ServNegotiator: A Negotiation Based System for Service Composition

Jing Zhao, Sherry X. Sun

***Abstract*—Service Oriented Architecture has offered an opportunity to quickly develop new business applications from the existing services developed independently. Given that many services provide the same functionality and differ in quality of service (QoS), e.g., cost and execution time, a critical challenge in service composition is to dynamically identify the appropriate services to meet the user's QoS requirements and preferences. In order to tackle this challenge, we have proposed an architecture for QoS-based service composition where negotiation is incorporated to help service consumers exchange offers and counter offers with providers and to enable dynamic agreements on QoS attributes. A proof-of-concept prototype ServNegotiator has been implemented to demonstrate the effectiveness of the proposed negotiation approach.**

***Keywords*: QoS, Service Oriented Architecture, Service Composition, Negotiation**

I. Introduction

The software industries have witnessed an increasing use of Service-Oriented Architecture (SOA) recently [1]. In an SOA environment, software components are packaged as independent services and can be accessed without the knowledge of the implementation platform. A service-oriented architecture does not depend on any specific technology and can be implemented using many interoperability standards, e.g., the Web Services Standard SOAP, WSDL, and UDDI [2].

As an emerging framework for distributed applications, Service-Oriented Architecture (SOA) allows integration of component services developed independently into complex business processes and applications, which are referred to as composite services. With the growing number of services offered by different service providers, many services offer the same functionality and differ in quality of service (QoS), such as price, response time, reliability, and reputation. Given that different service consumers may have different QoS requirements and preferences, it has become an important challenge to ensure the QoS requirements in forming new value-added applications through service composition.

The existing works in service composition mainly focus on the methods for selecting services with regards to the QoS requirements [3, 4]. Given a service composition request that includes a set of tasks and a list of functionally equivalent service candidates for each task, the service selection methods attempt to find one service candidate for each task to optimize the QoS of the composite service. Essentially, the selection is made based on the QoS properties of each service pre-defined by service providers. It is difficult for a service provider to offer the service with the QoS properties customized to different requests from consumers. In order to tackle this challenge, negotiation can be added to service composition to help service consumers and providers to exchange offers and counter offers and to enable dynamic agreements on some QoS criteria at runtime, thus providing a flexible way for service composition.

The purpose of this paper is to utilize negotiation to create composite services that meet the QoS requirements specified by a service consumer. The remainder of this paper is organized as follows. Section 2 discusses the basic concepts for service composition. In Section 3, we propose the architecture of a negotiation based system for service composition (ServNegotiator). In Section 4, we report a proof-of-concept prototype to demonstrate our negotiation based approach. Finally in Section 5, we conclude this paper and point out the future work.

II. Basic Concepts for Service Composition

In this section, we first define relevant concepts and then formulate the problem of service composition.

A. Composition Model

A composite service requested by a consumer includes a set of tasks $\{S_1, \ldots, S_n\}$. Each task corresponds to a service class which is a collection of functionally equivalent service candidates differing in QoS. The task needs to be accomplished by one candidate from the service class. Tasks can be executed sequentially, in parallel, conditionally, or in loops [5].

B. QoS Model

A service candidate s_i which belongs to the class S_i has m QoS attributes $[q_{i,1},\ldots q_{i,k}, \ldots, q_{i,m}]$, where $q_{i,k}$ is the k-th QoS attribute of service candidate s_i. The value for the k-th QoS attribute for a composite service, i.e., $q_{cs,k}$, can be determined by aggregating the corresponding attribute of each component service through aggregation functions such as summation, product, or maximum [6]. For each QoS attribute k (k=1, ..., m), the consumer has a requirement specified as a global constraint c_k for the aggregated value for the composite service. For negative attributes (the lower the better, such as price and response time), $q_{cs,k}<=c_k$ should be satisfied while $q_{cs,k}>=c_k$ should be satisfied for positive attributes (the higher the better, such as availability).

