the composite Web services. At the first step, the user submits the goal he/she wants the composite service achieves, along with some constraints and preferences that need to be satisfied [1]. Workflows can be used to model the composite Web service specification. During the second step, component Web services fulfilling the user's goal are selected among a set of available services. This WS selection could be done by hand (in this case, steps specification and selection are integrated) or could be automatically decided by the system. When component WS are selected at design time, the third step of the composition process consists in executing the selected component WS. At run-time, selection and execution of component WS are integrated and the selection is described as dynamic. In this paper, we focus on design-time WS selection and particularly on automatic selection where the user is relieved as much as possible from the composition process. We do not focus on the execution step neither on the recovery or re-planning problems.

While many works have been done for Web service selection, designing a composite Web service to ensure not only correct and reliable execution but also optimal Quality of Service (QoS) remains an important challenge [2]. Indeed, WSs composition based on transactional properties ensures a reliable execution however an optimal QoS composite Web service is not guaranteed. Moreover, composing optimal QoS Web services does not guarantee a reliable execution of the resulting composite Web service. Thus, QoS-aware and transactional-aware should be integrated. However, the problem is generally addressed from the QoS side or from the transactional side separately. The conventional QoS-aware composition approaches [3], [4], [5], [6], [7] do not consider the transactional constraints during the composition process, likewise transactional-aware ones [8], [9], [10], [11], [12] do not consider QoS. As far as we know, only [2] proposes a composition model in design-time which captures both aspects in order to evaluate the QoS of a composite WS with various transactional requirements. However, the authors do not consider the automatic selection step and only analyze the impact of the transactional requirements on the QoS of the composite WS.

Our research objective is to propose a design-time

- *Joyce El Haddad, Maude Manouvrier and Marta Rukoz are with the LAMSADE, Paris-Dauphine University, Place du Maréchal de Lattre de Tassigny 75775 Paris Cedex 16 - France. E-mail: elhaddad@lamsade.dauphine.fr,manouvrier@lamsade.dauphine.fr, Marta.Rukoz@dauphine.fr*
- *Maude Manouvrier and Marta Rukoz are with WISDOM, Federation of the three database research teams, LIP6 (Paris 6 University), LAMSADE and CEDRIC (CNAM). http://wisdom.lip6.fr/*
- *Marta Rukoz is with Paris Ouest Nanterre La Défense University, 200 avenue de la République, 92100 Nanterre, France. On leave from Universidad Central de Venezuela.*

selection algorithm for automatic WS composition where transactional and QoS requirements are both integrated in the selection process. It is evident that such integration could only be done by considering first the transactional requirements. In fact, as we mentioned above, a local or global optimized QoS composition may not guarantee transactional execution. If the selection is done in two separate steps (transactional selection followed by a QoS one), it is necessary to consider all the transactional combinations satisfying the global transactional requirements, which is a combinatorial problem. For all these reasons, we embedded the QoS-aware service selection with the transactional-aware service selection, using local QoS optimization in order to reduce the set of possible transactional solutions.

Our contribution is a Web service selection algorithm which guarantees that each selected component WS of a composite one is locally the best QoS WS among all the WS fulfilling the global transaction requirement. Moreover, our algorithm is scalable because the user has only to define a global transaction requirement and does not have to define the possible termination states of all component WS, as done for example in [8], [11], [13], because the complexity of such a process increases with the number of component WS.

This paper extends our previous work presented in [14], where the composition was limited to elementary (non composite) WS. In this paper, we extend our design-time selection algorithm to any component WS (elementary or composite). Moreover, it contains a formal analysis of our algorithm and a more extended state of the art study.

The paper is organized as follows. Section 2 presents our system architecture. Section 3 describes the behavioral and non-functional properties of elementary WS. Section 4 defines the properties of composite Web services. Our approach is presented in Section 5. Experimental results are shown in Section 6. A discussion about related work is done in Section 7. Finally, Section 8 concludes and gives perspectives.

2 System Architecture

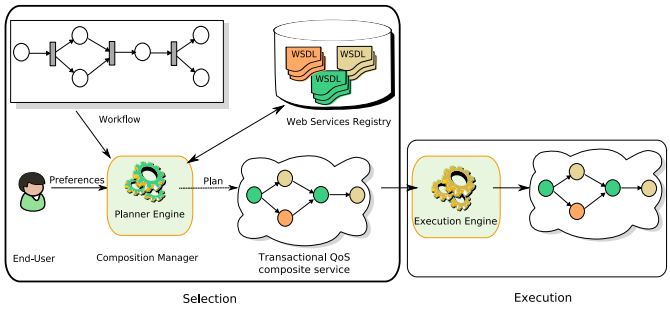


Fig. 1. Architecture of our system

Our system, represented in Figure 1, takes two inputs: a *workflow* and the *user's preferences*. The workflow defines the execution order of a set of n activities, each one being performed by a WS. It represents the skeleton of an application in terms of activities and temporal dependencies between them. The user's preferences are expressed as weights over QoS criteria and as risk levels. They define semantically the transactional requirements of the resulting Composite Web service (CWS), which corresponds to the assignation of one component WS to each activity of the workflow. In this paper, for the sake of simplicity, we consider that one WS executes only one activity.

Based on the input workflow, the *Composition Manager* contacts the *Web services registry* to search for candidate component WS for each activity of the workflow, according to the user's preferences. The *Web Service Registry* provides the means for registering and discovering WS, and managing associated metadata and artifacts securely and reliably. The metadata describe the functional, behavioral and non-functional properties of a WS. Based on the candidate WS retrieved, the *Planner Engine* generates an execution plan, i.e. an assignment of a component WS to each activities of the input workflow. Based on the execution plan, the *Execution Engine* then orchestrates the component WS to execute the instance of the composite WS, which enforces the transactional requirements and the QoS criteria defined by the user.

3 Web Service description

Since WS are intended to be discovered and used by other applications, they need to be described and understood in terms of functional capabilities as well as behavioral properties and non-functional properties. In this paper, we only discuss the last two types of properties. Section 3.1 defines the behavioral properties of a WS. Section 3.2 presents the QoS properties we consider for a WS in the paper.

3.1 Web Service behavioral properties

The behavioral description of a WS is regarding how the functionality of a WS can be achieved in terms of interaction with the other WS. In a composition where several component WS interact, unexpected behavior from a component WS might, not only lead to its failure, but also may bring negative impact on all the participants of the composition. Therefore, as for all cross-organizational collaborative systems, the execution of a CWS requires Transactional Properties (TP) so that the overall consistency is ensured. We consider three behavioral properties of WS, *pivot* (p), *compensatable* (c) and *retriable* (r). Inspired by [15], we have the following definitions:

Definition 1 (Pivot WS): A WS is said to be pivot (p) if once it successfully completes, its effects remains forever and cannot be semantically undone. If it fails it has no effect at all. A completed pivot WS cannot be rolled back.

Definition 2 (Compensatable WS): A WS s is compensatable (c) if it exists another WS s' which can semantically undo the execution of s.

Definition 3 (Retriable WS): A WS is retriable (r) if it guarantees a successfully termination after a finite number of invocations.

A WS can combine behavioral properties, then the set of all possible combinations for a WS is $\{p, c, pr, cr\}$.

3.2 Web service non-functional properties

When several functionally and transactionally equivalent WS are available to perform the same activity, their QoS properties such as price, availability, reliability and reputation become important in the selection process. In order to reason about QoS properties in WS, a model is needed to capture the descriptions of these properties from a user perspective. Such model must take into account the fact that QoS involves multiple dimensions. In this paper, we consider the following five generic quality criteria for a WS s:

1) Execution price ($q_{ep}(s)$): which is the fee that a requester has to pay for invoking s.
2) Execution duration ($q_{ed}(s)$): that measures the expected delay time between the moment when s is invoked and when the results are received.
3) Reputation ($q_r(s)$): which is a measure of trustworthiness of s. Generally this measure is defined as the average ranking given to the service by end users.
4) Successful execution rate ($q_{sr}(s)$): which is the probability that s responds correctly to the user request.
5) Availability ($q_a(s)$): which is the probability that s is accessible.

Properties 1, 2, 4 and 5 can be dynamically updated by the system.

The following section defines a CWS whose components have the five properties defined above.

4 Composite Web Service description

A CWS is a conglomeration of existing WS interacting together to offer a new value-added service. It coordinates a set of WS as a cohesive unit of work to achieve common goals. Section 4.1 presents how to specify a CWS. Sections 4.2 and 4.3 present the CWS behavioral and non-functional properties.

