A Distributed Algorithm for Web Service Composition based on Service Agent Model

Hongxia Tong, Jian Cao, Shensheng Zhang, and Minglu Li

Abstract—Agent-based service composition has provided a promising computing paradigm for the automatic Web service composition. In this paper, a formal service agent model is proposed, which integrates the Web service and software agent technologies into one cohesive entity. Based on the service agent model, a distributed planning algorithm for Web service composition called DPAWSC is presented. DPAWSC formalizes Web service composition into a graph search problem according to the dependence relations among service agents. The key to DPAWSC is that the alternative solution with smaller length has higher priority to be searched than one with larger length. DPAWSC is based on the distributed decision making of the autonomous service agents and addresses the distributed nature of Web service composition. We evaluate the algorithm by simulation experiments and the results demonstrate that DPAWSC is effective for its ability to produce the high quality solution at a low cost of communications.

Index Terms—distributed planning algorithm, Web service composition, software agent, service agent model.

1 INTRODUCTION

Economic globalization is transforming the global market into an interdependent and mutual influenced organic unity that requires unprecedented interoperability to efficiently integrate diverse information systems to share knowledge and collaborate among organizations. As a new distributed computing mode, Web service is an emerging paradigm in which very loosely coupled software components are published, located and invoked on the Web as parts of distributed applications [6]. The main objective of the Web service paradigm is to achieve interoperability between heterogeneous and distributed applications. Web service has been identified as the basic technical building block for the next generation of web-based business solutions, which is featured with application, platform and provider independence. From its emergence, many specialists have predicted that Web service will revolutionize the distributed computing paradigm.

One of the most important values of employing web service technologies is the composition of Web services to create value added service. Web service composition refers to the mechanisms that promote the collaboration of individual Web service to create software applications with a functionality that is the result of integration of the individual functionalities provided by each participating service, which has the potential to reduce development time and effort for new applications [36].

Web service composition has received great interest [10][21][28][29][33]. However, these centralized Web service composition approaches suffer from performance bottleneck and single point of failure. Web services are distributed across physical and geographical boundaries, they are constantly removed and updated. Therefore, any solution must support an equivalent degree of distribution. According to the reports of IDC research group [24][34], worldwide software spending for SOA-based initiatives and SOA-driven professional services engagements are expected to grow to nearly $14 billion and $40.8 billion by 2011 respectively. With the popularity of Web services on Internet, the distributed Web service composition is inevitable.

As an important paradigm for organizing many classes of distributed applications, software agents can promote the interoperability and collaboration among diverse organizations for their inherited characteristics of autonomy, sociality, communication, cooperation and pro-activeness [25]. Software agent can be considered as a complementary paradigm to the current Web service technologies [31]. Adopting software agent extends the Web service technologies in some important issues:

1) Web services are just self-describing software components, and they have knowledge just about themselves. In contrast, software agents have domain knowledge, by which they can reason and interact with outside world.

2) Web services are passive components. In contrast, agents are pro-active entities that can find the other components and communicate with them.

3) In order to use the potential Web services, the requesters are required to analyze the contents of Web service description to find out how to use the Web...
service. In contrast, agents have the ability to map the user requirement to the existing methods.

In this paper, a formal service agent model is proposed, which integrates the Web service and software agent technologies into a cohesive entity. The service agent model aims to exploit agent’s capabilities to enhance Web service’s behaviors. The operation template in the service agent model is used to abstract the features of a group of operations with similar functions, which shields the heterogeneity of distributed Web services. Meanwhile, the plan in the service agent model is used to encapsulate the business logics of how to use the operation templates to achieve a certain business goal. Therefore, common users only need to focus on what they want to achieve rather than how to achieve.