Jing Zhao is with Dept. Information Systems, City University of Hong Kong, Hong Kong, China, zhaojingwhut@126.com.

Sherry X. Sun is with Dept. Information Systems, City University of Hong Kong, Hong Kong, China, sherry.sun@cityu.edu.hk.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

C. Utility Function

In order to evaluate a given service, a utility function is used to map all the QoS attributes into a single value. Following the existing works [4], a Simple Additive Weighting (SAW) [7] technique is applied to define the utility function. There are two steps in applying SAW.

First, we need to normalize the values of the QoS attributes to the same scale in order to avoid inaccurate evaluation due to different measurement metrics used for different QoS attributes. In the normalization phase, positive and negative attributes are scaled in different ways as follows,

$$V_k(s_i) = \frac{q_{i,k}^{\max} - q_{i,k}}{q_{i,k}^{\max} - q_{i,k}^{\min}}, \text{ for negative attributes.} \quad (1)$$

$$V_k(s_i) = \frac{q_{i,k} - q_{i,k}^{\min}}{q_{i,k}^{\max} - q_{i,k}^{\min}}, \text{ for positive attributes.} \quad (2)$$

where $q_{i,k}^{\max}$ and $q_{i,k}^{\min}$ are the maximum and minimum values of the k-th attribute for service class S_i and $V_k(s_i)$ is the scaling value of the k-th attribute for the selected candidate s_i, and $s_i \in S_i$.

Second, a weight is assigned to each QoS attribute and the utility of service candidate s_i is the weighted summation given by

$$U(s) = \sum_{k=1}^{m} V_k(s_i) * w_k . \quad (3)$$

where $w_k \in [0,1]$, denoting the weight for attribute k and satisfying that

$$\sum_{k=1}^{m} w_k = 1 . \quad (4)$$

D. Problem Statement

For a composite service *CS* with *n* component services $\{S_1, \ldots, S_n\}$ and with *m* global QoS constraints $\{c_1, \ldots, c_m\}$, the goal of negotiation based service composition is to apply negotiation to obtain a composite service that meets the global constraints and achieves the optimal utility for the service consumer.

III. Architecture Design of Negotiation Based Service Composition System

In this section, we propose an architecture for the negotiation based service composition system, i.e., ServNegotiator. Figure 1 is the architecture of our system, which includes *Composite Service Specification Editor*, *Composite Service Specification Parser*, *Service Selection Manager*, *Negotiation Agent*, and *Service Negotiation Manager*.

Composite Service Specification Editor is a GUI editor from which the service consumer can define a composite service consisting of several component services. After the composite service is defined by the service consumer, it will be transformed into XML documents and sent to the *Composite Service Specification Parser*, an XML parser capable of analyzing the XML document describing the composite service. When the analysis is done, a number of component services and the executing relations of these services are derived. The parsing results of the composite service and the global QoS constraints specified by the service consumer are then sent to the *Service Selection Manager*. The *Service Selection Manager* first searches for service providers for each component service from the *Component Service Repository*, a repository where service providers register their services. Then it chooses a service provider for each component service for the system to negotiate with.

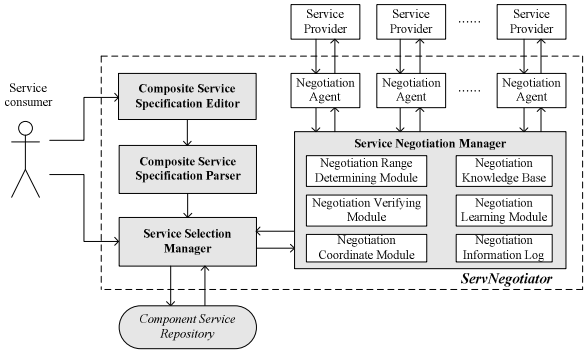

Figure 1. The architecture of the ServNegotiator

Service Negotiation Manager manages the negotiation process to satisfy the global constraints for the composite service. The process of negotiation based approach is depicted in Figure 2.