4.1 Composite Web Service specification

Currently, several process modeling languages including YAWL [16] and BPEL [17] have been proposed to capture the logic of a CWS. In this paper, we choose YAWL to represent the workflows model and to describe the composition. However, any other language could have been used. In a WS environment, a workflow represents a composition of WS. When each activity of a workflow is implemented by a component WS, we obtain a CWS. Several CWSs can be associated to the same workflow, depending on the assigned component WSs. The orchestration of the component WSs is defined by specifying dependencies between them. These dependencies are defined by the associated workflow patterns and by the transactional properties. The first ones specify how WSs are coupled and how the behavior of certain WSs influence the behavior of other one(s). While the transactional properties specify the behavior of certain WSs in case of failure.

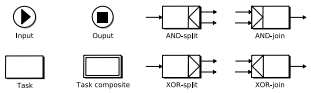


Fig. 2. Symbols used in YAWL

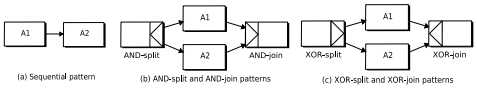


Fig. 3. Workflow patterns

In this paper, for the sake of simplicity, we restrain the temporal dependencies between activities to the following workflow patterns: sequence, parallel split (AND-split), exclusive choice (XOR-split), synchronization (AND-join) and simple merge (XOR-join), presented in Figure 2. Figure 3.(a) represents a sequential pattern between two activities A_1 and A_2. Using this pattern, when a service WS_1 is assigned to activity A_1 and a service WS_2 is assigned to activity A_2, the obtained composite Web service CWS_1 is represented in text by $(WS_1; WS_2)$ where symbol ; represents a sequential execution: WS_1 is executed before WS_2. Figure 3.(b) represents the AND-split and AND-join patterns. In this case, when a service WS_1 is assigned to activity A_1 and a service WS_2 is assigned to activity A_2, the resulting composite Web service CWS_2 is represented in text by $(WS_1 // WS_2)$, meaning that both services are executed in parallel. For a XOR-pattern (see Figure3.(c)), the obtained CWS_3 corresponds to one of its component. Then, when a service WS_1 is assigned to activity A_1 and a service WS_2 is assigned to activity A_2, the obtained CWS is represented in text by $(WS_1 \mid WS_2)$, meaning that either service WS_1 either service WS_2 is executed. For more details, not needed to understand the rest of this paper, readers are referred to [18] for a uniform approach to describe workflow characteristics and to [3] for a description of relevant patterns for the WS composition.

A workflow pattern can be composed by other workflows patterns. In this way, we can represent a CWS by the composition of other CWS, or by the composition of CWS with elementary WS. In the following sections, when it is not specified, WS represents an elementary WS or a CWS.

The next section describes the CWS behavioral (i.e. transactional) and QoS (i.e. non-functional) properties.

TABLE 1
Aggregation functions for QoS criteria

Criteria	Aggregation function
Price	$q_{ep}(CWS) = \sum_{i=1}^{n} q_{ep}(s_i)$
Duration	$q_{ed}(CWS) = \sum_{i=1}^{n} q_{ed}(s_i)$
Reputation	$q_r(CWS) = \frac{1}{n} \sum_{i=1}^{n} q_r(s_i)$
Success rate	$q_{sr}(CWS) = \prod_{i=1}^{n} q_{sr}(s_i)$
Availability	$q_a(CWS) = \prod_{i=1}^{n} q_a(s_i)$

4.2 Composite Web Service behavioral properties

CWS are often long-running, loosely coupled and cross-organizational applications. In this context, we are interested in transactional behavior of the resulting WS composition. The transactional properties of a CWS highly depend on the transactional properties of its component WSs and on the structure of the workflow. We have the following definitions:

Definition 4 (Atomic CWS): A CWS is atomic if once all its component WSs complete successfully, their effect remains forever and cannot be semantically undone. On the other hand, if one component WS does not complete successfully, then all previously successful component WSs have to be compensated. In the following, $\vec{a}$ is used to indicate an atomic CWS while $\tilde{a}$ is used to indicate a non-atomic one.

Definition 5 (Compensatable CWS): A CWS is compensatable (c) if all its component WSs are compensatable.

Definition 6 (Retriable CWS): An atomic or a compensatable CWS is retriable (r) if all its components are retriable.

Definition 7 (Transactional CWS): A Transactional Composite Web Service (TCWS) is a CWS whose transactional behavioral property is in $\{\vec{a}, \vec{a}r, c, cr\}$.

A TCWS takes advantage of component service behavioral properties to specify mechanisms for failure handling and recovery. It can be composed of elementary WSs, whose properties are in $\{p, pr, c, cr\}$, and/or can be composed of CWSs, whose properties are in $\{\vec{a}, \vec{a}r, c, cr\}$.

In this paper, we are interested in properly assigning component WSs (elementary or composite) in order to obtain a TCWS. Section 5 presents the assignation process.

4.3 Composite Web Service non-functional properties

A CWS has the same quality properties as an elementary WS, i.e. execution price, execution duration, reputation, successful execution rate, and availability. When a user wants to execute a CWS, it indicates, among other things, the quality of the wished result. This one is expressed as weight in each of the quality criterion. In this paper, the QoS of a CWS is evaluated by using the aggregation functions defined in Table 1. It is obvious, that activities in all execution paths between AND-split and AND-join are considered in the aggregation functions. While, activities in only one execution path between XOR-split and XOR-join constructs are considered. However, any other QoS evaluation could be used.

5 Transactional QoS-driven selection for WS Composition

In this paper, we are interested in properly selecting component WSs in order to obtain a TCWS which maximizes the user satisfaction in terms of QoS criterion and satisfies the transactional requirements set by the user and by the input workflow.

As explained in Section 2, the input of the assignation process is a workflow, a transactional requirement, expressed in term of risk (see Section 5.1), and a set of weights over QoS criteria. The output of the process is a TCWS which assigned WS components maximize the QoS criteria.

The process is sequential: WSs are assigned to each activity by analyzing the workflow from the left to the right. In a split pattern, services are assigned from top to down. The transactional driven service selection is presented in Section 5.2. The QoS-driven service selection is presented in Section 5.3. Finally, the algorithm of our Transactional QoS-driven (TQoS) selection for WS Composition is given in 5.4.

5.1 Definition of risk

In order to explain the transactional WS selection process, it is necessary to establish how the user can express their transactional criteria. Although, expressing the QoS criteria is significant for the user, the risk or the possibility that an application will be unsuccessfully executed has a more significant effect on the user's decisions. The importance of the uncertainty of application completion and recovery is semantically expressed under a criterion called *risk*. For example, the set $\{\vec{a}, \vec{a}r, c, cr\}$ of the behavioral properties of CWS can be divided into two subsets $\{\vec{a}, \vec{a}r\}$ and $\{c, cr\}$, each one can be associated with a level of risk. For instance, in terms of the transactional properties, we believe that properties $\vec{a}$ and $\vec{a}r$ are riskier than c and cr. Indeed, properties $\vec{a}$ and $\vec{a}r$ mean that once a service has been executed, it can not be rolled back. Therefore, we define two notions of execution risk in a transactional system like:

- **Risk 0***: the system guarantees that if the execution is successful, the obtained results can be compensated by the user.*
- **Risk 1***: the system does not guarantee the successful execution and even if the execution is successful, the system does not guarantee the result can be compensated by the user.*

In this paper, we only study the risk level 0 and 1 as defined above. However, other levels of risk could be defined in terms of compensation of the different components of the CWS. For example, the system can guarantee that if the execution is successful, some results

can be compensated by the user or some results cannot be compensated by the user. In this case, the transactional properties must be relaxed.

In the following section, we present the WS selection algorithm used by the composition manager for WS composition with risk and QoS preferences.

5.2 Transactional driven service selection

This section shows the process of assigning an WS to each activity in a workflow, in order to obtain a TCWS. For simplicity, we suppose a workflow containing only two activities. We first consider the assignation of two WSs to the activities of a sequential pattern (see Section 5.2.1). Then, we consider the assignation of two WSs to the activities of a parallel pattern (AND-split, see Section 5.2.2). We do not consider the XOR-pattern due to if the workflow contains only two activities in a XOR-pattern, then the resulting "Composite" WS contains only one Web service WS_i hence the WS transactional property corresponds to the transactional property of WS_i. Then, we present a generalization of the transactional driven service selection to a worklfow of n activities (see Section 5.2.3). A summary is presented in Section 5.2.4 and an example is given in Section 5.2.5.

5.2.1 Sequential pattern assignation

Prop. 1: In a sequential pattern, if the WS assigned to the first activity of the pattern is pivot (p), pivot retriable (pr), atomic ($\vec{a}$), or atomic retriable ($\vec{a}r$) then, to obtain a TCWS, the WS assigned to the second activity should be pivot retriable (pr), atomic retriable ($\vec{a}r$), or compensatable retriable (cr). The Transactional Property (TP) of the resulting TCWS is atomic ($\vec{a}$) and is moreover atomic retriable ($\vec{a}r$) if all its components are retriable.