Based on the service agent model, a distributed planning algorithm for Web service composition called DPAWSC is proposed. DPAWSC formalizes Web service composition into a graph search problem according to the dependence relations among service agents. The key to DPAWSC is that the alternative solution with smaller length has higher priority to be searched than one with larger length. The distinctive features of DPAWSC are twofold. Firstly, DPAWSC is fully distributed, which is based on the distributed decision making of the autonomous service agents and has better scalability. Secondly, taking the search result into account, DPAWSC can produce the high quality solution at a low cost of communications.

The contributions that we have made in the paper are highlighted as follows.

1. A formal service agent model is proposed, which integrates the Web service and software agent technologies into a cohesive entity.
2. DPAWSC is proposed, which is based on the distributed decision making of the autonomous service agents and addresses the distributed nature of Web service composition.
3. We have evaluated the algorithm by simulation experiments and the results demonstrate the efficiency of DPAWSC.

The remainder of this paper is organized as follows. After introducing the service agent model in section 2, the life cycle of Web service composition based on the service agent model is presented in section 3. Section 4 describes the dependence relation among service agents. DPAWSC is discussed in section 5 and experimental results are shown in section 6. The prototype system and a case study are presented in section 7. The related work is presented in section 8. Finally, section 9 concludes the paper with an outlook to future work.

2 **SERVICE AGENT MODEL**

The service agent model combines the Web service technologies and software agent technologies into a cohesive entity. By using this model, each service agent is a software agent that manages a group of Web services with similar functions in a particular business domain and offers the self-managed Web services to respond to the requests from the environment [12]. The structure of service agent model is illustrated as Fig. 1.

![Fig. 1. The structure of service agent model](image)

The service agent model is composed of four fundamental elements that are Beliefs, Goals, Actions and Plans. Beliefs represent the current state of the agent’s internal and external worlds. Goals are the set of goals that the service agent wants to achieve. Actions are the set of actions that the service agent is able to perform. Plans are the set of plans that the service agent has, which denote the capabilities of the service agent. Through the execution and monitor component, the service agent can communicate with the environment and react to the environment autonomously.

The service agent is defined based on the domain ontology. Ontology is an explicit specification of a conceptualization and its importance has been well recognized [39]. The domain ontology is structured as a set of individual generalization hierarchy terminology trees, with the more abstract concepts of the ontology forming the root terms of which other terms are specified. By the domain ontology, the definition of service agent can be shared and reused.

2.1 **Belief Model**

The set of beliefs is the knowledge base for the service agent, which denotes the knowledge about itself and the environment. The knowledge of the service agent is classified into three categories that are basic knowledge, constraint knowledge and social knowledge. The basic knowledge is the facts that the service agent knows. Constraint knowledge is a set of relationships among service agents and the social knowledge consists of other service agents’ information such as their capabilities and addresses. In order to represent these three categories of knowledge, the set of beliefs are divided into three sub-models that are world model, constraint model and acquaintance model respectively [13].
Basic knowledge is denoted as tuple \((n, t, v, u)\), where \(n\) is the name, \(t\) is the type, \(v\) is the value and \(u\) is the unit measure for the basic knowledge.

Constraint knowledge is denoted as tuple \((cT, cO)\), where \(cT\) is the constrain type and \(cO\) is the constrained objects set which can include one or more elements.

Social knowledge is denoted as tuple \((aN, aA, aC)\), where \(aN\) is the acquaintance’s name, \(aA\) is the acquaintance’s address and \(aC\) is the acquaintance’s capabilities.

Basic knowledge defines all parameters that support the execution of service agent. Constraint knowledge defines the constraint relations among service agents, which is an important aspect of quality optimization [1]. By the social knowledge, dependence relations among service agents can be constructed.

### 2.2 Action Model

There are altogether three types of actions for the service agent: internal action, communication and operation template. The operation template is a special action, which is used to characterize the features of actual operations with similar functions provided by different providers. The service provider can register the actual operation of Web service as the instance of the operation template according to the similarity between the actual operation and the operation template.