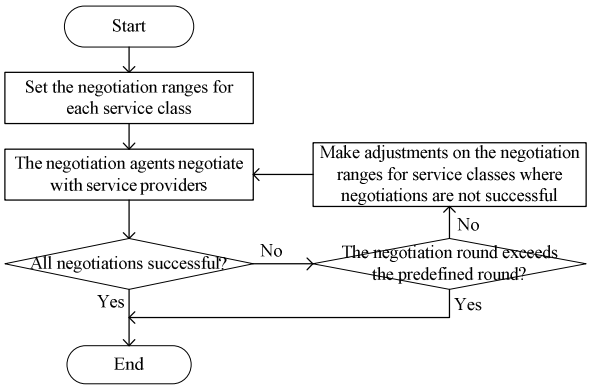

Figure 2. The process of negotiation based service composition

First, the negotiation ranges for each service class are set and this function is accomplished by the Negotiation Range Determining Module. All the QoS attributes of an offer sent to a service provider should fall into the range between the value most acceptable and the value least acceptable to the service consumer. The negotiation ranges are set for each service class and for each QoS attribute so that a negotiation agent can negotiate with the provider within those ranges. The most acceptable values are considered to be the best QoS values available on the service market. The least acceptable values for a negotiation agent are the reservation values. Since the negotiation between each negotiation agent and each provider is taken independently and concurrently, the negotiation ranges (particularly the reservation value for each QoS attribute) for each service class should be identified based on the principle that the offers falling into the ranges from each service class together can guarantee the global constraint for the composite service. Within the scope of this paper, we focus on the sequential composition structure. In such a sequential structure, the global constraints can be decomposed to help determine the negotiation ranges [8, 9].

Once *Negotiation Range Determining Module* determines

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the negotiation ranges, each negotiation agent negotiates with the corresponding service provider within this range. A negotiation process between a negotiation agent and a provider consists of an alternate succession of offers and counter offers. This continues until an offer is accepted or the negotiation is terminated by the negotiation agent or the provider (because the time deadline expires). The negotiation algorithm for the negotiation agent to negotiate with a provider can be identified as follows,

Step 0: Set the negotiation round counter r←0.

Step 1: Evaluate the service provider's offer using a utility function.

Step 2: Stop Criteria. If the stop criteria are true, the negotiation agent stops this negotiation and notifies the service provider; otherwise, go to step 3.

Step 3: Search for a new offer. If the utility of the generated offer is larger than that of the received one, then go to step 4; otherwise, go to step 2.

Step 4: The negotiation agent sends the new generated offer as a counter offer to the provider. r←r+1. Go to step 1.

The stop criteria in step 2 include the following situations: (1) The negotiation agent accepts the service provider's offer if the negotiation agent is unable to find any new offer that yields a better utility; or (2) The negotiation agent withdraws from the negotiation if service provider's offers are infeasible for a certain number of successive negotiation rounds predefined. In order to prepare a counter offer in step 3, a negotiation agent uses a concession or trade-off based approach [10, 11] to generate offers.

When all the negotiation processes between each negotiation agent and each provider are completed, *Negotiation Verifying Module* verifies whether the current negotiation outcome satisfies the global constraints for the composite service. If the negotiations for all service classes are successful, a feasible solution is generated. If the negotiations for some service classes are successful while for others are unsuccessful, *Negotiation Coordinate Module* is invoked to adjust the negotiation ranges for service classes where negotiation fails. For service classes where negotiations are successful, if for a QoS attribute, the agreed value is better than the reservation value, then the distance between the agreed value and the reservation value is considered as the saving which can be used to relax the reservation values in service classes where acceptable offers have not been identified. Once the negotiation ranges for those service classes are readjusted, the negotiation agents will continue negotiating with the corresponding service providers. The relaxation of reservation values makes it more likely to identify acceptable offers. The process of negotiation, verifying, and adjustment are repeated until a feasible solution is found or a maximum iteration round is reached.

