Selection of Web Services for Composition Using Location of Provider Hosts Criterion

Zakaria Maamar[1], Quan Z. Sheng[2], and Boualem Benatallah[2]

[1] College of Information Systems
Zayed University, Dubai, U.A.E
zakaria.maamar@zu.ac.ae
[2] School of Computer Science & Engineering
The University of New South Wales, Sydney, Australia
{qsheng,boualem}@cse.unsw.edu.au

Abstract. We present a Web service composition approach that relies on three selection criteria: execution cost, execution time, and location of provider hosts. A Web service is an accessible application that can be automatically discovered and invoked by other applications and humans. Web services can be composed into high level business-processes that users trigger in order to satisfy their needs. Because providers can have Web services in common, criteria are needed to select which Web services will be considered for composition. Location of provider hosts is among these criteria and aims for example at reducing the number of remote interactions between provider hosts.

1 Introduction

Nowadays, several businesses are adopting for their operation *Web-based solutions*, aiming for more process automation and more worldwide visibility. Thanks to the Web technology, users from over the world can satisfy their needs by browsing and triggering the services of these businesses. Such services are usually known as *Web services* [1]. The advantages of Web services have already been demonstrated in various projects [1] and highlight their capacity to be composed into high-level business processes. For example, a summer vacation business process calls for the *collaboration* of at least four Web services: flight reservation, accommodation booking, attraction searching, and user notification. These Web services have to be connected according to a certain flow of control (first flight reservation and then accommodation booking and attraction searching). Multiple technologies are associated with the success of Web services including WSDL, UDDI, and SOAP [4]. These technologies aim at supporting the definition of Web services, their advertisement, and their binding for triggering purposes.

With the progress of the telecommunication technologies, new Web services are devised for the benefit of persons who are most of the time on the move (e.g., sales representatives). These persons heavily rely on mobile devices (e.g. PDAs) to conduct their daily operations. *M-services* (M for Mobile) denote these new Web services [6] and are meant to be either 1) executed remotely from mobile devices or 2) transferred through a wireless channel from their host to mobile devices on which their execution is performed.

In general, composing multiple services[1] rather than accessing a single service is essential and provides benefits to users. Discovering the component services, inserting the services into a *composite service*, triggering the composite service and its component services for execution, and monitoring the execution of the composite service are among the operations that users will have to be in charge. Most of these operations are complex, although repetitive with a large segment suitable to computer aid and automation. Therefore, *Software Agents* (SAs) are deemed appropriate to assist users in their operations [7]. SAs are autonomous entities that act on behalf of users, make decisions, interact with other agents, and migrate to provider hosts if needed.

The identification of the component services to constitute a composite service is basically a process that relies on the use of selection criteria. Execution cost, execution time, reliability, and reputation, just to cite a few are among the selection criteria that have commonly been used in various projects. In this paper, we consider *location of provider hosts* as another criterion that is worthwhile to integrate in the selection process of services. Provider hosts are associated with computing resources on top of which services are executed. By gathering the maximum number of services for execution in the same provider host[2], the following advantages can be obtained: 1) remote interactions between provider hosts can be reduced, 2) migrations of agents to provider hosts can be avoided, and 3) remote exchange of data between provider hosts can be reduced, too. In this paper, we aim at presenting a service composition approach that uses the location of provider host as a major selection criterion.

Section 2 overviews Web services. The agentification of an environment of Web services is presented in Section 3. Section 4 discusses the preparation of services for composition. Section 5 presents our ongoing work. Finally, Section 6 overviews related work and draws our conclusions. It is made clear at that level that the mechanisms for discovering the component services of a composite service, while important, do not fall within the scope of this paper.

2 Web services

A Web service is an accessible application that can be automatically discovered and invoked by other applications (and humans as well). An application is a Web service if it is [2]: 1) independent as much as possible from specific platforms and computing paradigms; 2) developed mainly for inter-organizational situations rather than for intra-organizational situations; and 3) easily composable (i.e., its composition with other Web services does not require the development of complex adapters).

Maamar et al. introduce M-services as a specific type of Web services [5]. Two definitions are associated with an M-service. The *weak* definition is to remotely trigger a Web service from a mobile device for execution. In that case, the Web

[1] In the rest of this paper, Web service and service are interchangeably used.

