Supporting Web Service Composition in Wireless Environment Based on Mobile Agents

HaiYang Hu, Hua Hu

†College of Computer Science and Information Engineering, ZheJiang GongShang University
Hangzhou City, Zhejiang Province, 310018, China

Summary
The emergence of Web services has created unprecedented opportunities for organizations to establish more agile and versatile collaborations. As mobile computing becoming a standard of modern life, Web service coordination in mobile wireless environment brings new technology challenge. This paper presents a new service coordination framework based on mobile agents. By mobile agents' melting and splitting dynamically, it can perform a fault-tolerance and efficient autonomic composition. Compared with other approaches, our scheme shows its advantage by performance tests.

Key words:
mobile agents; web service composition; melting and splitting

1. Introduction

Web Services are capabilities that enable applications and are of crucial importance to pervasive computing in next-generation networks [1]. Web services composition [1], [2], [3] is the construction of complex application from primitive Web services, thus enabling rapid and flexible creation of new application. As mobile computing becoming a standard of modern life [4], Web services composition in this wireless environments bring new technology challenge to the underlying coordination platforms. First, with the limited capability of hardware, mobile devices can't run continuously for a long time and store too much data. Second, mobile devices access the services by wireless network which has a low bandwidth and the communication cost is very expensive. Third, the wireless network can't maintain stable connection continuously. Thus, to support mobile devices composing the distributed Web services, some efficient and reliable composition mechanism will be needed. It should not need mobile devices connecting with networks continuously during Web services composition. It should be automatic enough with the least participation of clients so as to save the resource of mobile devices. And it should be fault-tolerance to adapt to the dynamic environments, e.g. some candidate Web services can't be connected or some physical nodes are shut down. At last, the composition mechanism should be efficient enough to save the overall wireless communication cost of client application on the mobile devices for the wireless communication cost is very expensive.

In this paper, we propose a new coordination framework based on mobile agents. It supports automatic Web service composition without the participation of client application, so that client application on mobile device can disconnect itself from the network when performing composition, and after a period of time reconnect again to get the results. In composition phrase, a mobile agent can split itself into several different subagents or melt other agents into a new one, thus, to perform a fault-tolerance service composition and to enhance the efficiency of the composition.

The reminder of the paper is organized as follows. In section 2, we present our system model for Web service composition in wireless environments. In section3, we describe the dependences between Web services in composition, and present the running scheme of mobile agents. Performance study is given in section 4. In section 5, the related works are overviewed. Finally, section 6 concludes this paper.

2. System Model

To lighten the structure complexity of client application on mobile devices, in our approach, the whole composition process is implemented by mobile agents in wired networks without consuming any resource of mobile client terminal. To run the mobile agents on these nodes seamlessly, agent servers are needed to deploy on these nodes. These agent servers play an important role in the service composition process [5], [6], their responsibility including generating a new instance of mobile agent, sending it to another physical node, receiving a mobile agent migrating from another node here, helping one agent split into several subagent or melt several agents into an integrity one. During service composition process, client application first sends a service composition specification file to base station. The agent server on the base station parses it into a mobile agent (also named *coordination agent*) containing an itinerary file and an assignment

Manuscript received November 5, 2007
Manuscript revised November 20, 2007

file in itself, and then the coordination agent migrates in the wired network to implement its assignments.

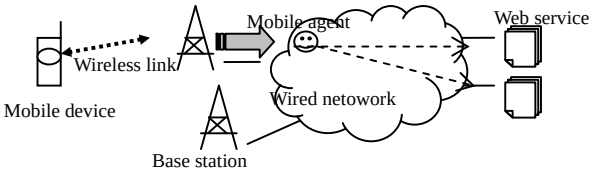

Fig.1 The system model

The detailed parsing procedure is shown in Fig.2. Including a set of candidate Web services info, the specification of Web service composition states the control flow of composition. Each Web service item in the specification may include the service's method interface, its physical location, and some of the input parameters. Among them, the interface of Web services is essential. When a candidate Web service can't be connected, the agent will search for other new Web services in the network depending on the certain specification of method interfaces. *Agent Manager* is responsible for managing the results carried back by coordination agents. After coming back, the agents will call the appropriate *put(...)* method in Agent Manager to return the results. And agent manager will send the result to client application when mobile client terminal reconnects to network again.