The adjustment can help relax the reservation values for service classes where agreements are not achieved. Through the adjustment, the negotiation agents can collaborate to obtain a composite service.

Negotiation Knowledge Base stores the negotiation strategies, such as time dependent tactics and protocols for each negotiation agent to negotiate with a service provider. The negotiation records are stored in the *Negotiation Information Log* for the *Negotiation Learning Module* to learn from the negotiation experiences to enhance the outcome.

IV. Proof-of-Concept Implementation

In this section, we report a proof-of-concept prototype to demonstrate the effectiveness of negotiation in the service composition. The *Composite Service Specification Editor*, *Composite Service Specification Parser*, and the service discovery function in the *Service Selection Manager* in Fig. 1 are implemented following [12, 13]. Each module in the *Service Negotiation Manager* and the negotiation agents along with the service providers are implemented as autonomous agents using the JADE Agent Framework (JADE) [14]. Those autonomous agents are packaged as Web services using Java Web-services Development Package (JWSDP) [15], which supports key Web Services standards, such as SOAP, WSDL, and UDDI. The Agent Communication Language (ACL) is wrapped by SOAP in the prototype for agents' communication. The communications between those modules are implemented using SOAP with Attachments API for Java (SAAJ).

In such context, SerNegotiator assigns two negotiation agents, the transcoding negotiation agent and the merging negotiation agent, to negotiate with the transcoding service provider and the merging service provider, respectively. Figure 3 shows the offers generated in the negotiation process for the transcoding service. From this figure, we can see that both sides concedes gradually and the transcoding provider accepts the offer proposed by the transcoding negotiation agent and the agreed offer is (3.6, 125).

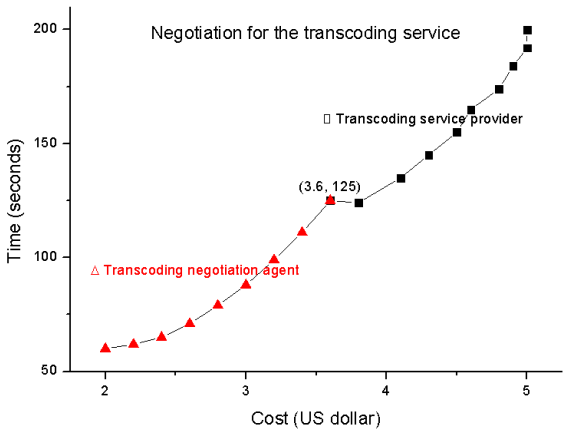

Figure 3. The negotiation for the transcoding service

Figure 4 shows the negotiation for the merging service. With the initial negotiating range (2.0-3.0, 80-170), the merging negotiation agent cannot reach an agreement with the merging provider because the offers proposed by the provider are outside the feasible negotiation ranges of the negotiation agent. Since the negotiation process is not yet successful for the merging service, the reservation values for the merging service need to be adjusted. The agreed QoS for the transcoding service is (3.6, 125) and there is still a distance between the agreed QoS and the reservation values (3.8, 140). The distance can be considered as a saving and used to relax the reservation values from (3.0, 170) to (3.2, 185) for the merging service. Given the adjusted reservation values, the

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

merging negotiation agent accepts the last offer (3.1, 181) sent by the provider.

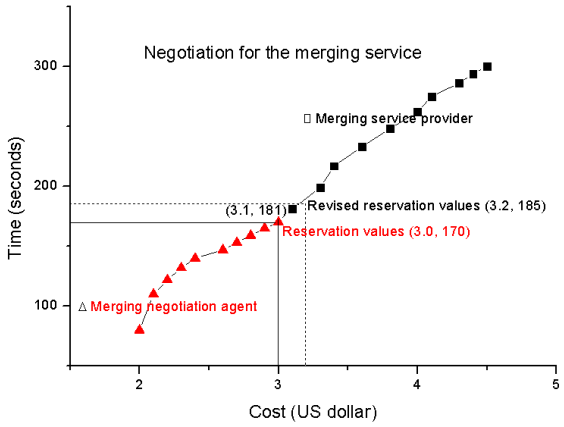

Figure 4. The negotiation for the merging service

From this experiment, we can see that the proposed negotiation approach can facilitate the negotiation agents to reach agreements on QoS with providers dynamically to reinforce the flexibility of the service composition.