[2] It is assumed that this host has the capabilities to meet the execution requirements of the services.

service is an M-service. The *strong* definition is to transfer a Web service through a wireless channel from its hosting site to a mobile device where its execution takes place. In that case, the Web service is an M-service that is: 1) transportable through wireless networks; 2) composable with other M-services; 3) adaptable according to the computing features of mobile devices; and 4) runnable on mobile devices. In this paper, we only consider the M-services that comply with the *weak* definition. The M-services that comply with the *strong* definition have been considered elsewhere [5].
In the rest of this paper,

- The term composite service (*C-service*) denotes the list of component services, whether composite services or primitive services (*P-services*), that are involved in a composition.
- A C-service is deployed in two different versions: composite Web service for users of fixed devices and composite M-service for users of mobile devices.
- A P-service always refers to a Web service.
- A P-service can be invoked for execution either remotely or locally.

3 Agentification of an environment of Web services

For the agentification needs of an environment of services, we decided to deploy a *multi-domain* architecture (Fig. 1). Compared to hosting sites, domains are spread across the network and administrators manage them. Two types of domain exist: *user-domain* and *provider-domain*. We assume the existence of one user-domain (independently of the issues of bottleneck or single point of failure) and several provider-domains. A domain is a computing platform on top of which portal of services are deployed and also agents undertake their operations. Basically, two types of software agents have been considered during the agentification process. User-agents act on behalf of users. And, provider-agents act on behalf of providers announcing their services to users. Users browse the portal of C-services from different devices: fixed devices such as desktops and mobile devices such as PDAs. The deployment of a C-service is not affected by the type of the device from which it is launched. The only difference occurs at the communication protocol that connects users to the portal of C-services. For users of fixed devices, HTTP is the protocol. For users of mobile devices, WAP is the protocol.

In Fig. 1, the user-domain has two zones: *service-zone* and *working-zone*. The service-zone has a dedicated portal from which C-services are managed in terms of specification, development, and deployment. The service-zone of the user-domain has also *a pool* from which user-agents are created. For their installation, user-agents are located in the working-zone of the user-domain. User-agents are mobile and thus, have the ability to migrate from one domain to another based on the strategy that will be adopted for invoking services (Section 4). For each C-service that a user selects, a user-agent is associated with that C-service for performance purposes.

A provider-domain consists of a working-zone and several portals of P-services. Each portal is related to a category of P-services such as education and travel. The working-zones receive user-agents arriving from the user-domain or from other provider-domains. Within these working-zones, installation and control procedures of user-agents are executed. Portals of provider-domains are associated with provider-agents that handle the invocation requests that user-agents submit to the P-services. A user-agent submits a local request to a provider-agent in case both agents reside in the same provider-domain. This means that the user-agent has migrated to that provider-domain; the user-agent arrives either from the user-domain or from a different provider-domain. In case the user-agent and provider-agent are in separate domains, the user-agent submits a remote request to the provider-agent so, a P-service can be executed.

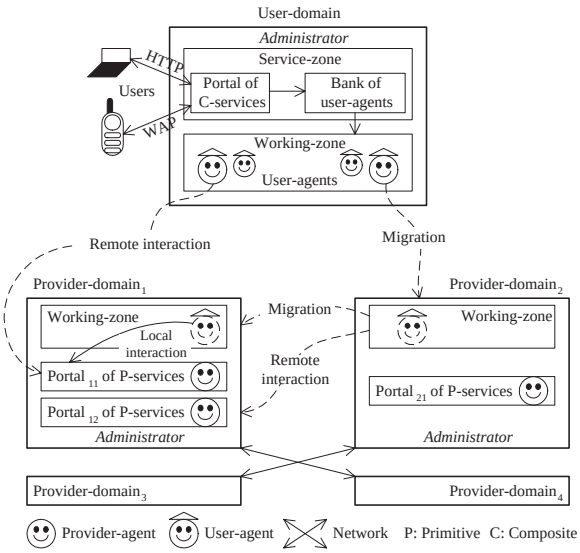

Fig. 1. Agent-based multi-domain architecture

4 Composition of Web services

For any C-service, the preparation of its component P-services for composition is based on three selection criteria: *execution time*, *execution cost*, and *location of provider hosts* (provider hosts correspond to provider-agents associated with their provider-domain). The first two selection criteria are intimately related to a P-service. The third selection criterion aims at gathering in the same provider-domain the maximum number of P-services for execution, reducing 1) the number of remote interactions between domains, 2) the number of migrations of user-agents to domains, and 3) the number of data transfer between domains. The ranking of domains is conducted as follows. First, the domain of where a