Proof: If the WS assigned to the first activity of the pattern is p, pr, $\vec{a}$, or $\vec{a}r$ then by definition (see def. 1 and 4), its effects cannot be semantically undone, thus the execution of the second WS should guarantee a successfully termination. The only condition to guarantee a successfully termination is the retriable property (see def. 3 and 6). Therefore, the WS assigned to the second activity should be pr, $\vec{a}r$, or cr. □

Prop. 2: In a sequential pattern, if the WS assigned to the first activity of the pattern is compensatable (c) or compensatable retriable (cr), then the resulting CWS is always transactional ($TCWS$) whatever the TP of the WS assigned to the second activity is. The TP of the resulting TCWS is atomic ($\vec{a}$) if the WS assigned to the second activity is either pivot (p), pivot retriable (pr), atomic ($\vec{a}$), or atomic retriable ($\vec{a}r$). The $TCWS$ TP is compensatable (c) if the WS assigned to the second activity is either compensatable (c) or compensatable retriable (cr). Moreover, when both component WSs are retriable, the resulting TCWS is retriable.

Proof: When service WS_1 assigned to the first activity of the pattern is c or cr, the resulting CWS is at least $\vec{a}$ (see def. 4) because if the WS assigned to the second activity fails, then WS_1 can be compensated. Moreover, the resulting CWS is c (see def. 5) when the WS assigned to the second activity is c or cr, because all the components of the resulting CWS are c. □

5.2.2 Parallel pattern assignation

Prop. 3: If a pivot (p) or an atomic ($\vec{a}$) WS is assigned to one activity of a parallel pattern, then the WS assigned to the other activity should be compensatable retriable (cr) to obtain a TCWS. The transactional property of the resulting TCWS is atomic ($\vec{a}$).

Proof: If a p or $\vec{a}$ Web service WS_1 is assigned to one activity of a parallel pattern and if it successfully completes, then by definition (see def. 1 and 4) its effects cannot be semantically undone. Therefore, the execution of the other WS should guarantee a successfully termination in case of WS_1 successfully termination and should be able to be compensated in case of WS_1 failure. By definition (see def. 3 and 6), the only condition to guarantee a successfully termination which can be compensated is property cr. Consequently, the WS assigned to the other activity of the pattern should be cr. □

Prop. 4: If a pivot retriable (pr) or an atomic retriable ($\vec{a}r$) WS is assigned to one activity of a parallel pattern, then the WS assigned to the other activity should be pivot retriable (pr), atomic retriable ($\vec{a}r$), or compensatable retriable (cr) in order to obtain a TCWS. The transactional property of the resulting TCWS is atomic retriable ($\vec{a}r$).

Proof: If a pr or an $\vec{a}r$ service WS_1 is assigned to one activity of a parallel pattern, then, by definition (see def. 3 and 6), it guarantees a successfully termination, but it cannot be compensated. Thus, the WS assigned to other activity cannot fail. So to obtain a TCWS, the only solution is to assign a pr, cr, or $\vec{a}r$ WS to the other activity of the pattern. □

Prop. 5: If a compensatable (c) WS is assigned to one activity of a parallel pattern, then the WS assigned to the other activity should be compensatable (c) or compensatable retriable (cr) in order to obtain a TCWS. The transactional property of the resulting TCWS is compensatable (c).

Proof: If a c service WS_1 is assigned to one activity of a parallel pattern and if it successfully complete, then, by definition (see def. 2 and 5), its effects can be semantically undone. But it can fail. Therefore, to obtain a TCWS the WS assigned to the other activity of the pattern should be at least c (and possibly cr) in order to be able to be compensated. □

Prop. 6: If a compensatable retriable (cr) WS is assigned to one activity of a parallel pattern, then the resulting CWS is transactional ($TCWS$) independently of the WS transactional property assigned to the other activity. The transactional property of the resulting TCWS is respectively atomic ($\vec{a}$), compensatable (c), atomic retriable ($\vec{a}r$), or compensatable retriable (cr), if the WS assigned to the other activity is respectively pivot/atomic

($p/\vec{a}$), compensatable (c), pivot/atomic retriable ($pr/\vec{a}r$), or compensatable retriable (cr).

Proof: If a cr WS is assigned to one activity of a parallel pattern, then, by definition (see def. 2, 3, 5 and 6) it guarantees a successfully termination and it can be compensated. Hence, whatever is the property of the WS assigned to the other activity, the resulting TCWS is at least $\vec{a}$ (see def. 4). Moreover, the resulting TCWS is c when the service assigned to the second activity is c or cr, because all the components of the resulting TCWS are c. It is retriable when the WS assigned to the second activity is pr, $\vec{a}r$, or cr, because all its components are r. □

Tables 2,3,4, and 5 summarize the sequential and the parallel pattern assignation.

5.2.3 Generalization to a workflow of n activities

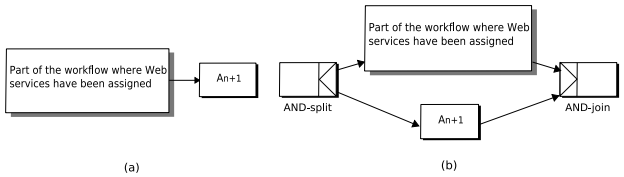


Fig. 4. Step $(n+1)$ of the selection process

WSs are assigned to the activities of the workflow from the left to the right in sequential patterns and from the top to the bottom for split patterns. In this way, the planner engine guarantees that each time a WS is assigned to one activity, the composite WS obtained is transactional. Let first consider elementary WS. After the assignation of the n elementary WS to the n first activities of the workflow, the assignation process consists on composing a TCWS (resulting from the composition of the first n WSs assigned) with an elementary WS (assigned to the activity $n+1$ of the workflow). Figures 4.(a) and 4.(b) show the different possible configurations of the process, after the assignation of the first n WSs to the first n activities of the workflow (when $n=2$ see Table 2 and when $n>2$ see Table 3).

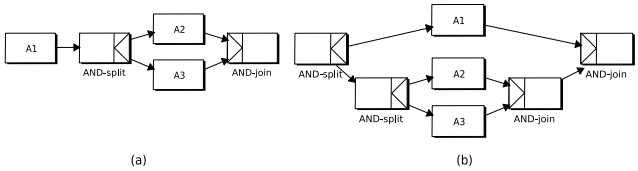


Fig. 5. Examples of workflow resulting in an elementary WS composed with a TCWS

All the WSs assigned to the activities of an AND-split pattern constitute a TCWS which can be sequentially composed or composed in parallel with an elementary WS. By convention, consider that a Web Service WS_i is assigned to a workflow activity A_i. For example, using the workflow of Figure 5.(a), the resulting TCWS is $(WS_1;(WS_2//WS_3))$ and using the workflow of Figure 5.(b) the resulting TCWS is $(WS_1//(WS_2//WS_3))$. Both resulting TCWSs can be viewed as the composition of an elementary Web service WS_1 with a TCWS composed by WS_2 and WS_3. Then, it is possible to compose an elementary WS with a TCWS (see Table 4).

TABLE 2
Transaction property of a sequential and a concurrent execution of two elementary WSs

	WS_1	WS_2	$WS_1;WS_2$	$WS_1//WS_2$
(1)	p	p	$\tilde{a}$	$\tilde{a}$
(2)	p	c	$\tilde{a}$	$\tilde{a}$
(3)	p	pr	$\vec{a}$	$\tilde{a}$
(4)	p	cr	$\vec{a}$	$\vec{a}$
(5)	pr	p	$\tilde{a}$	$\tilde{a}$
(6)	pr	c	$\tilde{a}$	$\tilde{a}$
(7)	pr	pr	$\vec{a}r$	$\vec{a}r$
(8)	pr	cr	$\vec{a}r$	$\vec{a}r$
(9)	c	p	$\vec{a}$	$\tilde{a}$
(10)	c	c	c	c
(11)	c	pr	$\vec{a}$	$\tilde{a}$
(12)	c	cr	c	c
(13)	cr	p	$\vec{a}$	$\vec{a}$
(14)	cr	c	c	c
(15)	cr	pr	$\vec{a}r$	$\vec{a}r$
(16)	cr	cr	cr	cr

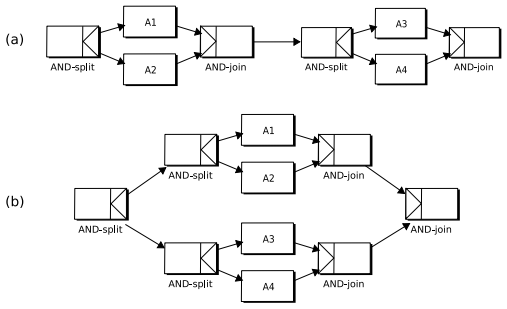


Fig. 6. Examples of workflow resulting in the composition of two TCWS

Several workflows can also result in the composition of two TCWSs, as shown in Figures 6.(a) and 6.(b). Table 5 presents the transactional behavioral property of sequential and concurrent execution of two TCWSs.

