PROVIDING A SEMI-CENTRAL MODEL FOR WEB SERVICE COMPOSITION

Saman Rouhi [1] and Mehregan Mahdavi[2]

[1]Department of Computer Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
saman.rouhi@gmail.com
[1]Department of Computer Engineering, University of Guilan
mehregan.mahdavi@gmail.com

ABSTRACT

In recent years, plenty of research has been done related to the methods of composition of web services. Notwithstanding much effort, it seems that this technology has a long way till it attains perfection. In this research, providing a suggestive model, we performed service orchestration semi-centrally. To do this, we need to group web services. We created groups or clusters of web services and put services in appropriate groups, by this way, the number of control flow and data flow notifications decreases considerably. For each group a controller and a Tuple space is considered and the controller is responsible for invoking services of that group Simulation results show that service orchestration can be influential in decreasing the amount of traffic, especially in composite services with a high level of communication in their component services. In addition, a decrease in the level of distribution in service orchestration can improve capability of execution monitoring and failure compensation.

KEYWORDS

Service Oriented architecture, Web Services Composition, Web Services Orchestration Models, Service Grouping

1. INTRODUCTION

Service-oriented architecture and applying web services, are two appropriate ways to create a relation between different programs and meeting growing and varying needs of users in organizations and business environments. Yet in many cases, using atomic services alone, doesn't grant customer needs. Composition of web services for organizations provides them with the possibility of complying with more complicated needs of users and adapting their possible changes. A principle step in service composition is orchestration. Service orchestration can be accomplished in two ways: distributed and central. Central systems e,g eflow and Sword, have some strengths like; ease of performing and high capability of execution monitoring. In contrast, some weak points like; low scalability, bottleneck problem , redundancy in data transfer and single point of failure exist. In systems where distributed orchestration is used, scalability is in a high level and the problem of bottleneck and single point of failure don't exist. But these systems have some faults as well. One of problems distributed systems face with, is the high level of traffic and the number of messages exchanged between hosts which can cause a huge cost. In addition, distribution makes the capability of monitoring composite service's execution decrease and the operation of failure compensation face difficulty.

In this article we provide a model, named "semi-central orchestrating model", which in addition to preserving maximum capabilities of distributed method, it decreases the exchange of messages and provides a possibility of better performance when facing a failure in this system as well. The approach of this new model is acquired from CCAP model which orchestrates web

DOI : 10.5121/ijitcs.2012.2201

services in a distributed manner and based on Tuples. In the following, the article is organized in this way: section 2, reviews related works and specially CCAP model. In section 3, a suggestive model is introduced for orchestration of services. Section 4 shows the experiments' results and finally, section 5 concludes.

2. RELATED WORKS

Variable approaches have been introduced for web service composition which are grouped based on different criteria. Some approaches perform the task of service composition in a static manner, like Microsoft Biztalk and Bea web logic. Others do this in a dynamic manner like [2]. In [4,5,6,7] the model-based approaches have been put for dynamic composition of web services. In these approaches, UML (Unified Modeling Language) is used to provide a high level of abstraction and to enable direct mapping to other standards such as BPEL4WS.

In [8,9], some approaches for declarative composition of web services are introduced. The declarative approach consists of two phases: the first phase takes an initial situation and the desired goal as starting point and constructs generic plans to reach the goal. The latter one chooses one generic plan, discovers appropriate services and builds a workflow out of them.
Finding and composing web services can be context-based, some examples of this approach have been used in [10,11]. E-Services in this context are defined by a request perspective and a provisioning perspective, both dealing with QoS issues in order to allow service negotiation based on user requirements, channels and provider constraints. UML can be used to describe the entities involved in both perspectives and show its associations.

The context framework has been proposed here provides several context types, such as location (GPS coordinates, country, local time and time zone), information about the consumer invoking a service (name, email address, preferences) and client context information such as hardware or software.