service is being executed is considered. By selecting this domain, remote data exchanges between services are avoided. Next, the domain of where the user-agent resides now is considered. By selecting this domain, local invocations of services as well as local data exchanges between services are enabled. Finally, in case none of the aforementioned cases happens, any other domain is considered.

4.1 Definitions

A C-service is an ordered set of component P-services$_{i\,=\,1,\,\cdots,\,n}$. Each P-service$_i$ is offered by several provider-agents$_{j\,=\,1,\,\cdots,\,m}$. In the preparation process, the user-agent aims at reaching two goals: associate a P-service $p.s_i$ with a provider-agent $pro.agt_j$, and define the strategy of invoking the P-service $p.s_i$ (remotely or locally).

Let us assume a C-service CS of p P-services, $CS = \{p.s_1,\ p.s_2,\ \cdots,\ p.s_p\}$. A deployment of the C-service CS aims at defining the set $\{< p.s_1, pro.agt_1, type >$, $< p.s_2, pro.agt_2, type >,\ \cdots,\ < p.s_p, pro.agt_p, type >\}$ where $\bigcup_{i=1}^{p} p.s_i = CS$ and for each $< p.s_i, pro.agt_i, type >_{(i\in[1,p])}$ the P-service $p.s_i$ is provided by the provider-agent $pro.agt_i$ and invoked according to remote or local *type*. It should be noted that the number of provider-agents that contribute to the deployment of a C-service CS is not necessarily equal to the number of P-services that are involved. Certain provider-agents may contribute to a C-service with more than one P-service (e.g., $< p.s_1, pro.agt_1, type >$ and $< p.s_2, pro.agt_1, type >$).

Given a P-service $p.s_i$, its execution cost is decomposed into two parts:

- $Remote.Cost(p.s_i)$ includes 1) the cost of establishing a communication link between the domain of the user-agent and the domain of the provider-agent of the P-service $p.s_i$, plus 2) the cost of performing the P-service $p.s_i$, plus 3) the cost of returning the results from the domain of the provider-agent to the domain of the user-agent.
- $Local.Cost(p.s_i)$ includes 1) the cost of transferring the user-agent from the domain in which it is currently located to the domain of the provider-agent of the P-service $p.s_i$, plus 2) the cost of performing the P-service $p.s_i$.

Therefore, the execution cost $Cost(CS)$ of a C-service CS is the sum of the costs of all the P-services, $Cost(CS) = \sum_{i=1}^{p}(Remote.Cost(p.s_i) \oplus Local.Cost(p.s_i))$. The execution cost of a P-service $p.s_i$ is either a remote cost or a local cost.

Given a P-service $p.s_i$, the execution time $Time(p.s_i)$ does not depend on the invocation strategy (i.e., local or remote). Therefore, the execution time $Time(CS)$ of a C-service CS is the sum of

- The execution times of all the P-services ($\sum_{i=1}^{p} Time(p.s_i)$);
- Plus, the estimated communication time of triggering the first P-service ($Time(CS, p.s_1)$) after the C-service CS has been initiated;
- Plus, the estimated communication time of triggering the next P-service $p.s_{i+1}$ after the execution of the current P-service $p.s_i$ is completed

$(\sum_{i=1}^{p-1}(Remote.Time(p.s_i, p.s_{i+1}) \oplus Local.Time(p.s_i, p.s_{i+1})))$. The communication time handles the case of whether the next P-service $p.s_{i+1}$ will be locally or remotely triggered. It is expected that the time of locally triggering a next P-service to be neglected because of the small duration $(Local.Time(p.s_i, p.s_{i+1}) \approx 0)$.