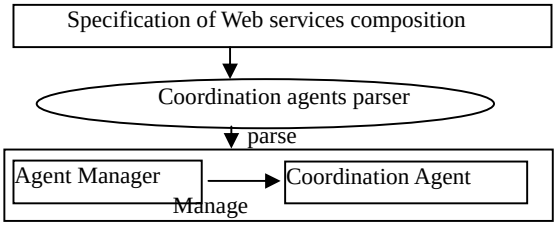

Fig.2. Generation of coordination agents

3. Framework of Web Service Composition

Using mobile agents to compose Web services, client application would represent the control flow and the structure of composition, and then tell it as a set of assignments to mobile agent. Depending on this set of assignments, mobile agent migrates in the network, splitting or melting, and at last gain its goal.

3.1 Dependences Among Web Services

In our approach, the service composition information carried by coordination agent is a tuple (*I, M, L, PM, R, Tpre, Tpost, CS*) where : *I* is the identifier of Web service; *M* is the correlated service method; *L* is the physical address of the service; *PM* is a set of input parameters of the service; *R* is the execution results of the service. *Tpre* is the set of services that this Web services depends on. *Tpost* is the set of services which depend on this service. We'll explain *Tpre* and *Tpost* below. *CS* is the current state of the service. The value of *CS* is "READY" or "NOTREADY". If the value is "NOTREADY", it means that the service can't be executed now, and must be waiting until all the conditions needed are satisfied.

There may be dependences between the services in composition. For example, one input parameter of Web service *A* may be the execution result of Web service *B*. Thus, service *A* can't be executed until service *B* has been called. We call this dependence as *parameter dependence*. Parameter dependence plays a critical role between the Web services. One Web service may be parameter dependence on several other Web services. Also, several services may be parameter dependence on one same Web service, with each depending on a certain input parameter.

Definition 1 (*Parameter dependence*). For Web service WS_i and WS_j, if one of the execution results of WS_i will influence the input parameters of WS_j, then WS_j is *Parameter dependence* on WS_i, and is denoted as $DEP(WS_i, WS_j)$.

For a set of Web services in composition given, we present an algorithm below to generate a directed parameter dependence graph (PDG), which has only one start point and one end point.

Algorithm 1:

1) For a set of Web services $ST = \{S_i, i = 1..n\}$, add one start point and one end point. Thus, $ST' = ST \cup \{S_{START}, S_{END}\}$. And $PDG = \{ (V, E) \mid V = ST',$ $E = \{ (S_i, S_j) \mid \forall S_i, S_j \in V , DEP(S_i, S_j) \}\}$. E is configured as follows:

$\forall S_i \in ST$, if there is no such service S_j in ST' that $DEP(S_i, S_j)$ holds, then $E = E \cup \{(S_i, S_{END})\}$; if there is no such Web service S in ST' that $DEP(S_i, S_j)$ holds, then $E = E \cup \{(S_{START}, S_i)\}$; if $\exists S_j \in ST'$, and $DEP(S_i, S_j)$, then $E = E \cup \{(S_i, S_j)\}$.

In PDG, the value of the directed arc (S_i, S_j) is the execution cost between service S_i and S_j. This cost includes three parts: a) the cost of agent finding the appropriate Web service S_i; b) the cost of agent migrating from the physical node of S_i to the node of S_j; c) the cost of agent executing the methods of S_j.

```
public class Assignment implements Serializable{
  private int identity;       // this assignment's identity is unique
  private Vector  Parameter;  //  the assignment's parameters, and the
  private String  methodName ; // the name that this assignment will call
                                //on the server node as the methodname;
  private Address  addr ;    // the physical address of this
                               //assignment's method;
  private Vector preAssignment;        //the assignment that will be
                              //performed directly before this assignment;
  private Vector postAssignment ;    // the assignment that will be
                                //performed directly after this assignment

  private boolean isReady;    //when ready ,that menas this assignment can
                               //be performed.
  private Vector result ;   // store the result of this as signment when has //been
                               performed
  private Vector preARC;    // the arcs directing to this assignment
  private Vector postARC;   // the arcs directing to the following assign//ments
}
```

