An Approach for Transactional QoS-driven Service Composition

Qingsheng ZHU, Quanwang WU †, Gang DAI, Mingqiang ZHOU

1College of Computer Science, Chongqing University, Chongqing 400044, China

Abstract

In the Service Oriented Architecture (SOA) there are always over one service candidates corresponding with the criterion of the service request, which are developed by different organizations and offer diverse, Quality of Service (QoS) values and transactional properties. 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 QoS remains an important challenge. However, the problem is generally addressed from the QoS side or from the transactional side separately by the conventional approaches. This paper tackles this issue by proposing a selection algorithm for service composition, which maximizes the QoS while meeting the transactional requirement. At last experimental results are presented.

Keywords: SOA; Web Service Composition; Transaction; Quality of Service; Local Optimization.

1. Introduction

Web Services (WSs) based on open XML standards like SOAP, WSDL, and UDDI are widely used for integration purposes within enterprises. Beyond this, Web Services have the potential to be composed to cross-organizational workflows [1]. Due to their loosely-coupled nature, Web Services hosted by external providers can be integrated at runtime. This vision aims at dynamic ad-hoc collaborations between different business partners. Thus, WSs are the most efficient implementation of service oriented architectures (SOA). Widely available and standardized Web services make it possible to realize Business-to-Business Interoperability by inter-connecting Web services provided by multiple business partners according to some business process. This practice is known as web service composition.

With the increasing number of WSs with similar or identical functionality, the non-functional properties of a web service have become more and more significant in web service composition. The non-functional properties include the quality of service (QoS) attributes (e.g. price, availability, response time, and reputation) and the transactional property. Those non-functional properties are crucial for companies to meet the requirements of customers.

Though many works have been presented on service composition [2] [3] [4] [5], few of them integrates QoS-driven and transaction-driven service composition at the same time. In this paper, the approach is transactional QoS-driven Web service composition, which integrate QoS-driven and transaction-driven approach together. In our approach, the QoS-driven service selection is local optimized and placed after the

† Corresponding author.

Email addresses: wuquanwang1@163.com (Quanwang WU).
transactional service selection, which is implemented by the classical Multiple Criteria Decision Making (MCDM). The transaction-driven service selection determines the set of WSs to be considered for QoS selection depending on their transactional properties.

2. Related Work

Many researches have been carried out on QoS-driven service composition. Zeng et al. [2] propose two QoS-driven service selection approaches: local optimization which is similar to the approach in this paper and global optimization which operates by using linear programming. As the time complexity of linear programming is exponential, Gerardo et al. [3] and Rainer et al. [5] improve the performance by employing the genetic algorithm and heuristic algorithm, respectively. M. C. Jaeger et al. [7] 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. However, none of these approaches takes the transactional behavior of the composite WS into account.

Making a set of WSs to have a transactional behavior has been tackled by workflow systems or by transactional protocols. In [8], the authors address the support of distributed transactions in workflow management systems based on processes by using exception handlers. Transactional protocols, such as standards like BTP [9], propose two-phase centralized orchestration of composite WS. In [10], the authors use an extension of the two-phase coordination protocol. Likewise, the above approaches do not take quality of service into account.

To our knowledge, only A. Liu et al. 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 [11]. However, as the composition is not automatically decided by the system, it can not work well in a dynamic environment.

3. Web Service Properties

WS are described in terms of functional capabilities as well as transactional properties and QoS properties. As the functional capabilities can be addressed by UDDI and BPEL, thus, in this paper, we only discuss the last two types of properties at Section 3.2 and 3.3, respectively. Above all, the model of composite WS is presented.

3.1. Composite Web Service Description

A composite WS can be specified as a collection of individual services. In this paper we use the state chart to model the composite WS. A state chart is made up of states and transitions. Transitions of a state chart are labeled with events, conditions, and operations. States can be basic or compound. Basic states (also called tasks or activities in the sequel) are labeled with an operation name of a given service. Intuitively, when the basic state is entered, the operation that labels this state is invoked over one of the services belonging to the designated service. Compound states come in two flavors: OR-states and AND-states. An OR-state contains a single region whereas an AND-state contains several regions executed concurrently.

A simplified state chart W specifying a “Travel Planner” composite Web service is depicted in Fig. 1. In this example, a search for attractions is done in parallel with a flight and an accommodation booking. After
the searching and booking operations complete, the distance from the hotel to the accommodation is computed, and either a car or a bike rental service is invoked.