4.2 Preparation process

The preparation process to deploy a C-service CS is decomposed into two phases. Phase 1 consists of searching for the provider-agents that have the P-services. Because provider-agents can have P-services in common, Phase 2 consists of selecting a specific provider-agent based on the criteria of execution time, execution cost, and location of provider hosts.

Phase 1: Search for provider-agents As stated in Section 1, through appropriate discovering mechanisms the provider-agents that offer the P-service $p.s_i$ are known to the user-agent that handles the execution of a C-service. For each P-service $p.s_i$, a set of potential provider-agents exists. This is similar to $< p.s_i, PRO.AGT_i, type >$ where $PRO.AGT_i = \{pro.agt_1, \cdots, pro.agt_n\}$ is the list of provider-agents that have the P-service $p.s_i$ in common. In the example of $CS = \{p.s_1, p.s_2, \cdots, p.s_p\}$, the following situations can occur
$< p.s_1, PRO.AGT_1, type >$: $PRO.AGT_1 = \{pro.agt_1, pro.agt_2, pro.agt_3\}$.
$< p.s_2, PRO.AGT_2, type >$: $PRO.AGT_2 = \{pro.agt_2, pro.agt_4\}$.

Phase 2: Definition of $< p.s_i, pro.agt_i, type >$ Because several provider-agents can have P-services in common ($PRO.AGT_1 = \{pro.agt_1, pro.agt_2, pro.agt_3\}$), the association of a P-service with a specific provider-agent has to be completed. In addition, because of the location criterion the P-services are treated in a serial way (i.e., one at a time). The definition of $< p.s_i, pro.agt_i, type >$ is divided into two sub-phases.

Phase 2.1 - Initially, the work starts with the P-service $p.s_{i\ (i=1)}$ of the C-service CS. At this level, only execution cost and execution time criteria are considered. Each criterion has a weight factor ($w \in [0, 1]$) that a user defines. From each provider-agent that offers the P-service $p.s_{i\ (i=1)}$, the user-agent receives the following details about that P-service: execution cost for remote and local types, and execution time (Equation 1). Because the execution time is evaluated in seconds, a function converts that time into a monetary cost.

$$User-agent:\ p.s_{i\ (i=1)} \begin{cases} pro.aget_1 : (Remote.Cost(p.s_i^1), Local.Cost(p.s_i^1)), \\ \qquad\qquad Fct.Cost(Time(p.s_i^1)) \\ \vdots \\ pro.aget_n : (Remote.Cost(p.s_i^n), Local.Cost(p.s_i^n)), \\ \qquad\qquad Fct.Cost(Time(p.s_i^n)) \end{cases} \tag{1}$$

For each offer, the user-agent selects the minimum cost between the invocation types and adds that minimum to the cost of the execution time. Furthermore, each selection criterion has a weight factor $w_{time}(Fct.Cost(Time(p.s_1)))+ w_{cost}(min(Remote.Cost(p.s_1), Local.Cost(p.s_1)))$. Finally, the user-agent selects for the P-service $p.s_{i\ (i=1)}$ the minimum offer among all the offers of the provider-agents. For example,

$$User{-}agent:\ p.s_{i\ (i=1)} \ \mapsto\ pro.agt_2 : Remote.Cost(p.s_i^2), Fct.Cost(Time(p.s_i^2)) \quad (2)$$

Based on Equation (2), $< p.s_1, pro.agt_2, remote >$ is set. The user-agent has first decided to select $pro.agt_2$ to provide $p.s_1$ and second to remotely invoke $p.s_1$. This means that the user-agent and $pro.agt_2$ will be in two different domains during the invocation process.

Phase 2.2 - After the preparation of the P-service $p.s_{i\ (i=1)}$ is finished, the user-agent starts working on the remaining P-services$_{i\,=\,2,\,\cdots,\,p}$. This time the three selection criteria are simultaneously considered. We recall that the location criterion is privileged to the other two criteria because of the previously-cited reasons. According to the location criterion, the provider-agent that will be associated with the P-service $p.s_{i\ (i\in[2,p])}$ depends on the provider-agent that has been selected to offer the predecessor P-service $p.s_{i-1}$. The user-agent proceeds according to the algorithm of Table 1 (it is assumed that $< p.s_{i-1}, pro.agt_{i-1}, type >$ is established). When the user-agent finishes working on the P-service $p.s_i$, the provider-agent and invocation strategy of that P-service are known.