Fig.3. The structure of Assignment implemented in Java

```
public class CoordinationAgent implements Serializable{

  private Assignment originAssignment; //an identifier given by client
  private String Identity;          // the identifier of this agent
  private Assignment currentAssignment; //this agent's current
                                        //assigenment
  private Vector    result ; // the execution result stored in agent
  private String    CS;   // the current state of this agent
     private AgentServer  as;
  public void run(AgentServer as){}
     /* when agent migrates to the physical node of its current assigenment,
     it tries to perform the execution */
  private void performAssignment(Assignment cua ){}
     /*  split the agent corresponding to the postassignment of current
       assignment,and put the new agentresponsible for each assignment; */
   private void splitAgent(Vector postAssignment){}

      /* let the agent know which assignment it will want to perform
   */
   public void setCurrentAssignment(Assignment ass){}
   public Assignment getCurrentAssignment(){}

      /* If the agent's currentAssignment is ready then the agent is
      ready  */
   public boolean isReady(){}
}
```

Fig.4 The structure of coordination agent implemented in Java

We denote the cost of a) as $TF(S_j)$, the cost of b) as $TM(S_i.L,S_j.L)$, here, $S_i.L$ being the physical address of service S_i and the cost of c) as $TS(S_i)$. If S_i and S_j are distributed on the same physical node, then the value of $TM(S_i.L,S_j.L)$ equals to zero. Fig.3 gives the structure of Assignment in PDG implemented in Java language.

3.2 Structure of Coordination Agent

In our approach, the coordination agent is a tuple (G, I, CT, CS, F) where: G is the PDG carried by agent; I is the identifier of this agent; CT is the current service executed by agent; CS is current state of agent. Its value is "WAITING", "NOTREADY" or "READY"; F is the agent's function body.

3.3 Running of Coordination Agents

To gain a high efficient composition, coordination agent will dynamically split and melt to compose the services as parallel as possible, so that the total cost will be minimized. We define agent splitting and melting as follows.

Definition 2 (*Dynamically splitting*). For PDG g carried by agent A, $|g.V|=k$, there are no such two services in $g.V$ that S_i, $S_j \in g.V$, and $DEP(S_i,S_j)$. Then

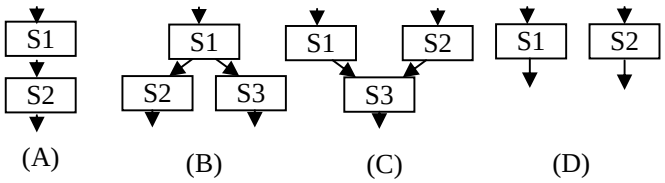

Fig.5 Four structures of services composition

$SPL(A \mid g.V) = \{A_i \mid 1 \le i \le k\}$, and for each subagent A_i, $A_i.I$ is unique, $A_i.CT = S_i \in g.V, 1 \le i \le k$.

Definition 3 (*Dynamically melting*)**.** For Agent A_i, A_j if $A_i.CT = A_j.CT = S_C$, $A_i.CS$ = "WAITING", $A_j.CS$ = "NOTREADY", then $A_N = MEL(A_i, A_j \mid S_C)$, $A_N.I$ is unique, $A_N = A_i \cup A_j$, and $A_N.CT = S_C$. If $S_C.CS$ = "READY", then $A_N.CT$ = "READY".

In PDG, there are four possible kinds of control flow between the Web services as shown in Fig.5. In (A), service S_2 only depends on S_1 and there is only one service S_2 depending on S_1. In this case, coordination agent sequentially executes S_2 after the execution of S_1. In (B), both S_2 and S_3 depend on S_1. In this case, after the execution of S_1, agent A_1 splits itself into two subagents A_{12}, A_{13}, and then A_{12} begins to execute S_2 and A_{13} to execute S_3 respectively. In (C), S_3 depends on both S_1 and S_2. In this case, suppose there are two agents A_1, A_2 executing services S_1 and S_2 respectively. When agent A_1 and A_2 finish their execution of S_1 and S_2, they melt into a new agent A_3 with the help of agent server, and A_3 starts to executing service S_3. In (D), neither S_1 nor S_2 depends directly or indirectly on the other. And in this case, as shown in (A)-(C), there are two different agents in the environment to call them in parallel.