**Definition 1.** The operation template is defined as tuple \((\text{OperationName}, \text{Inputs}, \text{Outputs}, \text{Function}, \text{Category}, \text{Quality})\), where:

1. \(\text{OperationName}\) defines the name of the operation template.
2. \(\text{Inputs}\) and \(\text{Outputs}\) define the input and output parameters of the operation template.
3. \(\text{Function}\) is the business functionality provided by the operation template.
4. \(\text{Category}\) defines the operation template’s area of interest.
5. \(\text{Quality}\) is a group of quality criteria to denote the non-functional attributes of the operation template.

In the above definition, the former five attributes denote the function of the operation template and the last one denotes the non-functional attributes of the operation template.

### 2.3 Plan Model

The plan of the service agent encapsulates the business logics of how to use operation templates to achieve a certain business goal, which dictates how the operation templates can be combined, synchronized and coordinated. Each plan denotes one capability that the service agent has.

**Definition 2.** The plan is defined as tuple \((\text{Os}, \text{Goal}, E_p, D_p)\), where:

1. \(\text{Os}\) is a set of operation templates.
2. \(\text{Goal}\) is the business goal that the plan achieves, which is denoted as tuple \((\text{Inputs}, \text{Outputs})\), where the \(\text{Inputs}\) and \(\text{Outputs}\) denote the input parameters and output parameters of the plan respectively.
3. \(E_p\) and \(D_p\) are the control flows set and data flows set respectively.

Given a plan denoted as \(p\), the following conditions are satisfied:

1. For any operation template \(ot_i\) without precursor, \(\text{GetInputs}(ot_i) \subseteq \text{GetGoalInputs}(p)\), where the functions of \(\text{GetInputs}(ot_i)\) and \(\text{GetGoalInputs}(p)\) return the input parameters of the operation template \(ot_i\) and the plan \(p\) respectively.
2. For any operation template \(ot_j\) with precursor, \(\text{GetInputs}(ot_j) \subseteq (\text{GetGoalInputs}(p) \cup (\cup_{ot_k\in \text{Pre}(ot_j)} \text{GetOutputs}(ot_k)))\), where the functions of \(\text{Pre}(ot_j)\) and \(\text{GetOutputs}(ot_k)\) return the precursors of the operation template \(ot_j\) and the output parameters of the operation template \(ot_k\) respectively.
3. \((\cup_{ot_k\in Os} \text{GetOutputs}(ot_k)) \supseteq \text{GetGoalOutputs}(p)\), where the functions of \(\text{GetGoalOutputs}(p)\) returns the output parameters of the plan \(p\).

The control flows of plan specify the logic relationships among the operation templates and the data flows signify the types of message dependency among operation templates in plan. The control flows in conjunction with data flows provide a sound basis for developing the business logics in the plan.

The service agent model has one to one mapping with the famous and widespread BDI agent architecture developed by Rao and Georgeff [3]. The beliefs of service agent capture the informational attitudes of BDI agent. The goal of service agent represent the concrete motivation that the service agent desires to achieve, which captures the motivational attitudes of BDI agent. The plan of service agent represents the concrete actions to achieve the goal, which captures the deliberative attitudes of BDI agent.

### 3 Life Cycle of Agent Based Web Service Composition

In terms of the above service agent model, the agent based Web service composition proceeds through four stages: requirement definition, distributed service agent planning, quality optimization and execution, which is denoted as Fig. 2.

**Fig. 2.** Life cycle of agent based Web service composition
The service requirement is denoted as tuple \((r^i, r^o, qos)\), where \(r^i\) is the set of input parameters that the user provides, \(r^o\) is the set of output parameters that the user desires and \(qos\) is the specification of quality requirement. \(r^i\) and \(r^o\) are the functional requirement and \(qos\) denotes the non-functional requirement.

The second stage is distributed service agent planning, which is used to form a plan workflow. 

**Definition 3.** The plan workflow is defined as 
\[ PlanWF = (Ps, Ep, Dp), \]
where \(Ps\) is a set of plans, \(Ep\) and \(Dp\) are the set of the control flows and data flows among plans in \(Ps\) respectively.