In a different viewpoint, methods of web service composition are put in 3 groups including manual, automatic and semi-automatic. By manual composition, we mean that the composite service is designed by a human designer (i.e., service provider) and the whole service composition takes place during the design time. This approach works fine as long as the service environment, business partners, and component services do not or rarely change.

A user needs to execute all these services one by one and these tasks can be time and effort consuming. Due to constant changes in output/input parameters values, interfaces, networking issues, it is difficult to integrate and maintain these services [12].

Automatic service composition approaches typically exploit the Semantic Web and artificial intelligence (AI) planning techniques. By giving a set of component services and a specified requirement (e.g., user's request), a composite service specification can be generated automatically [13,14].

Actually, semantic web aims at decreasing human intervention in the communication of software systems as much as possible.

2.1. CCAP Approach

The CCAP system (Configurable Composition and Adaptive Provisioning of composite services) has been developed [14,15] that provides a system infrastructure for distributed, adaptive, context-aware provisioning of composite Web services.

CCAP allows service designers to focus more on specifying service composition requirements at a high level of abstraction such as business logic of applications, generic exception handling policies, and contextual constraints.

Because the suggestive approach is acquired from CCAP, in this section of the article, basic concepts and performance of CCAP approach are reviewed.

A controller is considered for each state and a service corresponding to that controller is responsible for performing the task of that state.

User agent acts as the controller of a personalized composite service. The messages exchanged between the controllers for notifying that a given state should/may be entered are called control-flow notifications. The execution of the composite service is orchestrated through peer-to-peer interactions between the task controllers. the messages exchanged between the controllers and the component services are called service invocations/completions.

The scenario is that when service is invocated, the user agent sends a control flow notification to the controller corresponding to the state considered as the first place. After the related task is performed, this controller recalls other controllers by control flow notification and orchestration continues to the end. In final stage, a control flow notification is sent to user agent showing the completion of composite service operation. The messages exchanged between controllers containing the values of the output parameters of a task t1 that need to transmit to another task t2 are called data flow notifications. In this approach the orchestration of web services is performed in a peer to peer environment and in a distributed manner. You can see an example of distributed orchestration by Tuples and controllers in figure 1.

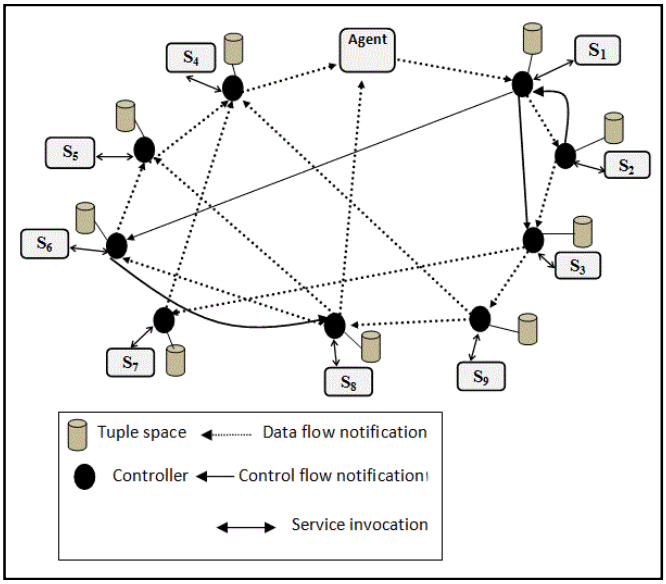

Figure 1. Distributed Orchestration

3. SEMI CENTRAL MODEL

According to the order of distributed system, each service corresponds to a controller and a Tuple space. In figure one, services are named from $\mathbf{S_1}$ to $\mathbf{S_9}$. As we can see, web services have a high amount of communication in a way that control notifications are sent reciprocally in $\mathbf{S_1}$ and $\mathbf{S_2}$ or $\mathbf{S_6}$ and $\mathbf{S_8}$ services. Suppose that each pair sends the message to one another twice. Then the total number of control flow notification sent among controllers will be 20.