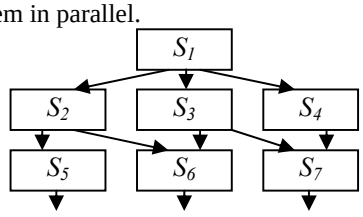

Fig.6 An example of service composition

For example, in fig.6, the seven services in such a control structure are represented in the PDG. A coordination agent A_1 parsed by agent server depending on this PDG is sent out to fulfill the composition. After calling service S_1, A_1 splits itself into three subagents A_{12}, A_{13} and A_{14}, with each to call one of the following services S_2, S_3 and S_4 respectively, so that the three services are executed in parallel. And after that, agent A_{12} splits into A_{125} and A_{126} to execute S_5 and S_6 in parallel. Agent A_{13} splits into A_{136} and A_{137} to execute S_6 and S_7 in parallel. As S_6 is parameter dependence on S_2 and S_3, neither A_{126} nor A_{136} can fulfill the execution of S_6 alone. So, while calling service S_6, agent A_{126} and A_{136} melt together into a new agent A_6 to fulfill the execution. And agent A_{137} and A_{14} melt into agent A_7 to call the service S_7. After calling the services S_5, S_6 and S_7, the three agents A_{125}, A_6 and A_7 proceed with their execution in parallel. From this example, we can see that the services are composed as parallel as possible to save the total cost.

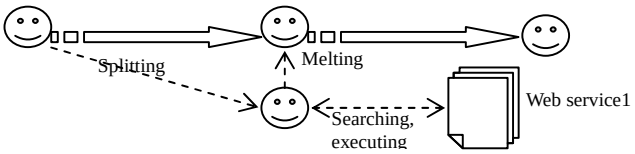

Fig.7. Fault-tolerance searching and executing

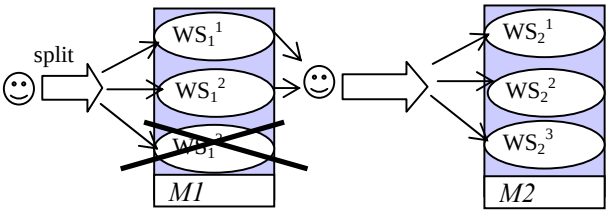

Fig.8. Fault-tolerance searching and executing

In the dynamic environments, often some candidate Web service in composition are shut down and can't be connected, thus, coordination agent must dynamically find some other Web service having the matching behavior in the environment to perform a fault-tolerance composition at runtime. As shown in Fig.7 and 8, to realize a fault-tolerant composition, the agent dynamically splits into multiple subagents to search the environment for the candidate Web services providing the same service method. And during this process, the failure of several subagents won't prevent the coordination agent from proceeding with its composition. As in figure 9, there being three candidate Web services (WS_1^1, WS_1^2, WS_1^3) providing the same service method *M1*, coordination agent splits itself into three subagents, each calling a certain Web services. After the calling, these subagents will again melt into a new one to call the following Web services.

Based on the rules of *interface compatibility* and *behavior compatibility*, coordination agent *A* searches the new web service *S* in environment to substitute the candidate service in the PDG , which can't be accessed.

Definition 4 (*interface compatibility*). For *interface compatibility*, *S* provides at least the service interfaces methods required by *A*. And $\forall$ *ipa1*: input parameter of request interface method in *A*, *ipa2*:the corresponding parameter of service interface method of *S*, then *ipa1* has the same type of *ipa2*, or *ipa1* is a subtype of *ipa2*. The requirement result type in *A* is the same type of the result type of *S*, or a supertype of that in *S*.

Definition 5 (*behavior compatibility*). For *behavior compatibility*: first, each request interface method in *A* must have a corresponding service interface method in *S*, and the two interfaces are *interface compatibility*; Second, let L_A be the set of request method traces in *A*, L_S be the set of service method traces of *S*, and $L_A \subseteq L_S$. Thus, *S* is behavior compatibility to *A*'s requirements.

3.4 An Efficient Composition Scheme

In this section, we propose an efficient Web service composition scheme based on coordination agents' dynamically splitting and melting. We fist define the concepts of $T(Num(\rho_{SPL}))$ and $T(Num(\rho_{MEL}))$, and then provide the algorithm.