5.2.4 Summary

To summarize:

- A p or $\vec{a}$ WS can only be sequentially composed with a pr, $\vec{a}r$, or cr WS and can only be executed in parallel with a cr WS.
- A pr or $\vec{a}r$ WS can only be executed in sequential or in parallel with a pr, $\vec{a}r$, or cr WS.
- A c WS can be sequentially composed with any transactional WS but can only be executed in parallel with a c or cr WS.
- A cr WS can be executed in sequential or in parallel with any transactional WS.

TABLE 3
Transaction property of a sequential and a concurrent execution of a CWS with an elementary WS

	CWS	WS	$CWS;WS$	$CWS//WS$
(1)	$\vec{a}$	p	$\tilde{a}$	$\tilde{a}$
(2)	$\vec{a}$	c	$\tilde{a}$	$\tilde{a}$
(3)	$\vec{a}$	pr	$\vec{a}$	$\tilde{a}$
(4)	$\vec{a}$	cr	$\vec{a}$	$\vec{a}$
(5)	$\vec{a}r$	p	$\tilde{a}$	$\tilde{a}$
(6)	$\vec{a}r$	c	$\tilde{a}$	$\tilde{a}$
(7)	$\vec{a}r$	pr	$\vec{a}r$	$\vec{a}r$
(8)	$\vec{a}r$	cr	$\vec{a}r$	$\vec{a}r$
(9)	c	p	$\vec{a}$	$\tilde{a}$
(10)	c	c	c	c
(11)	c	pr	$\vec{a}$	$\tilde{a}$
(12)	c	cr	c	c
(13)	cr	p	$\vec{a}$	$\vec{a}$
(14)	cr	c	c	c
(15)	cr	pr	$\vec{a}r$	$\vec{a}r$
(16)	cr	cr	cr	cr

TABLE 4
Transaction property of a sequential and a concurrent execution of an elementary WS with a CWS

	WS	CWS	$WS;CWS$	$WS//CWS$
(1)	p	$\vec{a}$	$\tilde{a}$	$\tilde{a}$
(2)	p	c	$\tilde{a}$	$\tilde{a}$
(3)	p	$\vec{a}r$	$\vec{a}$	$\tilde{a}$
(4)	p	cr	$\vec{a}$	$\vec{a}$
(5)	pr	$\vec{a}$	$\tilde{a}$	$\tilde{a}$
(6)	pr	c	$\tilde{a}$	$\tilde{a}$
(7)	pr	$\vec{a}r$	$\vec{a}r$	$\vec{a}r$
(8)	pr	cr	$\vec{a}r$	$\vec{a}r$
(9)	c	$\vec{a}$	$\vec{a}$	$\tilde{a}$
(10)	c	c	c	c
(11)	c	$\vec{a}r$	$\vec{a}$	$\tilde{a}$
(12)	c	cr	c	c
(13)	cr	$\vec{a}$	$\vec{a}$	$\vec{a}$
(14)	cr	c	c	c
(15)	cr	$\vec{a}r$	$\vec{a}r$	$\vec{a}r$
(16)	cr	cr	cr	cr

TABLE 5
Transaction property of sequential and concurrent execution of two CWSs

	CWS_1	CWS_2	$CWS_1;CWS_2$	$CWS_1//CWS_2$
(1)	$\vec{a}$	$\vec{a}$	$\tilde{a}$	$\tilde{a}$
(2)	$\vec{a}$	c	$\tilde{a}$	$\tilde{a}$
(3)	$\vec{a}$	$\vec{a}r$	$\vec{a}$	$\tilde{a}$
(4)	$\vec{a}$	cr	$\vec{a}$	$\vec{a}$
(5)	$\vec{a}r$	$\vec{a}$	$\tilde{a}$	$\tilde{a}$
(6)	$\vec{a}r$	c	$\tilde{a}$	$\tilde{a}$
(7)	$\vec{a}r$	$\vec{a}r$	$\vec{a}r$	$\vec{a}r$
(8)	$\vec{a}r$	cr	$\vec{a}r$	$\vec{a}r$
(9)	c	$\vec{a}$	$\vec{a}$	$\tilde{a}$
(10)	c	c	c	c
(11)	c	$\vec{a}r$	$\vec{a}$	$\tilde{a}$
(12)	c	cr	c	c
(13)	cr	$\vec{a}$	$\vec{a}$	$\vec{a}$
(14)	cr	c	c	c
(15)	cr	$\vec{a}r$	$\vec{a}r$	$\vec{a}r$
(16)	cr	cr	cr	cr

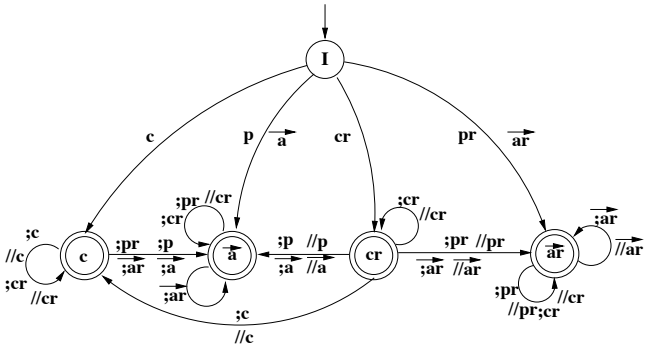


Fig. 7. Automaton modeling all possible TCWSs obtained by the selection process

Figure 7 represents the automaton modeling all possible TCWSs which could be obtained by the selection process detailed above. It contains five states. State I is the initial one. The final states $c, cr, \vec{a}$, and $\vec{a}r$ correspond to the transactional properties of a TCWS. When one of the final states is reached, a TCWS is obtained. The alphabet of the language accepted by the automaton is $\{ 'p' , 'c' , 'cr' , 'pr' , ';p' , '//p' , ';pr' , '//pr' , ';c' , '//c' , ';cr' , '//cr', ';\vec{a}', ';\vec{a}r', '//\vec{a}r' \}$. This alphabet represents the transactional properties of components (elementary WSs or CWSs) executed in sequence (;) or in parallel (//). For example, using a transition p or a transition $\vec{a}$ from the initial state (i.e. selecting a pivot WS or an atomic CWS for the first activity of the workflow), the following state is $\vec{a}$. Indeed, any sequential execution beginning by a pivot WS or an atomic TCWS can be atomic, but will never be able to be compensated. The next state could be again $\vec{a}$ using a transition $'//cr',';cr',';pr'$ or $';\vec{a}r'$, meaning that an atomic CWS can be sequentially composed with a retriable component but can only be executed in parallel with a compensatable retriable component.

5.2.5 Example

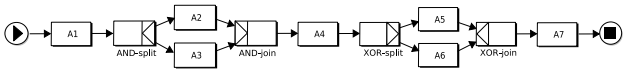


Fig. 8. An example of workflow

Figure 8 represents an example of an input workflow, where, for simplicity, each activity is performed by an elementary WS. If a pivot (p) WS_1, is assigned to the first activity A_1, then in the automaton of Figure 7, we go from initial state I to the state $\vec{a}$. Since the pattern following A_1 is an AND-split, then WS_1 will be executed in sequential with a composite service CWS_2 corresponding to a parallel execution of two elementary WSs (respectively assigned to activities A_2 and A_3). According to the automaton, to obtain a TCWS corresponding to $WS_1;CWS_2$, CWS_2 should be atomic retriable or compensatable retriable (using either transition $;\vec{a}r$ or $;cr$). Thus, as shown in Table 2, CWS_2 has to be

composed by either $(pr//pr)$ or $(pr//cr)$ to be atomic retriable. Similarly, to be compensatable retriable it has to be composed by $(cr//cr)$.

In both cases, the TCWS resulting from the composition of WS_1 with CWS_2 is atomic ($\vec{a}$). Continuing in the same way, using the automaton and the tables presented above, we can deduce that the elementary WSs assigned to activities A_4 to A_7 could only be pivot retriable (pr) or compensatable retriable (cr)

5.3 QoS-driven service selection

In our approach, the QoS service selection is embedded within the transactional service selection. The set of potential WSs for each workflow activity is restricted by the transactional requirement. Indeed, the selection of a WS for an activity depends on the transactional property of the WSs already assigned to the previous activities of the workflow. As a consequence, we can not use a global QoS selection process, such as the approach of [19], but we can use a local QoS optimization selection algorithm. In this paper, we use a local optimization selection algorithm as follows: according to the transactional requirements for each activity, a set of transactional WSs is selected, then a QoS-driven service selection, as defined in [6], is executed. For the selection of a WS for each activity, the system uses the classical Multiple Criteria Decision Making (MCDM) approach [20]. This selection is based on the weights assigned by the user to each quality criterion. A simple additive weighting technique is used to assign a quality score to each WS as follows: $Score(s_i) = \sum_j w_j q_{ij}$, where $w_j \in [0,1]$ is the weight assigned by the user to the quality criterion j such that $\sum_j w_j = 1$ and q_{ij} is the value of criterion j for service s_i. The WS with the maximal score is selected to execute activity a_i. If there are several services with maximal score, one of them is selected randomly.