For a given service requirement, there are a group of service agents \(Sa = \{sa_1, sa_2, \ldots, sa_n\}\), and the capabilities of the service agent are the plans that it has. The function of distributed service agent planning is to form a plan workflow \(PlanWF = (Ps, Ep, Dp)\) such that:

1. For the plan \(pi \in Ps\) without precursor, \(GetGoalInputs(pi) \subseteq r^i\).
2. For the plan \(pj \in Ps\) with precursor, \(GetGoalInputs(pj) \subseteq ((\cup p_i \in prec(pj)) \cap GetGoalOutputs(pi)) \cup r^i\).
3. \(r^o \subseteq (\cup p_i \in Ps \cap GetGoalOutputs(pi))\).

The plan workflow does not refer to any service instance and can not be directly executed, which is usually called the abstract workflow [5].

The function of the quality optimization stage is to bind the operations of Web services to the operation templates of plans contained in the plan workflow according to the quality requirement. The outcome of this stage is usually called the composite Web service.

In the last stage, the composite Web service is executed by the service agents and the results are returned to user.

This paper is focused on the distributed service agent planning stage. Discussion on quality optimization [22][38] is out of scope of this paper due to the space limitation.

4 Dependence Relation Among Service Agents

An agent in the multi-agent system is said to be dependent on another if the latter can help or prevent him to achieve one of his goals [18]. As no agent involved in the multi-agent system is assigned a privileged role, the dependence relation between agents enables the agent to adapt to an evolving environment and to take into consideration of the other members. In terms of the above service agent model, the dependence relation among service agents is defined [9].

**Definition 4.** Given two service agents denoted as \(sa_i = (Beliefs_i, Actions_i, Goals_i, Plans_i)\) and \(sa_j = (Beliefs_j, Actions_j, Goals_j, Plans_j)\), if there exist two plans \(p_p \in Plans_i\) and \(p_q \in Plans_j\) satisfying 
\[ GetGoalOutputs(p_q) \cap GetGoalInputs(p_p) \neq \phi, \]
then the plan \(p_p\) of \(sa_i\) is said to be dependent on the plan \(p_q\) of \(sa_j\), which is denoted as \(Dep_{sa_i}(sa_i, p_p, sa_j, p_q, x)\), if \(x = GetGoalOutputs(p_q) \cap GetGoalInputs(p_p)\).

For a dependence relation \(Dep_{sa_i}(sa_i, p_p, sa_j, p_q, x)\), if \(x = GetGoalInputs(p_p)\), then it is called a fully dependence relation, otherwise, it is called a partially dependence relation. Fully dependence means that the output parameters of \(p_p\) can fully match the input parameters of \(p_q\), and the partially dependence means that the output parameters of \(p_q\) can only partially match the input parameters of \(p_p\).

### TABLE 1
Goals of plans

<table>
<thead>
<tr>
<th>Goal of GetConferenceInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Inputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;conferenceName&quot; type=&quot;xs: String&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Inputs&gt;</td>
</tr>
<tr>
<td>&lt;Outputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;conferenceDate&quot; type=&quot;xs: Date&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;conferenceAddress&quot; type=&quot;xs: Address&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Outputs&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal of GetFlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Inputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;departureDate&quot; type=&quot;xs: Date&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;departureCity&quot; type=&quot;xs: Address&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;destinationCity&quot; type=&quot;xs: Address&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Inputs&gt;</td>
</tr>
<tr>
<td>&lt;Outputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;flightInfo&quot; type=&quot;xs: FlightDetail&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Outputs&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal of GetHotel</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Inputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;checkinDate&quot; type=&quot;xs: Date&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;city&quot; type=&quot;xs: Address&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Inputs&gt;</td>
</tr>
<tr>
<td>&lt;Outputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;hotelInfo&quot; type=&quot;xs: HotelDetail&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Outputs&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal of PlanTrip</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Inputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;departureDate&quot; type=&quot;xs: Date&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;departureCity&quot; type=&quot;xs: Address&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;destinationCity&quot; type=&quot;xs: Address&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;userName&quot; type=&quot;xs: String&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;password&quot; type=&quot;xs: String&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Inputs&gt;</td>
</tr>
<tr>
<td>&lt;Outputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;flightInfo&quot; type=&quot;xs: FlightDetail&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;hotelInfo&quot; type=&quot;xs: HotelDetail&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Outputs&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal of ReserveTrip</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Inputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;flightInfo&quot; type=&quot;xs: FlightDetail&quot;&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;hotelInfo&quot; type=&quot;xs: HotelDetail&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Inputs&gt;</td>
</tr>
<tr>
<td>&lt;Outputs&gt;</td>
</tr>
<tr>
<td>&lt;part name=&quot;tripArrangement&quot; type=&quot;xs: TripDetail&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/Outputs&gt;</td>
</tr>
</tbody>
</table>