Definition 6 (*Splitting point*). Suppose $\rho =< S_{START}, S_1, S_2, \ldots S_k, S_{END} >$ is a directed path in *PDG*, if for some $S_i \in \rho$, $1 \le i \le k$, $\exists S_j \in V$, $(S_i, S_j) \in E \wedge S_j \notin \rho$, then we call S_i a *split point* of ρ. Split point states that when agent finishes executing S_i, it will split into several subagents to execute the following services. Suppose $Num(\rho_{SPL})$ is the total number of split points in ρ, then $T(Num(\rho_{SPL}))$ is the overall cost of agent performing all split operations along path ρ, then it's easy to see that $T(Num(\rho_{SPL})) = c_1 Num(\rho_{SPL})$, and c_1 is a constant.

Definition 7 (*Melting point*). Suppose $\rho =< S_{START}, S_1, S_2, \ldots S_k, S_{END} >$ is a directed path in *PDG*, if for some $S_i \in \rho$, $1 \le i \le k$, $\exists S_j \in V$, $(S_j, S_i) \in E \wedge S_j \notin \rho$, then we call S_i is a *melting point* of ρ. Melting point states that when agent finishes executing S_{i-1} along path ρ, it will wait for other agents coming from other paths here and melt them together into a new agent to execute S_i. Suppose $Num(\rho_{MEL})$ is the total number of melting points in ρ, $T(Num(\rho_{MEL}))$ is the overall cost of agent performing all melting operations along path ρ, then it's easy to see that $T(Num(\rho_{MEL})) = c_2 \ Num(\rho_{MEL})$, and c_2 is a constant.

Theorem. Suppose coordination agent carries such a *PDG* $g =< V, E >$, and the set of all the critical paths [7] in g is $\{\rho_k \mid 1 \le k \le m\}$. $\exists \rho_j \in \{\rho_k \mid 1 \le k \le m\}$, $\rho_j = < S_{START}, S_j^1, S_j^2, \ldots S_j^n, S_{END} >$, and $T(Num(\rho_{j(MEL)})) + T(Num(\rho_{j(SPL)})) = Min_k\{T(Num(\rho_{k(MEL)})) + T(Num(\rho_{k(SPL)}))\}$. So the cost of coordination agent fulfilling the overall composition in PDG is at least $T(\rho_j) + T(Num(\rho_{j(MEL)})) + T(Num(\rho_{j(SPL)}))$. Here, $T(\rho_j)$ is the cost of one single agent executing along path ρ_j from the beginning node to the end node without considering the cost of melting and splitting.

Proof With the concept of critical path, it's easy to prove this theorem.

Based on the theorem, we present an efficient algorithm for Web service composition.

Algorithm 2:

1) For a set of Web service to be composed, use algorithm 1 to generate the interrelated PDG. Agent server sends out an agent *A* carrying the PDG to fulfill the composition.
2) Generate the set of *critical paths* in PDG. If there is only one such critical path $\rho =< S_1, S_2, \ldots >$, then for agent *A*, it sets its current assignment $A.CT = S_1$; If there are several critical paths, $\{\rho_i \mid 1 \le i \le m\}$, then select such a path ρ_j that $Num(\rho_{j(MEL)}) + Num(\rho_{j(SPL)}) = Min_i\{Num(\rho_{i(SPL)}) + Num(\rho_{i(MEL)})\}$, $1 \le i \le m$. Suppose such a path $\rho_j =< S_1, S_2, \ldots >$, then the agent sets its current assignment $agent.CT = S_1$.
3) If the agent's current assignment $CT = S_{START}$, then goto 8); If If the agent's current assignment $CT = S_{END}$, then goto 9); If it's current assignment $CT \ne S_{START} \wedge CT \ne S_{END}$, goto 4).
4) Coordination agent migrates to the physical address of its current assignment.
   a) If the current state *CS* of agent's current assignment *CT* is "NOTREADY", then agent applies to the local agent server for entering into the waiting queue, goto 5).
   b) If the current state *CS* of agent's current assignment *CT* is "READY", then agent call the service methods of the Web service. After that, goto 8).
5) Agent server searches local waiting queue, if it finds such an agent *agent1* that has originated from the same client application and is waiting to execute the same service methods, then goto 6); If agent server can't find such an agent, then goto 7).
6) Agent server helps the two agent to melt together,

suppose *agent2=MEL(agent, agent1 |agent.CT)*; After the melting, if the current state *CS* of *agent2's* current assignment *CT* is "READY", agent2 will execute its current assignment, and then goto 8); If the value of *CS* is still "NOTREADY", goto 7).