In order to evaluate the QoS of a CWS, it is necessary to look into workflow composition patterns. A split pattern produces several execution paths composed of one or more activities. Similarly, a join pattern aggregates several execution paths in one path. In [21], the most frequently used path is considered in the aggregation functions for the XOR-split. In our approach, we select the path obtaining a CWS with the maximal score, according to the previous selection algorithm. If there are several paths with the maximal score, one of them is selected randomly.

5.4 Algorithm

The input of the TQoS-driven selection algorithm is a workflow WF composed of n activities and the output is a TCWS corresponding to a list of elementary WSs or CWSs (WSs for short) assigned to each activity of the input workflow. When a WS is assigned to an activity of the workflow, its transactional property influences the selection of the WS for the next activities – see Section 5.2.4. Thus, after each WS assignation in the algorithm,

Algorithm 1 TQoS-driven Selection Algorithm

Input: WF /* A workflow of n activities */
Output: $TCWS$ /* List of m assigned WS ($m < n$ in case of XOR-SPLIT) */,
QoS /* QoS property of $TCWS$ */,
$State$ /* Transactional property of $TCWS$ */
BEGIN
1: $State \leftarrow I$ /* Current state of the resulting TCWS */
2: $i \leftarrow 1$ /* Counter used for WF */
3: $j \leftarrow 1$ /* Counter used for TCWS */
4: $TCWS \leftarrow EmptyList$
5: $WS\_Set \leftarrow (S_p \cup S_{pr} \cup S_{\vec{a}} \cup S_{\vec{a}r} \cup S_c \cup S_{cr})$
6: $NTP \leftarrow Null$
7: **while** $IsOutput(WF, i) = false$ **do**
8: **if** $State \in \{\vec{a}, \vec{a}r\}$ **then**
9: $WS\_Set \leftarrow WS\_Set \cap (S_{pr} \cup S_{\vec{a}r} \cup S_{cr})$
10: **end if**
11: ASSIGN-NEXT(WF,i,$State$,$TCWS$,j,NTP,WS_Set) /* See Algorithm 2 */
12: $i \leftarrow i + 1$
13: **end while**
14: QoS=ComputeQoS($TCWS$)
15: **return** $TCWS$, $State$, QoS
END

the state of the resulting TCWS is evaluated and stored in a variable $State \in \{I, \vec{a}, \vec{a}r, c, cr\}$. A variable NTP stores the transactional property of the WS newly assigned: $NTP \in \{p, pr, c, cr, \vec{a}, \vec{a}r\}$.

Let us consider S_x a class of WS which is a collection of component WSs with common transactional property x but different non-functional (i.e. QoS) properties. For instance, S_p is the class of all the pivot WSs. S_{pr}, $S_{\vec{a}}$, $S_{\vec{a}r}$, S_c, and S_{cr} are the classes of all WSs that have respectively transactional property pr, $\vec{a}$, $\vec{a}r$, c, and cr. A variable WS_Set contains the set of the permitted WSs which can be assigned to the next non-assigned activity of the workflow. Its value depends on the value of the state ($State$) or of the TP of the WS newly assigned (NTP). For example, in a sequential pattern, if the current state is $\vec{a}$ or $\vec{a}r$, then only retriable WSs can then be assigned to the next non-assigned activities of the workflow: $WS\_Set = (S_{pr} \cup S_{\vec{a}r} \cup S_{cr})$. In an AND-split pattern, when a pivot or an atomic WS has been assigned to an activity, then only compensatable retriable WSs can be assigned to the following activities of the pattern: $WS\_Set = S_{cr}$.

The element (activities and patterns) of the workflow are numbered[1] from the left to the right and from the top to the bottom. To be analyzed in the TQoS-driven selection algorithm, each workflow is translated into an XML file. Then, we have defined several functions (e.g. `IsActivity`, `IsAndSplit`, `IsOutput`) which return the type (e.g. activity, AND-split pattern, output) of an element of the workflow. For example, function

1. The input activity is numbered by zero.

`IsActivity(WF,i)` returns true if the ith element of the workflow WF is an activity.

When the risk chosen by the user is 0 (see Section 5.1), then the TQoS-driven selection algorithm only assigns a compensatable or a compensatable retriable WS to each activity of the workflow. In this case, the algorithm only consists in choosing the compensatable or compensatable retriable WS having the best QoS for each activity (by calling function `GetBestofQoS(`$S_c \cup S_{cr}$`, i)` for each activity i of the workflow).

When the risk chosen by the user is 1, the algorithm is more complex (see Algo. 1). At the beginning, the $State$ of the resulting TCWS is I (initial state) and the set WS_Set is initialized with all the possible WSs, with transactional properties in $\{p, pr, \vec{a}, \vec{a}r, c, cr\}$ (see lines 1 and 5 of Algo. 1). Then, while the output pattern of the workflow is not reached (see line 7), a WS is assigned to the next non-assigned activity of the workflow (calling function `ASSIGN_NEXT` – see line 11) according to the current state of the current resulting TCWS. Therefore, if the current resulting TCWS is atomic or atomic retriable, then set WS_Set is restrained to retriable WS before calling function `ASSIGN_NEXT` (see lines 8 to 10).

Function `ASSIGN_NEXT` (see Algo. 2) analyzes the input workflow, WF, from the current position i. If the ith element of WF is an AND-split (resp. XOR-split) pattern (see lines 1 resp. 4 of Algo. 2), it calls function `ASSIGN_AND` (resp. `ASSIGN_XOR`). If the ith element of WF is a sequence pattern (see line 7) which is inside an AND-split or a XOR-split pattern, the function is recursively called after the update of set WS_Set, depending on the state of the current resulting TCWS (see lines 11 to 13). Otherwise, the ith element of WF is an activity (see line 17), then function `ASSIGN_NEXT` assigns a WS to the jth activity of the workflow having the best QoS and variables $State$ and NTP are updated (see lines 20 to 35). Note that function `GetBestofQoS(WS_Set, j)` returns the WS having the best QoS among the set of WSs which can execute the jth activity of WF (subset of WS_Set), and function `GetTPOf(`WS`)` returns the transactional property of the Web service WS.

Function `ASSIGN_AND` (see Algo. 3) assigns one WS to each activity of an AND-split pattern from top to down, managing a local set of permitted WSs for the activities of the pattern (see line 1). The activities of the first branch of the pattern are first assigned (by calling function `ASSIGN_NEXT` – see lines 3 and 4). Then, the activities of the other branches are assigned (see lines 7 to 16) after updating the local AND_WS_Set and depending on the state of the current resulting TCWS.

Function `ASSIGN_XOR` (see Algo. 4) assigns WS to the workflow activities of only one branch of the XOR-SPLIT pattern. To do this, all branches of the pattern are evaluated (see lines 7 to 24) and the WS corresponding to the best branch are assigned to the workflow (see lines 26 to 34).

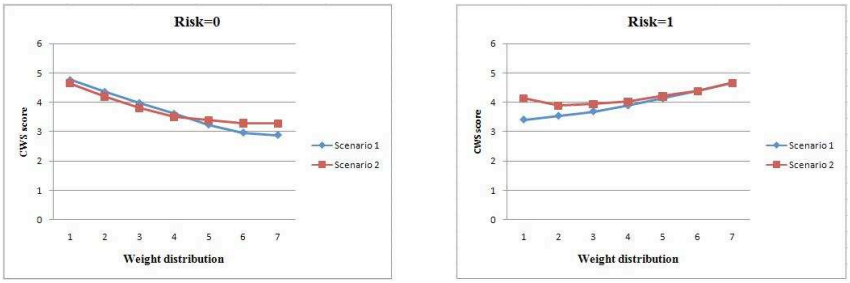
A function, `ComputeQoS(`$TCWS$`)`, is implemented to evaluate the QoS of the resulting TCWS from the QoS of its component WSs (see line 14 of Algo. 1).