Of course in some of these cases, control flow and data flow notifications can be sent in a single message. Here, each service and its corresponding controller are put on one machine. So the traffic between controller and its corresponding service will be local. Yet the traffic among controllers will be physical and on the web.

If we can create groups or clusters of web services and put services in appropriate groups, we can decrease the number of control flow and data flow notifications considerably. For each group a controller and a Tuple space is considered and the controller is responsible for invoking services of that group. Tuple space contains control Tuples corresponding to all services existing in the group. In this manner there is no need to consider a separate Tuple space and controller for each service, technical costs decrease as well. In figure 2 we can see that services are classified into 3 groups and the number of data flow and control flow notifications is reduced from 20 to 7.

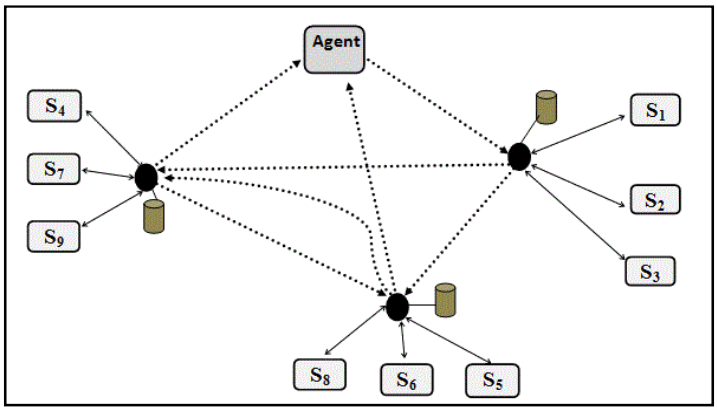

Figure 2. Semi-Central Web Service Orchestration

The reason is that web services existing in one group don't need to send a notification to communicate each other anymore. Because all use a common controller. Control Tuples related to each 3 services are in one Tuple space. Instead, this kind of service clustering causes the controller and its corresponding web service not to exist on one machine, so the operation of web service invocation contains a higher cost compared to distributed system.

Certainly, controller tasks in new model will be different compared to distributed one. Here, in addition to the management of communication among controllers, the controller should manage the services existing in one group, so the task will be more difficult. Steps of passing through distributed to semi-central model are shown in table 1.

Table 1. Steps of passing to semi-central model

Step 1	**Measuring the amount of communication among controllers**
Step 2	**Grouping web services using the suggestive greedy algorithm**
Step 3	**Creating appropriate controllers for managing the orchestration**
Step 4	**Creating necessary changes in control Tuples and Tuple space**

3.1. Composition steps

Service composition steps performed by controller using semi-central approach are as follows:

Step 1: the service personalized by a user is sent to corresponding user agent.

Step 2: the user agent produces control Tuples based on information received and sends them to Tuple spaces corresponding to the groups considered. these groups, actually, include those services which should be executed.

Step 3: controllers make relationships with Tuple space.

Step 4: controller carries out considered service invocation based on control Tuples.

After controller reads the Tuple space, two states may happen.

First, another service of this group should be performed, in this situation there is no need for control flow notification. Second, if a service invocation from another group is needed or we have to send a notification to a service outside this group, then the controller should send a control notification to another controller of a considered group in order for the service to be invocated from the next group.

Steps 5 and 6: considering control Tuples, the controller sends service results to user agent and in a suitable time, user agent sends the results to the user.

3.2 Messages exchange

In composite services, the more messages exchanged, the more efficient will be the semi-central system compared to distributed method.