5 Current work - Reliability of execution

The reliability of a Web service is the probability that a request submitted to a Web service is correctly responded within the maximum expended time frame [9]. This time frame is mostly published as part of the Web service description. Reliability is a technical measure that depends on hardware and/or software configuration of Web services and on network connections between requestors and service providers. The reliability value can be computed from historical data about past invocations using for example the number of times that a Web service has been successfully delivered within the maximum expected time frame, with regard to the total number of invocations.

Because reliability deals with service execution failures, backup approaches are deemed appropriate. A Web service cannot be executed for multiple reasons: network connection problems, service disconnected for maintenance, service overloaded, just to cite a few. In the following, we present the way reliability is integrated into the operating of the multi-domain architecture of Fig. 1.

Because we aim at interleaving Web services composition and execution, two types of agents will be required: user-agent and delegate-agent. Initially, the delegate-agent associates a Web service with a provider-agent and submits that information to the user-agent that must be running either in the user-domain or in one of the multiple provider-domains. The selected provider-agent is part

Table 1. Algorithm of provider-hosts selection

$\forall\ i,\ i = 2, \cdots, p$
for each $< p.s_i, PRO.AGT_i, type >$
if $((pro.agt_{i-1} \in PRO.AGT_i) =$ true) // does $pro.agt_{i-1}$ offer $p.s_i$?
then begin
| establish $< p.s_i, pro.agt_{i-1}, type >$ // type $p.s_i$ depends on type $p.s_{i-1}$
end
else begin
| $A \leftarrow \phi$ // set of provider-agents that are in the same domain as $pro.agt_{i-1}$
| $B \leftarrow \phi$ // set of provider-agents that are in the same domain as user-agent
| $C \leftarrow \phi$ // set of provider-agents that are in other domains
| for $(j = 0; j < \|PRO.AGT_i\|; j++)$ // $pro.agt_i \in PRO.AGT_i$
| begin
| | if domain$(pro.agt_j)$ = domain$(pro.agt_{i-1})$
| | then $A \leftarrow A \cup pro.agt_j$
| | else if domain$(pro.agt_j)$ = domain$(user - agent)$
| | then $B \leftarrow B \cup pro.agt_j$
| | else $C \leftarrow C \cup pro.agt_j$
| end // $A \cup B \cup C = PRO.AGT_i$
| if $p.s_{i-1}$ executed remotely // $< p.s_{i-1}, pro.agt_{i-1}, remote >$
| then begin
| | if $A \neq \phi$
| | then contact provider-agents of A - Go to Phase 2.1
| | else if $B \neq \phi$
| | then contact provider-agents of B - Go to Phase 2.1
| | else contact provider-agents of C - Go to Phase 2.1
| end
| else begin // $A = B$ and $< p.s_{i-1}, pro.agt_{i-1}, local >$
| | if $A \neq \phi$
| | then contact provider-agents of A - Go to Phase 2.1
| | else contact provider-agents of C - Go to Phase 2.1
| end
end

of a pool of potential provider-agents ($PRO.AGT_i$) that have a Web service in common (Equation 2). Before the delegate-agent starts working on the next component services, it stores the information about the pool of provider-agents (e.g., the x best ranked provider-agents from sets A, B, and C of $PRO.AGT_i$, $x <= (\|PRO.AGT_i\| - 1)$) for a later use. If the user-agent faces any difficulties in the execution of a service, it immediately contacts the delegate-agent which is always located in the user-domain. Because the delegate-agent is now working on the preparation of the remaining component services, it stops its preparation work and browses the stored pool of provider-agents for the service in trouble. The objective is to identify a new provider-agent, inform the user-agent about this provider-agent, and finally store the newly updated pool of potential provider-agents. Information on a pool of potential provider-agents are not deleted until the delegate-agent receives a notification message from the user-agent that the execution of a service has been successfully completed. During that confirmation exchange, the delegate-agent submits to the user-agent the details on the next service to execute.