Algorithm 2 ASSIGN-NEXT Algorithm

Input: WF, i, $State$, $TCWS$, j, NTP, WS_Set
Output: /* Updated variables */
$TCWS$, $State$, NTP, WS_Set
BEGIN
1: **if** $IsAndSplit(WF, i) = true$ **then**
2: $i \leftarrow i + 1$
3: ASSIGN-AND(WF,i,$State$,$TCWS$,j,NTP,WS_Set) /* See Algorithm 3 */
4: **else if** $IsXorSplit(WF, i) = true$ **then**
5: $i \leftarrow i + 1$
6: ASSIGN-XOR(WF,i,$State$,$TCWS$,j,NTP,WS_Set) /* See Algorithm 4 */
7: **else** /* Sequential pattern */
8: **if** $IsSequence(WF, i) = true$ /* A sequence inside an AND-SPLIT or a XOR-SPLIT */ **then**
9: $i \leftarrow i + 1$
10: **while** $IsEndOfSequence(WF, i) == false$ **do**
11: **if** $State \in \{\vec{a}, \vec{a}r\}$ **then**
12: $WS\_Set \leftarrow WS\_Set \cap (S_{pr} \cup S_{\vec{a}r} \cup S_{cr})$
13: **end if**
14: ASSIGN-NEXT(WF, i, $State$, $TCWS$, j, NTP, WS_Set)
15: $i \leftarrow i + 1$
16: **end while**
17: **else** /* $WF[i]$ is an activity */
18: /* Assign a WS to the jth activity */
19: /* The assigned WS is added in list TCWS */
20: $TCWS.Add(\ GetBestofQoS(WS\_Set, j)\ )$
21: $j \leftarrow j + 1$
22: /* Recall transactional property of the last assigned WS */
23: $NTP = GetTPOf(TCWS.LastAssignedWS())$
24: /* Compute the new state after assigning the jth WS */
25: **if** $State \in \{I, cr\}$ **then**
26: **if** $NTP \in \{p, \vec{a}\}$ **then**
27: $State \leftarrow \vec{a}$
28: **else if** $NTP \in \{pr, \vec{a}r\}$ **then**
29: $State \leftarrow \vec{a}r$
30: **else if** $NTP = c$ **then**
31: $State \leftarrow c$
32: **else**
33: $State \leftarrow cr$
34: **end if**
35: **else if** $State = c$ and $NTP \in \{p, pr, \vec{a}, \vec{a}r\}$ **then**
36: $State \leftarrow \vec{a}$
37: /* else $State$ does not change */
38: **end if**
39: **end if**
40: **end if**
END

6 Experimentation

In order to evaluate the behavior of our WS selection approach, experiments were conducted by implementing



(a) (b)

Fig. 9. Experimental results for risk 0 and risk 1 by varying price and duration weights

Algorithm 3 ASSIGN-AND Algorithm

Input: WF, i, $State$, $TCWS$, j, NTP, WS_Set
Output: /* Updated variables */
$TCWS$, $State$, NTP, WS_Set
BEGIN
$AND\_WS\_Set \leftarrow WS\_Set$ /* Local sets of permitted WS for the activities of the AND-split pattern */
/* Assigned WSs to the activities of the first branch of the AND-split pattern */
ASSIGN-NEXT(WF, i, $State$, $TCWS$, j, NTP, AND_WS_Set) /* See Algorithm 2 */
if $IsEndOfSequence(WF, i) == false$ **then**
  $i \leftarrow i + 1$ /* Go to the next branch */
end if
/* Assigned services to the activities of the other branches of the AND-split pattern */
while $IsAndJoin(WF, i) = false$ **do**
  /* Update of AND_WS_Set depending on the property of the previous activity in the parallel pattern */
  if $State = \vec{a}$ **then**
    /* A pivot or an atomic WS could only be executed in parallel with a cr one */
    $AND\_WS\_Set \leftarrow S_{cr}$
  else if $State = c$ **then**
    /* A compensatable WS could only be executed in parallel with a c or cr one */
    $AND\_WS\_Set \leftarrow S_c \cup S_{cr}$
  else if $State = \vec{ar}$ **then**
    /* A pivot or an atomic retriable WS could only be executed in parallel with a retriable one */
    $AND\_WS\_Set \leftarrow S_{pr} \cup S_{\vec{ar}} \cup S_{cr}$
  end if
  ASSIGN-NEXT(WF, i, $State$, $TCWS$, j, NTP, AND_WS_Set) /* See Algorithm 2 */
  $i \leftarrow i + 1$
end while
END

the proposed TQoS algorithm on a PC Core II 1.07GHz with 1014Mo RAM, Windows Vista, Java 2 Enterprise Edition V1.5.0. The experiments involved composite services varying the number of activities and varying the number of Web services. Experimentations were done over the following four scenarios. In the first one, we assume that the execution price of a *compensatable* WS is more expensive than a *pivot* or *atomic* one. Indeed, the former provide additional functionality in order to guarantee that the result can be undone. Similarly, we believe that a *retriable* WS has an execution duration higher than a *non-retriable* one. Indeed, the former provides additional operation in order to guarantee that it successfully finishes after a finite number of invocations. The second scenario is a variant of the first one in which the intervals of values for the criteria price and duration overlap of 20%. A random generation of different WSs that can implement the activities of workflow WF of Figure 8 was accomplished as follows: for each scenario and for each activity, we randomly generate from 1 to 10 WSs for each of the transactional property $\{p, pr, c, cr, \vec{a}, \vec{ar}\}$. For each WS, we randomly generate QoS criteria values according to the scenario. For example, Table 6 shows the different set of values considered for each QoS criterion depending on transactional service property for the first scenario. In the third scenario, the number of activities varies from 1 to 62. Also, in the last scenario, the number of candidate component services varies per activity and per transactional property from 1 to 8 with steps of 2. For these two last scenarios, we randomly generate QoS criteria values of services according to the following: $q_{ep}(s)$ and $q_{ed}(s)$ between 1 and 100, $q_r(s)$ between 1 and 6, $q_{sr}(s)$ and $q_a(s)$ between 0 and 1.

We apply our selection algorithm considering both levels of risk. For the first two scenarios, we consider weights assigned by the user in such a way that price and duration constraints have always 60% of the total weight. With this condition, we execute the selection process for the weight distribution shown in Table 7. This experiment was executed 10 times for both scenarios. Figure 9.(a) shows the score results obtained for the risk level 0 and for the two scenarios with different weights over the criteria. Figure 9.(b) shows the score results obtained for the risk level 1 and for the two scenarios with different weights over the criteria. As depicted in Figure 9, for both scenarios, the more important the price criteria to the user, the better a composition with risk 1 compared to a composition with risk 0. Additionally, the more important the duration criteria to the user,

Algorithm 4 ASSIGN-XOR Algorithm

Input: WF, i, $State$, $TCWS$, j, NTP, WS_Set
Output: /* Updated variables */
$TCWS$, $State$, NTP, WS_Set
BEGIN
/* Informations of the "best" branch of the XOR pattern */
$Best\_TCWS \leftarrow EmptyList$
$Best\_State \leftarrow State$
$Best\_NTP \leftarrow NTP$
$Best\_WS\_Set \leftarrow WS\_Set$
$Best\_QoS \leftarrow 0$
$nb \leftarrow 0$ /* Number of analysed branches in the pattern */
while $IsXorJoin(WF, i) = false$ **do**
/* For each branch, recall the $State$ of the branch, the WS assigned to the activities of the branch and so on. */
$Branch\_TCWS \leftarrow EmptyList$
$Branch\_State \leftarrow State$
$Branch\_NTP \leftarrow NTP$
$Branch\_WS\_Set \leftarrow WS\_Set$
$k \leftarrow 1$ /* Index for the activities of the branch */
/* Assigned WSs to the activities of the current branch */
ASSIGN-NEXT(WF,i,$Branch\_State$, $Branch\_TCWS$, k, $Branch\_NTP$,$Branch\_WS\_Set$) /* See Algorithm 2 */
if $IsEndOfSequence(WF, i) == false$ **then**
$i \leftarrow i + 1$ /* Go to the next branch */
end if
$Branch\_QoS \leftarrow ComputeQoS(Branch\_TCWS)$
/* If the QoS of the current branch is better of the QoS of the previous branches, recall the new "best" branch */
if $Branch\_QoS > Best\_QoS$ **then**
$Best\_TCWS \leftarrow Branch\_TCWS$
$Best\_State \leftarrow Branch\_State$
$Best\_NTP \leftarrow Branch\_NTP$
$Best\_WS\_Set \leftarrow Branch\_WS\_Set$
$Best\_QoS \leftarrow Branch\_QoS$
end if
$nb \leftarrow nb + 1$
end while
/* Update TCWS with the WSs of the "best" branch which contains $length()$ assigned WSs */
for $k = 1$ to $k = Best\_TCWS.length()$ **do**
$TCWS[j] \leftarrow Best\_TCWS[k]$
$j \leftarrow j + 1$
end for
$State \leftarrow Best\_State$
$NTP \leftarrow Best\_NTP$
$WS\_Set \leftarrow Best\_WS\_Set$
END

the better a composition with risk 0 compared to a composition with risk 1.