Fig. 1 State Chart of a “Travel Planner” Composite Service

3.2. Web Service Transactional Properties

In a composition where several component WSs interact, unexpected behavior from a component WS might, not only lead to its failure, but also may bring negative impact on all the other participants of the composition. The execution of a composite web service requires transactional properties (TP) so that the overall consistency is ensured.

The transaction mechanism is fairly mature in relation database management system (RDBMS), which is employed for failure recovery and concurrent control. However, in the SOA environment where anyone can publish and invoke any service at any time, it is almost impossible to operate concurrent control. Thus the transaction mechanism for service composition is specifically for failure recovery.

In the composite service of Fig.1, when t1, t2 and t3 service successfully execute and the process comes into t4. If t4 fails, the composite service will enter an inconsistent state, which should be avoided. If there is an extra service, which can roll back the effects of previous services, then the consistency is ensured.

Based on the above analysis, we define the following properties for both individual and composite WS: compensable (c), atomic (a), non-atomic (ã) and recovery-irrelevant (i):

**Definition 1 (Compensable WS).** An individual WS s is compensable (c) only if there is a WS, which can semantically rollback the execution of s. If all its component WS is compensable, the composite web service can be called compensable. Take the "Hotel Booking" service in the above composite service as an example: If there is a "Hotel Book Canceling" service, then the "Hotel Booking" service is compensable

**Definition 2 (Atomic WS).** A WS is atomic if once it completes successfully, their effect remains forever and once it fails, it has no effect at all. Thus, all single WS are atomic. A composite WS is atomic if once one component WS fails, all previously successful component WSs can be roll backed. In the following, a is used to indicate an atomic WS while ã is used to indicate a non-atomic one.

**Definition 3 (Recovery-irrelevant WS).** A WS is recovery-irrelevant if it has no effect at all without rollback when it fails and no modification is done to the corresponding system when it completes successfully. In other words, it does not need recovery. A search operation is a typical WS in this kind.

**Definition 4 (Transactional WS).** A transactional web service is a WS whose transactional property is in \{a, c, i\}.

3.3. Web Service QoS Properties

Web services will typically be grouped together in a single community. To differentiate the members of a community during service selection, their QoS properties need to be considered. For this purpose, we adopt
a Web service quality model based on a set of quality criteria. Although the adopted quality model has a limited number of criteria (for the sake of illustration), it is extensible: new criteria can be added without fundamentally altering the service selection techniques built on top of the model.

We consider five generic quality criteria for individual services:

1) Execution Price: which is the fee that a requester has to pay for invoking service s. Providers either advertise the execution price of their operations, or provide means for potential requesters to inquire.

2) Response Time: that measures the expected delay between the moment when s is invoked and when the results are received. The response time is the sum of the processing time and the transmission time.

3) Reputation: this is a measure of trustworthiness of WS. Generally this measure is defined as the average ranking given to the service by end users.

4) Successful execution rate: it is the probability that WS responds correctly to the user request.

5) Availability: it is the probability that WS is accessible.

4. Transactional QoS-driven Service Composition

In this section, we are interested in properly selecting component WS based on the input of workflow and user preferences, in order to obtain a transactional composite web service which maximizes the QoS criterion. The process consists of two steps: transaction-driven service selection and QoS-driven service selection. The first step determines a set of WS which can compose a transactional WS. From that set of WS, the second step select out a composite web service who’s QoS is the largest.

4.1. Transaction-driven Service Selection

The transactional property of a composite WS highly depends on the transactional properties of its component WS and on the structure of the workflow. The set of transactional property for a WS is \{a, ā, c, i\}. As the property recovery-irrelevant does not have an impact on the transactional property of a composite WS, we will not take it into account in the transaction-driven service selection.

At first, we suppose a workflow containing only two activities, and then we present a generalization of the transactional driven service selection to a workflow of n activities. For the two activities, we first consider the assignation of two WSs to the activities of a sequential pattern. Then, we consider the parallel pattern (AND-pattern). 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 WS and then the WS transactional property corresponds to the transactional property of that WS.

If the transactional property of one of the two WSs is ā, then no matter what the transactional property of the other WS is and no matter how they are combined, the transactional property of the composite WS is non-atomic (ā).

Without considering property i and property ā, the set of all possible combinations for two web services is \{c + c, c + a, a + c, a + a\}.