V. Conclusions and Future Work

In this paper, we have proposed a negotiation based architecture for service composition which can help service consumers and service providers reach agreement on QoS dynamically through exchanging offers and counter offers. A proof-of-concept prototype has been implemented based on the proposed architecture to demonstrate our negotiation based approach. The negotiation mechanism in this paper not only enhances the flexibility of the dynamic service composition but also makes the constraints for the composite service easier to be satisfied through adding collaboration among different negotiation processes between *ServNegotiator* and different service providers offering different component service. We are in the process of extending our work in the following directions. First, we plan to utilize the trade-off strategy to enhance the performance of our negotiation approach. Second, we will extend our method to support more complex composition model including parallel structure, conditional structure, and loop structure other than the sequential structure.

References

[1] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, Service-oriented computing: state of the art and research challenges, IEEE Computer, vol. 40, pp. 38–45, 2007.

[2] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, Unraveling the Web Services web: An introduction to SOAP, WSDL, and UDDI, IEEE Internet Computing, vol. 6, pp. 89-93, 2002.

[3] D. Ardagna, B. Pernici, Adaptive service composition in flexible processes, IEEE Transactions on Software Engineering, vol. 33, pp. 369–384, 2007.

[4] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnamam, and H. Chang, QoS-aware middleware for web services composition, IEEE Transactions on Software Transactions, vol. 30, pp. 311–327, 2004.

[5] S.X. Sun, J.L. Zhao, J.F. Nunamaker, O.R. Sheng, Formulating the Data Flow Perspective for Business Process Management, Information Systems Research, vol. 17, pp. 374-391, 2006.

[6] J. Cardoso, J. Miller, A. Sheth, J. Arnold, Quality of service for workflows and web service processes, Journal of Web Semantics, vol. 1, pp. 281–308, 2004.

[7] C.L. Hwang, and K. Yoon, Multiple Attributes Decision Making, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 1981.

[8] M. Alrifai, T. Risse, P. Dolog, W. Nejdl, A scalable approach for QoS-based web service selection, Lecture Notes in Computer Science, vol. 5472, pp. 190–199, 2009.

[9] S.X. Sun, J. Zhao, H. Wang, A Negotiation Based Approach for Service Composition, In Proceedings of 5th Design Science Research in Information Systems and Technology (DERIST), Switzerland, 2010, In Lecture Notes in Computer Science, vol. 6105, pp. 381-393, 2010.

[10] P. Faratin, C. Sierra, and N.R. Jennings, Negotiation decision functions for autonomous agents, Journal of Robotics and Autonomous Systems, vol. 24, pp. 159–182, 1998.

[11] P. Faratin, C. Sierra, and N.R. Jennings, Using similarity criteria to make issue trade-offs in automated negotiations, Artificial Intelligence, vol. 142, pp. 205-237, 2002.

[12] L. Zeng, B. Benatallah, A.H.H. Ngu, and P. Nguyen, "AgFlow: Agent-Based Cross-Enterprise Workflow Management System (Demonstration Paper)," Proc. 27th Int'l Conf. Very Large Data Bases, 2001.

[13] R. Aggarwal, K. Verma, J. Miller, W. Milnor, Constraint Driven Web Service Composition in METEOR-S, in Proceedings of the 2004 IEEE International Conference on Services Computing (SCC'04), 23 – 30.

[14] JADE, Java Agent DEvelopment Framework, http://jade.tilab.com/, 2010.

[15] JWSDP, Java Web-services Development Package, http://java.sun.com/developer/technicalArticles/WebServices/WSPack/, 2010.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)