The highest amount of exchanging physical messages required for executing a composite service in semi-central method including n component service invocation, can be bounded to g(1+2n) relation, where g shows the number of groups. The reason of this relation is that; first, control flow notifications are exchanged just among group indicators. Second, data flow notifications and web service invocations require physical messages as well. When a principle state is executed, at most, g notifications are sent by the controller of the group where this state exists one for user agent and g-1 for other controllers. As a result, after invocating a service and executing related task, g+2 physical messages have been sent. Now, if executing a composite service includes n service invocation, n× (g+2) messages are exchanged along it. In addition, when executing a composite service is started, g messages are sent from user agent to controllers. Therefore, according to the relation 3-1 the total number is:

$$n(g+2)+g = ng+2n+g = (n+1)g+2n \qquad (3\text{-}1)$$

Now, we compare this relation to m (n+1) which is the number of physical messages in distributed state. If m>g+1, then we can conclude that the number of messages in semi-central method has decreased.

Actually, the more the number of groups increases, the more the number of messages will be. Yet we will have a higher reliability and a better scalability. On the contrary, a decrease in the number of groups, results in a decrease in the number of physical messages and reliability and scalability will decline as well. So, we should try to build a balance in the number of groupings. Actually, the worst state, in term of the number of messages sent, shows the highest superiority of semi-central method to distributed one.

4. SIMULATION AND EVALUATION

We simulated this system in order to measure it. MPI.Net was used for simulation which makes process parallel performance possible, in a way that there is a capability of defining separate codes for each of coordinators [17], a net library which helps to provide parallel programs with high functionality. Class Assistant composite service [14] was selected as an example for simulation. Class Assistant is a program which helps students with managing their class activity.

Component services existing in this composite service were put in two groups and a semi-central orchestrating system like figure 3 was created. We have considered three different states that have been named with service 1, service 2 and service 3.

We need appropriate criteria for measurement. We try to decrease the costs in system by creating a semi-central model. So the amount of messages exchanges and the traffic transferred in system can be two appropriate criteria for measurement of suggested model.

4.1. Measurement based on the amount of notification exchange

We simulated semi-central and distributed models for class assistance composite service and analyzed them measuring the amount of messages exchanged. Table 2 shows the number of exchanged messages in different states of class assistance composite service.

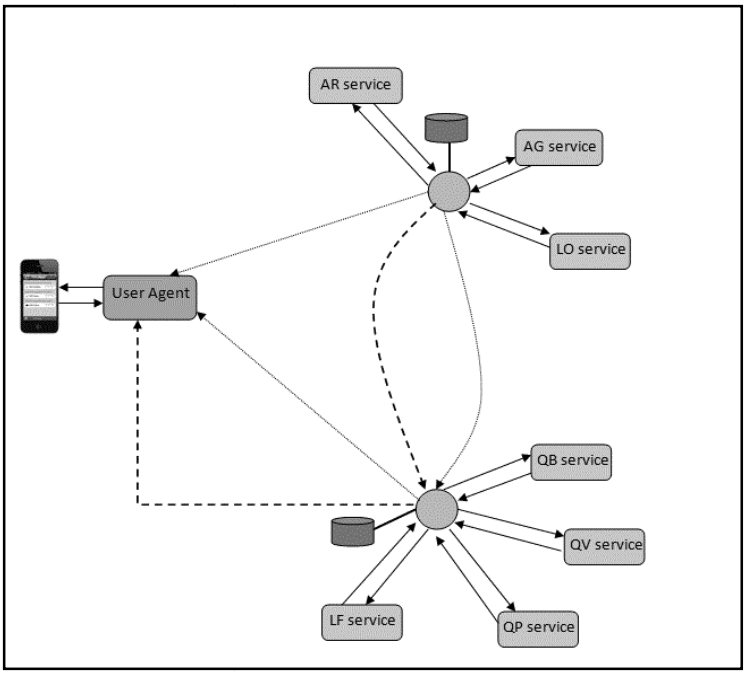

Figure 3. notification exchange between controllers and component services in semi-central model

The messages in each of two models were classified into two groups of messages between two controllers and messages between service and controller.