When a delegate-agent receives a message from a user-agent in case of problems, it interrupts its operations and starts new operations as described in the above paragraph. When the delegate-agent resumes its work, it has two options. The first option consists of pursuing the suspended operations. The second option consists of dropping all the suspended operations. This is due to the location criterion that may suggest a new set of potential provider-agents to consider. We recall that the identification of a provider-agent for a service depends on the current position of the provider-agent of the direct predecessor service.

6 Related work and conclusion

In [3], Chakraborty et al. have introduced a reactive service composition architecture for pervasive computing environments. The architecture consists of five layers: network, service discovery, service composition, service execution, and application. While reviewing Chakraborty et al.'s work, we were interested in the service execution layer. During the execution of services, this layer might want to optimize the bandwidth required to transfer data over the wireless links between services and hence, execute the services in an order that minimizes the bandwidth utilization. This optimization approach is similar to the location of provider hosts criterion that we have introduced. With the location criterion, we aimed at reducing the number of remote interactions between domains, the number of migrations of the user-agent to domains, and the number of data transfer between domains. Another relevant work to the location criterion is presented in [8]. Because it will be challenging to create services that can execute well on the large variety of devices (problems of diversity and resource constraints), Messer et al. suggest to transparently offload portions of a service code from resource-constrained devices to nearby servers [8]. Code offloading requires *partitioning* strategies. If two components interact frequently (e.g., because of many method invocations), then a partitioning strategy should suggest placing these components together on one machine; splitting them across the network could

severely affect performance. The aforementioned partitioning strategy has similarities with the location of computing hosts criterion. This criterion promotes the use of local interactions between services as well as between agents. In [8], the selection of the same host may cause an overloading for that host. In our research, this situation can be avoided for two main reasons. First, the work is done at the level of domains of computing hosts and not at the level of computing hosts. Second, the location criterion helps in finding and ranking domains (sets A, B, and C in the algorithm). When a domain is considered, traditional selection criterion (such as execution cost and execution time) are applied to identify the best computing hosts, and thus the best providers of services.

We presented our work on composing services. Our long-term objective is to allow users to satisfy their needs regardless of the resources (fixed or mobile) that are involved in the execution of the services. We suggested two types of agents (user-agent and provider-agent) in order to undertake the agentification of the components of a Web service environment. We also suggested an approach for service composition that is deployed by these agents. The approach uses different selection criteria for the identification of the best component services of a composite service. Location of provider hosts is among these criteria and aims at gathering the maximum number of component services to be executed in the same computing host (i.e., using the same resource).

References

1. B. Benatallah and F. Casati (*Guest Editors*). Special Issue on Web Services. *Distributed and Parallel Databases, Kluwer Academic Publishers*, 12(2-3), September 2002.
2. B. Benatallah, Q. Z. Sheng, and M. Dumas. The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1), January/February 2003.
3. D. Chakraborty, F. Perich, A. Joshi, T. Finin, and Y. Yesha. A Reactive Service Composition Architecture for Pervasive Computing Environments. In *Proceedings of the 7th Personal Wireless Communications Conference (PCW'2002)*, Singapore, 2002.
4. 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*, 6(2), March/April 2002.
5. Z. Maamar and W. Mansoor. Design and Development of a Software Agent-based and Mobile Service-oriented Environment. *e-Service Journal, Indiana University Press*, 2003 (forthcoming).
6. Z. Maamar, W. Mansoor, and Q. H. Mahmoud. Software Agents to Support Mobile Services. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2002)*, Bologna, Italy, 2002.
7. P. Maes. Agents that Reduce Work and Information Overload. *Communication of the ACM*, 37(7), July 1994.
8. A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. J. Giuli, and X. Gu. Towards a Distributed Platform for Resource-Constrained Devices. In *Proceedings of the IEEE 22nd International Conference on Distributed Computing Systems (ICDCS'2002)*, Vienna, Austria, 2002.
9. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality Driven Web Service Composition. In *Proceedings of The Twelfth International World Wide Web Conference (WWW'2003)*, Budapest, Hungary, 2003.