7) Agent server adds the agent into its local waiting queue.

8) For the current assignment *CT* executed,
   a) if $|CT.Tpost|>1$, then the agent *A* splits itself into several subagents. Suppose the set of subagents $\{A_i \mid 1 \le i \le |CT.Tpost|-1\} = SPL(A \mid CT.Tpost \neq Next(\rho(CT)))$, here $Next(\rho(CT))$ is the following service of *S* in path ρ, for $\forall A_i \in SPL(A \mid CT.Tpost \neq Next(\rho(CT)))$, goto 2); While for agent *A*, it sets its current assignment $A.CT = Next(\rho(CT))$, then goto 3).
   b) if $|CT.Tpost|=1$, then the agent sets its current assignment $A.CT = Next(\rho(CT))$, then goto 3).

9) Coordination agent migrates back to the agent manager and puts back the results.

4. Performance Study

We use IBM NetVistas to run Client application, and use a set of Dell PowerEdge 1400SC servers to run as agent Servers. The Web services are also distributed on these servers with the connection of 100MB/S Ethernet network. Client application connects to the network with a common 10Kb modem to simulate as the mobile device.

Figure 9 shows the cost of dynamically agent splitting. The splitting cost includes dynamically creating instances of subagent, setting the correlated assignments and sending them out. Figure 10 shows the cost of agents' dynamically melting. The process includes agent server searching the wait queue, and agent server helping the agents melting together into an absolutely new agent. As seen from the figures, both agents' melting and splitting activities cost little.

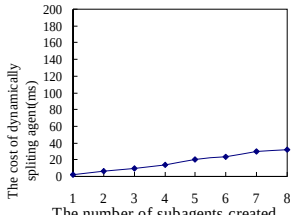

Fig.9 Agent dynamically splitting

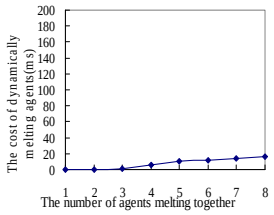

Fig.10 Agents dynamically melting

In the second experiment we compare the performance of our approach with the traditional Axis Web service composition framework using standard SOAP protocol both in wired and wireless networks (as shown in fig. 11 and 12). In the former, both client application and web services are located on wired networks. While in the latter, client application is on a mobile device. In this scenario, by the approach of agents, client application sends a specification of the Web service composition to one of the agent servers, the server parses it into a coordination agent, then the coordination agent fulfill the overall composition by dynamically splitting and melting. During the process, client application can disconnect itself from the network freely. By the approach of SOAP, client application uses RPC-SOAP protocol to call all the Web services one by one, while the network connection needing to be maintained continuously. In this experiment, we compare the time cost of the two approaches at the different level of application payloads (500 Bytes, 5 KBytes, and 50 KBytes). And the result is shown in Fig.11 and 12. Seen from the figures, our approach is greatly efficient than the Axis composition approach in wireless networks. And this is because, in our approach, the unnecessary network transporting from mobile device to the Web services in wired network is avoided and agents can perform the composition in the network by itself. However, in wired networks, the advantage of agent approach is not very apparent because of the cost of agents' migration between the nodes in wired networks.