If the two web services are sequential, the analysis to the transactional property of the composite WS is as follows (in the sequel, the left WS is run first):

1) c + c: no matter which service fails, the composite WS can be rolled back. As a result, the composite web service is still compensable.

2) c + a: if the right WS fails, the left finished WS can be rolled back. If the left WS fails, it has no effect
at all, either. Therefore, the composite web service is atomic.

3) \(a + c\): if the right WS fails, the left finished WS can not be rolled back so the composite WS is non-atomic.

4) \(a + a\): the composite WS is non-atomic due to the same reason as above.

If the two web services are parallel, the analysis is as follows:

1) \(c + c\): no matter which service fails, the other WS can be rolled back, so the composite WS is compensable.

2) \(c + a\): in parallel, \(c + a\) is the same as \(a + c\). If the compensable WS fails, whether the atomic WS is executed is uncertain. In the case that atomic WS is successfully executed, it can’t be rolled back, which makes the composite WS non-atomic.

3) \(a + a\): the composite WS is non-atomic based on the same reason as above.

We can summarize the above analysis in the Table.1.

<p>| Table.1 Transaction Property of Two WSs in a Sequential and a Concurrent Execution |
|---------------------------------|-----------------|-----------------|-----------------|</p>
<table>
<thead>
<tr>
<th>WS1</th>
<th>WS2</th>
<th>Sequence</th>
<th>And</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

At present, we can generalize the conclusion to n WSs:

1) If the component WS are all compensable, the composite WS is still compensable.

2) If one of the component WS is atomic and it is at the end of the sequence, the composite WS is atomic. Otherwise, the composite WS is non-atomic.

3) If over one component WSs are atomic, the composite WS is non-atomic.

To ensure the composite web service transactional, the atomic WS is only permitted at the end of sequence.

The algorithm for transaction-driven service selection is as follows.

**Algorithm 1** transaction-driven service selection Algorithm

**Input**: Workflow, WS Set /* WS Set is the set of WS candidates for all activities in Workflow */

**Output**: New Set /* from this set, transaction is ensured */

**BEGIN**

1: \(i \leftarrow 1\) /* Counter used to identify which activity is in the Workflow */
2: \(\text{New Set} \leftarrow \text{EmptyList}\)
3: \(\text{while } \text{isLastActivity(Workflow, } i) = \text{false } \text{do}\)
4: \(\text{Permitted Set} = \text{getCSet(WS Set, Workflow, } i)\)
5: \(\text{New Set} = \text{New Set} \cup \text{Permitted Set}\)
6: \(i \leftarrow i + 1\) /* Go to the next activity */
7: \(\text{end while}\)
8: \(\text{if } \text{isSequential(Workflow,} i) = \text{true } \text{then}\)
9: \(\text{Permitted Set} = \text{getAllSet(WS Set, Workflow, } i)\) /* the atomic WS is permitted here */
10: \(\text{else}\)
Permitted Set = getCSet(WS Set, Workflow, i)

end if

New Set = New WS Set \cup \text{Permitted Set}

return New Set

END

In the algorithm, isLastActivity is a function returns whether the i-th activity is the last activity in the Workflow, getCSet returns the set of compensable WS for the i-th activity in Workflow and isSequential judges whether the i-th activity is sequential in the workflow.

Finally, the algorithm returns the new WS set which can support the transaction.

4.2. QoS-driven Service Selection

In our approach, the QoS service selection is executed after the transactional service selection. The set of candidate WSs is restricted by the transactional requirement. In order to obtain the optimized result as quickly as possible, we adopt the local optimization rather than the global optimization, in spite of that the latter can reach a more precise solution.

For each activity in the workflow, a local optimization is executed to get the best-performance WS to compose from the transactional WS set. For the selection of a WS for each activity, the system uses the classical Multiple Criteria Decision Making (MCDM) approach [12], which incorporates the scaling phase and the weighting phase as follows:

1) Scaling Phase. Some of the criteria could be negative, i.e., the higher the value, the lower the quality. This includes criteria such as response time and execution price. Other criteria are positive, i.e., the higher the value, the higher the quality. For negative criteria, values are scaled according to (1). For positive criteria, values are scaled according to (2).