For the third and fourth scenarios, weights were generated randomly. In the third scenario, different workflows were generated varying the number of activities from 1 to 62. For each workflow and per risk level, we executed the selection process 10 times and computed the average computation cost. Figure 10 shows computation cost (in seconds) of selecting services for composite services. In this experiment, the number of candidate services per activity and per transactional property varies between 1 and 2 (in total 712 services were generated). We observe that the computation cost remains constant and do not increase with the number of activities. This is normal since for each activity at the beginning of the workflow, the set of candidate services is much higher than the one for an activity at the end of the workflow. For 17 activities and 1 to 2 candidate services per activity and per transactional property (in total 712 services), the computation cost for risk 1 ($0,24$ second) is almost the same as for risk 0 ($0,23$ second).

TABLE 6
Set of values for each QoS criterion

Criteria	p	pr	c	cr	$\vec{a}$	$\vec{a}$r
$q_{ep}(s)$	$[0, 60]$	$[0, 60]$	$[60, 100]$	$[60, 100]$	$[0, 60]$	$[0, 60]$
$q_{ed}(s)$	$[10, 60]$	$[60, 100]$	$[10, 60]$	$[60, 100]$	$[10, 60]$	$[60, 100]$
$q_r(s)$	$[1, 6]$	$[1, 6]$	$[1, 6]$	$[1, 6]$	$[1, 6]$	$[1, 6]$
$q_{sr}(s)$	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$
$q_a(s)$	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$

TABLE 7
Weight distribution

Criteria	(1)	(2)	(3)	(4)	(5)	(6)	(7)
$q_{ep}(s)$	0	10	20	30	40	50	60
$q_{ed}(s)$	60	50	40	30	20	10	0
$q_r(s)$	10	10	10	10	10	10	10
$q_{sr}(s)$	15	15	15	15	15	15	15
$q_a(s)$	15	15	15	15	15	15	15

In the fourth scenario, different sets of services were generated varying the number of services per activity and per transactional property from 1 and 8 with steps of 2. For each set and per risk level, we executed the selection process 10 times and computed the average computation cost. Figure 11 shows computation cost (in seconds) of selecting services for composite services. In this experiment, four sets were generated. The first one contains 712 candidate services (the number of services per activity and per transactional property is between 1 and 2). The second set contains 1692 candidate services (the number of services per activity and per transactional property is between 3 and 4). The third set contains 2651 candidate services (the number of services per activity and per transactional property is between 5 and 6). Finally, the last set contains 3602 candidate services (the number of services per activity and per transactional property is between 7 and 8). For both levels of risk, the computation cost increases when the number of candidate services increases. For a workflow with 40 activities and from 30 to 36 candidate services per activity (in total 2651 services), the computation cost of risk 1 is $1,15$ seconds, almost the same as risk 0 (1 second).

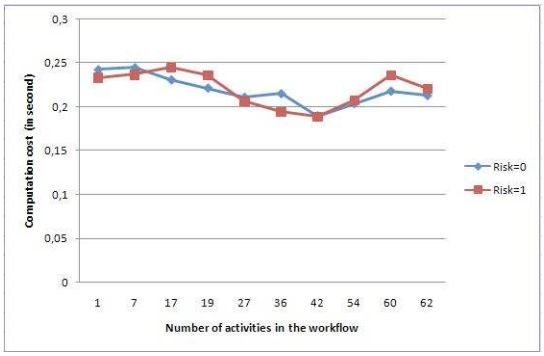


Fig. 10. Computation cost by varying the number of activities

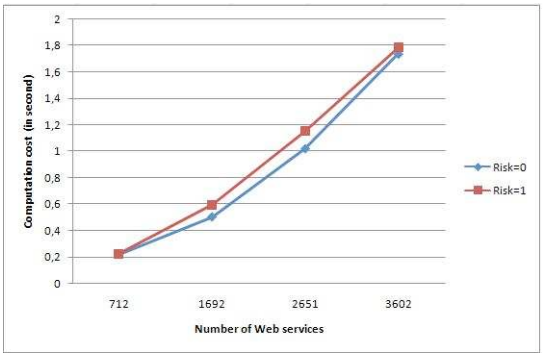


Fig. 11. Computation cost by varying the number of services

7 Related work

Recently, several QoS-aware Web services selection mechanisms have been developed. Menasce [4] studies the QoS of component WSs in terms of cost and execution time. He employs probability techniques to measure the cost and execution time of component WSs by considering different execution scenarios. This study helps in selecting appropriate component WSs for Web service composition. Jaeger *et al.* [3] propose a mechanism for composite WSs with pattern-based QoS aggregation. The QoS aggregation is used to verify that a set of WSs satisfies the QoS requirement for the selected composite WS. Zeng *et al.* [6] propose two QoS-driven service selection approaches: local optimization and global planning. Recently, Kokash [22] modifies the Zeng *et al.* method in order to consider the probability of component WS failures, their response time and the execution cost along with the structure of composite graph. In [7], the authors propose a solution for WS selection taking into account one or more global constraints set by users. In [5], the authors use an approach allowing service selection with best results with slow selection time, by using dynamic programming or good enough results with fast selection time. However, none of these approaches takes into account the transactional behavior of the composite WS. In our proposition, the QoS-driven service selection is embedded within the transactional service selection. The transactional-driven service selection determines, in each step, the set of WSs to be considered for QoS selection depending on their transactional properties.

Coordinating a set of activities to achieve a common goal having a transactional behavior has been tackled by workflow systems, by transactional protocols and by Advanced Transactional Models (ATM).

In [23], the authors address the support of distributed transactions in workflow management systems based on processes by using exception handlers. They do not consider the coordination of black boxes, such as WSs, with dissimilar capabilities with respect to their transaction behavior. [9] extends workflows patterns in order to consider the transactional behavior in case of failures and recovery. The transactional patterns can be used by the designer to construct a TCWS. In [8], Bhiri *et al.* present an approach specifying relaxed atomicity requirements for composite WSs based on Acceptable Termination States (ATS) model and transactional rules to validate a given composite WS with respect to defined Transactional Requirements (TR). In [11], Montagut *etal.* propose a selection mechanism enabling the automatic design of transactional composite WSs by using the ATS model. This mechanism has been extended to manage composite WS execution using context-driven policies in [13] or adaptive transactional protocol in [24]. The drawback of these approaches is the definition of all the ATS by the user, which is neither simple nor scalable; the bigger the workflow the bigger is the number of ATS to be defined by the user. Moreover, the mechanism does not take into account any QoS criterion in the selection process. In [10], the transactional behavior of CWS in presence of transactional component WSs are studied but without taking into account the QoS and no selection algorithm is proposed. Moreover, the authors do no specify the behavior of retriable WSs in case of compensation or abortion of a part of the workflow. The authors of [2] propose several transactional composition operators for WSs and evaluate the QoS of the composite WS, considering the abortion cases. They only analyze the transactional effects on QoS, without ensuring the optimal QoS requirement. Their approach does not help to design a composite WS ensuring, not only a correct execution, but also an optimal QoS. In our approach, we propose a model for the selection of transactional WSs with the best QoS. Thus, our approach not only fulfills the global transactional requirement but also guarantees locally the best QoS component WS.

Transactional protocols, such as standards like BTP [25] and WS-TXM [26], propose two-phase centralized orchestration of composite WS. In [27], the authors use an extension of the two-phase coordination protocol. In addition, their approach allows the user to express maximality and minimality constraints over the set of WSs expected to the validation phase. Several approaches use ATM to implement transactional behavior for WSs. In [28], the authors present a multi-level model for WS composition that does not support users' constraints. In [29], the authors propose an approach based on

open nested transaction model in a peer-to-peer context allowing users to express their constraints over the set of composite WSs. In [30], the authors propose a framework based on Sagas nested transactions. In transactional protocols and ATM approaches, the execution control is explicitly defined within the application logic which are difficult to maintain and hardly adaptable to different application requirements.

Other approaches use workflows and components' transactional properties to test the transactional property of the CWS. In [31], the authors propose a mechanism based on a set of mediators to gather functionally similar but transactionally different WSs. Their approach allows resolving the heterogeneity among WSs. The authors of [12] use contracts to express the transactional behavior of a given CWS. To our knowledge, our approach is the only one that selects appropriate transactional component WSs to construct a TCWS that satisfies the user requirements. In our approach, the designer focuses on the construction of the desired functionalities of the applications leaving the complex process of selection of WSs based on their QoS and transactional behavior to the composition manager.

8 Conclusion

In this paper, we have presented a TQoS-driven approach. It consists of a Web service selection approach supporting transactional and quality driven WS composition. The selection of the component WSs is done by matching the WSs properties with the user's desires. More precisely, the selection is realized depending on transactional and QoS user requirements. The former is established by means of a risk notion that indicates if the results can be compensated or not. The latter is expressed as a weight over each QoS criterion.

Our contribution is twofold. On the one hand, the composition manager selects WSs according to the quality and transactional behavior of the application, leaving to the user to focus on the construction of the desired functionalities. On the other hand, we proposed and formally analyzed a selection algorithm based on the workflow patterns and the transactional properties of the component WSs (elementary or composite).