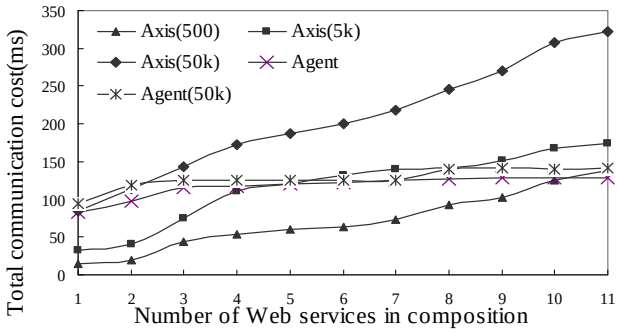

Fig.11 Compared with SOAP protocol in wired network

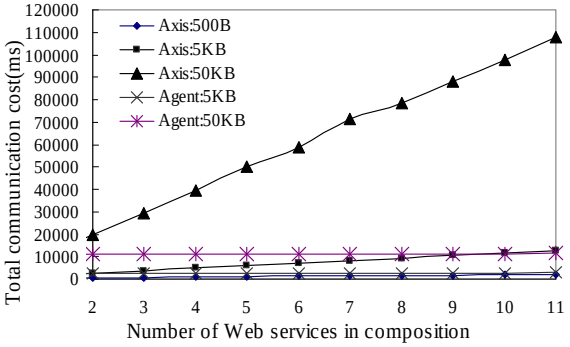

Fig.12 Compared with SOAP protocol in wireless network

5 Related Works

Research on Web service composition has going on from the last decade. An architecture for service composition in pervasive computing environment is proposed in [8]. In their work, service descriptions include platform specific information such as processor type, speed and memory availability. In [9], the system uses a rule-based exert system to automatically determine whether a desired composite service can be achieved using existing services. CMI [10] and eFlow [11] have investigated the possibility of performance dynamic service selection based on user requirements. CMI's service definition model states the concept of a placeholder activity to cater for dynamic composition of services. In [12], SAHARA system proposed a model for service composition which recognizes two different models: the cooperative composition model and the brokered composition model. The work proposed In [13] implemented dynamic QoS-aware service composition selection, Thus, the service composition manager in the architecture acts as a broker between the composite service clients and the services participating the composition.

However, none of the works concern on how to optimize the total cost of service composition, and most of them need the client's participating in during the service composition. Thus, they can't best satisfy the requirements of service composition in wireless environment.

6 Conclusion

This paper presents a new Web service composition framework in wireless environment based on mobile agents. It supports automatic Web service composition without the participation of client application, so that the mobile client terminal can disconnect itself from the network,. During composition procedure, the coordination agent can dynamically split itself into several different subagents or melt other agents into a new one to perform a fault-tolerance and high efficient composition.

Reference

[1] G. Alonso, F.Casati, H.Kuno, V.Machiraju. Web Services. Springer Verlag, 2003.
[2] B.Benatallah, M.Dumas, Q.Z.Sheng, A.H.Ngu. Declarative composition and peer-to-peer provisioning of dynamic Web services. ICDE 2002,297-308
[3] B.Medjahed, A.Bouguettaya, et al. Composing Web services on the semantics Web. The journal of VLDB, 2003, 12(4): 333-351.
[4] Chander Dhawan. Mobile Computing: A systems Integrator's Handbook. McGraw-Hill, USA.
[5] Hu Hai-Yang, Yang Mei, Tao Xian-Ping, Lv Jian. Research and implementation of late assembly technology in Cogent. ACTA ELECTRONICA SINICA, 2002, 30(12): 1823~1827(in Chinese)
[6] Lv Jian, Zhang Ming, Liao Yu, Tao Xian-Ping. Research on componentware framework based on mobile agent technology. Journal of Software, 2000, 11(8):1018~1023(in Chinese)
[7] Bruno R, Preiss. Data structures and Algorithms with Object-Oriented Design Patterns in C++. John Wiley & Sons, 1998
[8] Chakraborty D, Perich F, Joshi A, Finin T. A reactive service composition architecture for pervasive computing environments. In proceedings of the 7th personal wireless communications conference, Singapore 2002, 53-62.
[9] Ponnekanti SR, Fox A. SWORD: A developer toolkit for Web service composition. In proceedings of the 4th international ACM workshop on Web information and data management, McLean, VA, 2002, 56-62.
[10] D. Georgakopoulos, H.Schuster, A.Cichocki, and D.Baker. Managing Process and Service fusion in virtual enterprises. Information system, special issue on Information system support for electronic commerce, 1999, 24(6):429-456.
[11] F.Casati, M.C.shan. Dynamic and adaptive Composition of E-Services. Information Systems, 2001, 6(2): 143-162.
[12] B.Raman, S.Agarwal, Y.Chen, M.Caesar, et al. The SAHARA model for service composition across multiple providers. In proceedings of first international conference on pervasive computing, 2002, 1-14.

Haiyang Hu received the B.E. and M.E. degrees, from Nanjing Univ. in 2000 and 2003, respectively. He received the Dr. Eng. degree from Nanjing Univ. in 2006. He is now working as a teacher (from 2006) at Zhejiang GongShang University. His research interest includes software engineering and Internet computing. He is a member of CCM.

Hua Hu received the B.E., M. E., and Dr. Eng. degrees from Zhejiang University. He is now working as a full professor at Zhejiang GongShang University. His research interest includes software engineering and Internet computing.