Table 2. number of messages between distributed and semi-central models

SERVICE NUMBER	DISTRIBUTED MODEL		SEMI CENTRAL MODEL	
	CONTROLLER - CONTROLLER	CONTROLLER - SERVICE	CONTROLLER - CONTROLLER	CONTROLLER - SERVICE
1	24	12	15	12
2	32	16	19	16
3	37	18	21	18

As we can see in the table, the number of messages transferred between two controllers, decreased in semi-central model compared to distributed one. Because the number of component services' execution is the same in both models for a single scenario, the number of messages exchanged between services and controllers is the same as well. Actually these are invocation and service-completion notifications. Of course, it should be noted that the amount of traffic this kind of messages impose to system, is different in two models. Because in distributed model, these messages are exchanged between the controller and the service placed on a machine. But in semi-central model, machines are different. As a result, although the number of messages is the same, the amount of traffic caused by invocation/completion messages in semi-central model is higher than in distributed model.

4.2. Traffic assessment

To asses created model, we can measure and analyze transmitted traffic when composite service is being executed. To measure traffic, we need to use some software designed for this task. Fiddler and Wireshark are among mostly-used software. In figure 4, a comparison between distributed and semi-central models' traffic on some composite services ,defined in the section of measuring number of messages, is shown. The values of traffic are in kilobyte. In composite service number 1, the amount of traffic in semi-central orchestration is a little better than distributed one's. The reason is the reduction of control flow and data flow notifications in semi central model. The traffic between controller and web service increased when semi-central orchestration took place, this is why the amount of traffic is not so different in two models. In composite services 2 and 3, communication between component services increase gradually and the number of control flow and data flow notifications will increase as well. As a result, optimization grows and the difference between distributed and semi-central orchestration in composite services 2 and 3 is high now. As you can see, while communication among component services grew higher, the traffic in distributed model, compared to semi-central one, increased as well. So it is better we use semi-central model in composite services with high communication among their component services.

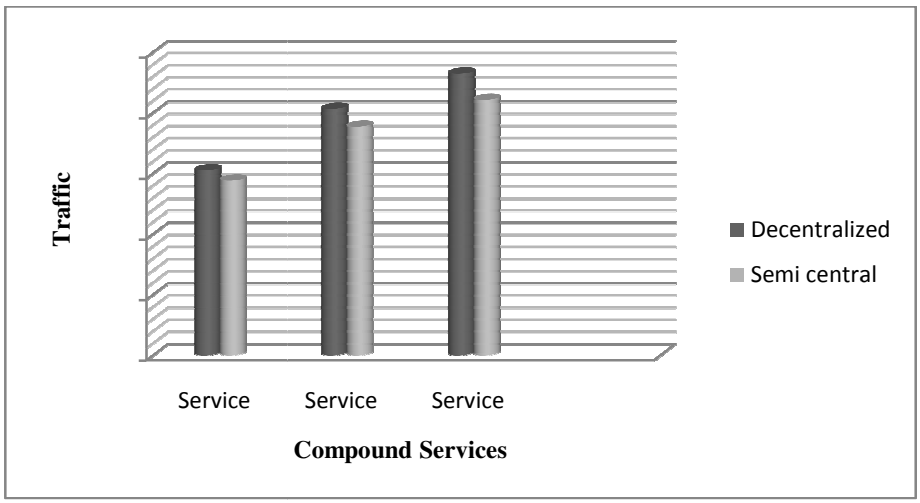

Figure 4. comparison between distributed and semi-central models

5. CONCLUSION

In this article we provide a semi-central model for web services orchestration. Web service orchestration for executing a composite service based on Tuples will be possible through this model. We can group existing component services optimally. Then controllers' tasks were

defined in order to create coordination among them. In this manner, the number of controllers and Tuple spaces in service orchestration systems decreased.

Experiment results show that message exchange between two controllers in semi-central model is low compared to distributed one. While invocation/completion message exchange are equal in both models. The amount of system traffic imposed by invocation/completion messages of semi-central model is totally more than that of distributed one. The more communication among services in a composite service increases, the more different will be the level of notification exchange in semi-central and distributed models. It means that semi-central model will stay in a better situation. The amount of traffic transmitted in semi-central model is low compared to distributed one. This is because notifications among controllers decrease. Therefore, this semi-central model in composite services, where communication among component services is high, will have a more efficient performance.