The equations (2) and (3) can normalize QoS parameters to pure quantity between 0 and 1:

\[ V_{ij} = \begin{cases} 
\frac{Q_{ij}^{\max} - Q_{ij}^{\min}}{Q_{ij}^{\max} - Q_{ij}^{\min}} & \text{if } Q_{ij}^{\max} - Q_{ij}^{\min} \neq 0 \\
1 & \text{if } Q_{ij}^{\max} - Q_{ij}^{\min} = 0
\end{cases} \]  \tag{2}

\[ V_{ij} = \begin{cases} 
\frac{Q_{ij}^{\min} - Q_{ij}^{\max}}{Q_{ij}^{\max} - Q_{ij}^{\min}} & \text{if } Q_{ij}^{\max} - Q_{ij}^{\min} \neq 0 \\
1 & \text{if } Q_{ij}^{\max} - Q_{ij}^{\min} = 0
\end{cases} \]  \tag{3}

In the equations (2) and (3), \( Q_{ij}^{\max} \) is the maximum of the j-th quality criteria and \( Q_{ij}^{\min} \) is the minimum. \( V_{ij} \) is the normalized result of \( Q_{ij} \).

2) Weighting Phase. The following formula is used to compute the overall quality score for each Web service:

\[ \text{Score}(s_j) = \sum_{j=1}^{m} (V_{ij} \cdot W_j) \]  \tag{4}

where \( W_j \in [0,1] \) and \( \sum_{j=1}^{m} W_j = 1 \). \( W_j \) represents the weight of criterion j. End users express their preferences regarding QoS by providing values for the weights \( W_j \).

For a given activity, the system will choose the web service which has the maximal score. If there are
several services with maximal score, one of them is selected randomly.

5. Experimentation

To evaluate the behavior and performance of our transactional QoS-driven service composition approach, experiments were conducted by implementing the proposed algorithm on a PC Pentium 4 2.8GHz with 504MB RAM, Windows XP SP3, Java 2 Enterprise Edition V1.6.0.

Table 2 Set of values for QoS and Transactional Property for Individual WS

<table>
<thead>
<tr>
<th>Price</th>
<th>Response Time</th>
<th>Reputation</th>
<th>Success Rate</th>
<th>Availability</th>
<th>Transactional Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>([0, 100])</td>
<td>([0, 100])</td>
<td>([0, 20])</td>
<td>([0, 1])</td>
<td>([0, 1])</td>
<td>{a, c}</td>
</tr>
</tbody>
</table>

First of all, we generate a workflow with \(m\) activities and for each activity, we randomly generate from 5 to \(n\) candidate web services. Then we randomly generate QoS criteria values of services according to the following range: price and response time between 1 and 100, reputation between 1 and 20, success rate and availability between 0 and 1. The transactional property of candidate individual WS is randomly set to atomic or compensable. Set of values for QoS and transactional property are shown in Table 2. To simplify the experiments, the weights of different QoS criteria in the equations (4) are set to the same.

![Fig.2 Comparison of QoS-driven and Transactional QoS-driven approaches](image)

To compare our approach with the approach of QoS-driven selection (which is the same as the approach in this paper without transaction-driven), we set \(n\) and \(m\) both to 10. Executing both approaches for 10 times, we can see the scores of each composed WS in Fig 2. As can be seen from Fig.2, the QoS-driven approach always achieves higher scores than transactional QoS-driven approach, as the latter has an extra step to restrict the candidate web services in order to ensure a transactional composite WS.

![Fig.3 Computation Cost with m Arising](image)  ![Fig.4 Computation Cost with n Arising](image)
To test the computational complexity, first we keep \( m \) invariable as 10 and increase \( n \) from 10 to 100 with the step of 10, and then we keep \( n \) invariable as 10 and increase \( m \) from 10 to 100 with the step of 10. For each situation, we execute the approach for 10 times and calculate the average of computation cost. The computation costs are illustrated in Fig.3 and Fig.4, from which we can get that the rising of activities’ amount in a workflow will lead to more computation cost than the arising of the candidate WSs’ amount for each activity.

6. Conclusion

In this paper, we present the transactional QoS-driven approach, in which the QoS-driven service selection is local optimized and placed after the transactional service selection. The transaction-driven service selection determines the set of WSs to be considered for QoS selection depending on their transactional properties and the QoS-driven service selection maximizes the QoS of the composite web service by the MCDM. In the experiment, we can see the approach works well even if the scale of activities’ amount in the workflow and candidate web services.

Acknowledgment

This work is supported by the National Science & Technology Pillar Program Project No. 2011BAH25B01 and supported by the Fundamental Research Funds for the Central Universities No. CDJXS11181161.

References