In this paper, five QoS criteria (execution price, execution duration, reputation, successful execution rate and availability) have been used and a local QoS-driven service selection related to these criteria has been chosen. However, other properties could have been taken into account, such as performance-related ones (e.g. machine resources), as done for example in [32]. The local QoS-aware selection is based on MCDM approach, however any other approach could have been used. The only constraint is to use a local optimization process in order to chose the WS having the best QoS among a set of potential WSs resulting from the transaction-aware selection process.

In the experimentation, in order to give a semantic meaning to the risk notion, we have considered two scenarios where the execution duration and execution price of a WS depend on additional operations required to guaranty their transactional properties. We used the risk notion for these scenarios. Under these conditions the implementation shows that the QoS of TCWS is in conformity with the user preferences. If the execution price criterion is important to the user (i.e. price minimum), then the better solutions are the ones with the lowest level of risk. A contrario, if the execution duration criterion is more important to the user (i.e. execution time minimum), then the riskier solutions are the best ones. The results also show that risk 0 is equivalent to risk 1 if compensatable services do not cost more than the others. Moreover, we have evaluated the scalability of our TQoS algorithm. In fact, the experimental results, of two other scenarios, show that the number of activities does not affect the computation cost of the algorithm since the selection is done incrementally and therefore, the sets of candidate services for beginning activities is much bigger than the ones for ending activities. More experiments are needed to consider different scenarios and compare the performance of our algorithm with related ones.

Currently, we are studying by one side other risk levels by relaxing the transactional properties of a CWS and considering penalty. By the other side, we are studying the execution step of the Web service composition. This step can be done using for example a hierarchical execution model [29]. We are particularly interested on finding dynamic selection solutions which takes into account failures or dynamic changes (e.g. a component WS becomes unavailable or the QoS of one of the component WS changes significantly). More particulary, we are interested in automatic selection where the user is relieved as much as possible from the composition and execution processes. Due to, since in our approach after each WS selection, we compute and record the current transactional property of the resulting composite WS, a component WS substitution, in case of failure or dynamic changes, can be done. This is possible by analyzing the transactional property of the TCWS obtained before the failed component. This information can also be used to complete context-policies. Our future work will focus on these aspects.

References

[1] Q. Yu, X. Liu, B. Athman, and B. Medjahed, "Deploying and managing Web services: issues, solutions, and directions," *The VLDB Journal*, vol. 17, no. 3, pp. 537–572, 2008.

[2] A. Liu, L. Huang, and Q. Li, "QoS-Aware Web Services Composition Using Transactional Composition Operator," *Proc. of the 7th Int. Conf. Advances in Web-Age Inf. Manag. (WAIM'2006), LNCS 4016*, pp. 217–228, June 2006.

[3] M. C. Jaeger, G. Roec-Goldmann, and G. Muehl, "QoS Aggregation for Web Service Composition using Workflow Patterns," *Proc. of the 8th IEEE Int. Enterprise Distributed Object Comp. Conf. (EDOC'04)*, pp. 149–159, 2004.

[4] D. Menasce, "Composing Web Services: A QoS view," *IEEE Internet Computing*, vol. 6, no. 8, pp. 88–90, December 2004.

[5] B. Wu, CH.Chi, and S.Xu, "Service Selection Model Based on QoS Reference Vector," *Proc. of theIEEE Congress on Services (Services 2007)*, pp. 270–277, 2007.

[6] L. Zeng, A. N. B. Benatallah, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web services Composition," *IEEE Trans. on Software Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[7] W. Zhang, Y. Yang, S. Tang, and L. Fang, "QoS-driven Service Selection Optimization Model and Algorithms for Composite Web Services," *Proc. of the 31st Annual Int. Computer Software and Applications Conf. (COMPSAC 2007), IEEE Computer Society*, vol. 2, pp. 425–431, 2007.

[8] S. Bhiri, O. Perrin, and C. Godart, "Ensuring required failure atomicity of composite Web services," *Proc. of the 14th Int. Conf. on WWW*, pp. 138–147, 2005.

[9] ——, "Extending workflow patterns with transactional dependencies to define reliable composite Web services," in *Proc. of AICT-ICIW*, Washington, DC, USA, 2006, p. 145.

[10] L. Li, C. Liu, and J. Wang, "Deriving Transactional Properties of Composite Web Services," *Proc. of the IEEE Int. Conf. on Web Services (ICWS 2007)*, pp. 631–638, July 2007.

[11] F. Montagut, R. Molva, and S. T. Golega, "Automating the Composition of Transactional Web Services," *Int. J. Web Service Res.*, vol. 5, no. 1, pp. 24–41, 2008.

[12] A. Portilla, G. Vargas-Solar, C. Collet, J.-L. Zechinelli-Martini, and L. Garca-Bauelos, "Contract Based Behavior Model for Services Coordination," in *Proc. of the 3rd Int. Conf. on Web Inf. Sys. and Technologies (WEBIST 2007)*, Barcelona (Spain), March 2007, pp. 109–123, lecture Notes in Business Inf. Proc. 8 Springer 2008.

[13] Z. Maamar, N. Narendra, D. Benslimane, and S.Subramanian, "Policies for Context-Driven Transactional Web Service," in *19th Int. Conf. on Advanced Information Systems Engineering (CAiSE'07)*, Trondheim, Norway, 2007, pp. 249–263.

[14] J. E. Haddad, a. G. R. M. Manouvrier, and M. Rukoz, "QoS-Driven Selection of Web Services for Transactional Composition," in *Proc. of the IEEE Int. Conf. on Web Services (ICWS 2008)*, 2008, pp. 653–660.

[15] S. Mehrotra, R. Rastogi, H. Korth, and A. Silberschatz, "A transaction model for multidatabase systems," *Proc. of the Int. Conf. on Distributed Computing Sys.*, pp. 56–63, June 1992.

[16] Yet Another Workflow Language - http://www.yawl-system.com/.

[17] Business Process Execution Language for Web Services - www.ibm.com/developerworks/library/ws-bpel/.

[18] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros., "Advanced Workflows Patterns," *Proc. of the 7th Int. Conf. on Cooperative Information Systems (CoopIS 2000), LNCS 1901*, pp. 18–29, 2000.

[19] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut", "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 281 – 308, 2004.

[20] M. Zeleny, *Multiple Criteria Decision Making*. McGraw-Hill, 1982.

[21] M. C. Jaeger, G. Muehl, and S. Golze, "QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms," *OTM Confederated Int. Conf. CoopIS, DOA, and ODBASE 2005, LNCS 3760*, pp. 646–661, October 2005.

[22] V. D. N. Kokash, "Evaluating Quality of Web Services: A Risk-Driven Approach," *Business Inf. Sys. (BIS'2007), LNCS 4439*, pp. 180–194, 2007.

[23] C. Hagen and G. Alonso, "Exception Handling in Workflow Management Systems," *IEEE Trans. Softw. Eng.*, vol. 26, no. 10, pp. 943–958, 2000.

[24] F. Montagut, R. Molva, and S. T. Golega, "The Pervasive Workflow: A Decentralized Workflow System Supporting Long-Running Transactions," *The Pervasive Workflow: A Decentralized Workflow System Supporting Long-Running Transactions. IEEE Transactions on Systems, Man, and Cybernetics - Part C*, vol. 38, no. 3, pp. 319–333, 2008.

[25] OASIS Business Transaction Protocol - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=business-transaction.

[26] WS-TXM from OASIS WS-CAF - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf.

[27] M. Fauvet, H. Duarte, M. Dumas, and B. Benatallah, "Handling transactional properties in web service composition," in *Proc. of the Int. Conf. WISE 2005*, New York City (USA), 2005, pp. 273–289.

[28] K. Vidyasankar and G. Vossen, "A multilevel model for web service composition," in *Proc. of the IEEE Int. Conf. on Web Services (ICWS 2004)*, 2004, pp. 462–469.

[29] J. E. Haddad, M. Manouvrier, and M. Rukoz, "A Hierarchical Model for Transactional Web Service Composition in P2P Networks," in *Proc. of the IEEE Int. Conf. on Web Service (ICWS 2007)*, 2007, pp. 346–353.

[30] N. B. Lakhal, T. Kobayashi, and H. Yokota, "FENECIA: failure endurable nested-transaction based execution of composite Web services with incorporated state analysis," in *The VLDB Journal*, vol. 18, 2009, pp. 1–56.

[31] P. F. Pires, M. R. F. Benevides, and M. Mattoso, "Building Reliable Web Services Compositions," in *Web Databases and Web Services, LNCS 2593*, 2003, pp. 59–72.

[32] J. Jin and K. Nahrstedt, "On exploring performance optimizations in web service composition," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX Int. Conf. on Middleware*, 2004, pp. 115–134.