The manner of grouping web services in distributed model and changing it into a central one is of much importance. Because inappropriate grouping can have a reversed effect and increase the costs. Our future work is about this subject.

Due to the decrease in the number of controllers, execution monitoring and service compensation in semi-central model, compared to distributed one, is simpler.

One of weak points of our suggested model compared to distributed one, is its low scalability. Yet we improved this property of our model relative to the central one.

REFERENCES

[1] Ponnekanti, S & Fox, S, (2002), "SWORD: A Developer Toolkit for Web Service Composition", *Proceedings of the 11th International WWW Conference*.

[2] Casati, F, Ilnicki, S. Jin & Shan, M, (2000), "An open, flexible, and configurable system for service composition", IEE- Advanced Issues of E-Commerce and Web-Based Information Systems, pp. 125-132.

[3] Chen, X., Zeng, H., & Wu, T., (2010), "*Decentralized Orchestration with Local Centralized Orchestration for Composite Web Services*", IEEE- Parallel and Distributed Computing, Applications and Technologies, pp. 255-260.

[4] ZHAO, C., DUAN, Z., & ZHANG, M., (2009), "A Model-Driven Approach for Dynamic Web Service Composition", Software Engineering, WCSE '09. WRI World Congress , ISBN: 978-0-7695-3570-8, pp. 273-277.

[5] SCOGAN, D., GRONMU, R., & SOLHEIM, I., (2004), "Web Service Composition in UML" , Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf, ISBN: 0-7695-2214-9, pp. 47-57.

[6] Orriens, B., Jian, Y., & Mike Papazoglou, P., (2003), "Model Driven Service Composition", Springer-Verlag Berlin Heidelberg, pp.75-90.

[7] Orriens, B., Jian, Y., & Mike Papazoglou, P., (2003), "A Framework for Business Rule DrivenvWeb Service Composition", Springer-Verlag Berlin Heidelberg, ER 2003 Workshops, LNCS 2814, PP. 52-64.

[8] BENATALLAH, B., SHENG, Q., (2003), "The Self-ServEnvironment for WebServices Composition", Internet Computing, IEEE , ISSN: 1089-7801 , pp. 40-48.

[9] Ambroszkiewicz, S., (2003), "enTish: An Approach to Service Composition", SPRINGER - Lecture Notes in Computer Science, PP. 168-178.

[10] Keidl, M., Kemper, A., (2004), "A Framework for Context-Aware Adaptable Web- Services", 2004. Springer-Verlag Berlin Heidelberg, pp. 826-829.

[11] Doulkeridis, Christos, Efstratios Valavanis & Michalis Vazirgiannis, (20030, "Towards A Context-Aware Service Directory", Springer-Verlag Berlin Heidelberg, pp. 54-65.

[12] FAISAL, M., 2088, (2008), " Dynamic Web Service Composition", Proceedings of Computing and Engineering Annual Researchers' Conference, ISBN 978-1-86218-067-3, pp. 48-54.

[13] NEWCOMER, E., (2002), "Understanding Web Services: XML, WSDL, SOAP, and UDDI", Addison-Wesley Professional, ISBN-10: 0-201-75081-3.

[14] D. Berardi, G.D. Giacomo, & D. Calvanese, (2005), "Automatic Composition of Process-Based Web Services: A Challenge", Proc. 14th Int'l World Wide Web Conf.

[15] Sheng, Q.Z., (2006), "Composite web services provisioning in dynamic environments", University of New South Wales. Computer Science and Engineering.

[16] Benatallah, B., Sheng, Q.Z., (2009), "Configurable Composition and Adaptive Provisioning of Web Services", IEEE Transactions on Services Computing, pp. 34-49.

[17] Gregor, D., & Martin, B., (2008), "MPI.NET Tutorial in C#", The Trustees of Indiana University.