By using the dependence relation, the dependence graph among service agents can be constructed. Formally, the dependence graph is a directed graph \(DRG(V, DRG, E(DRG))\) with the following characteristics:

1. \(V(DRG) = V_{sa}(DRG) \cup V_p(DRG)\), where \(V_{sa}(DRG)\) is the set of service agents, and \(V_p(DRG)\) is
the set of plans that the service agents have.

(2) \( E(DRG) \) is the set of edges.

For an agent community with five service agents named as ConferenceAgent, FlightConsultant, HotelConsultant, TripAssistant and TripReservationAgent. Each Service agent has one plan and their plans are GetConferenceInfo, GetFlight, GetHotel, PlanTrip and ReserveTrip respectively. The goals achieved by these plans are defined in table 1:

1. Given the conference name, GetConferenceInfo plan of ConferenceAgent returns the conference date and conference address of the conference.
2. Given the departure date, departure city and destination address, GetFlight plan of FlightConsultant recommends a flight for user.
3. Given the check in date and city name, GetHotel plan of HotelConsultant recommends a hotel for user.
4. Given the departure date, departure city, destination city, user name and password, PlanTrip plan of TripAssistant recommends the flight and the hotel for user.
5. Given the detailed information of the flight and the hotel, ReserveTrip plan of TripReservationAgent reserves the flight and the hotel for user.

According to the definition of dependence relation, the dependence graph is constructed. Only dependence relations among plans are illustrated for concise representation.

![Dependence Graph](image)

**Fig. 3. Dependence graph**

### 5 DPAWSC

#### 5.1 Introduction of DPAWSC

In order to clearly introduce DPAWSC, the following concepts are defined.

**Definition 5.** For a service requirement, if there is a set of service agent-plan-parameters triples denoted as \( S = \{(a_{s_1}, p_{s_1}, x_1), (a_{s_2}, p_{s_2}, x_2), \ldots, (a_{s_n}, p_{s_n}, x_n)\} \) satisfying \( r^a \subseteq (x_1 \cup x_2 \cup \ldots \cup x_n) \), then \( S \) is called the solution for the service requirement. For any element \( e \in S \), if \( S - e \) is not the solution for the service requirement, then \( S \) is called the minimal cover solution.

The number of elements contained in the minimal cover solution is called the length of the minimal cover solution. All the minimal cover solutions are alternative solutions for the service requirement.

**Definition 6.** For the service agent \( sa_i \), if there is a set of dependence relations denoted as \( Ds = \{Dep_{sa_i}(s_{a_1}, p, sa_1, p, x_1^1), Dep_{sa_i}(s_{a_2}, p, sa_2, p_2, x_2^2), \ldots, Dep_{sa_i}(s_{a_m}, p, sa_m, p_m, x_m^m)\} \) satisfying \( GetGoalInputs(p) \subseteq (x_1^1 \cup x_1^2 \cup x_1^3 \cup \ldots \cup x_m^m) \), then \( Ds \) is called the dependence solution of plan \( p \). For any element \( e \in Ds \), if \( Ds - e \) is not the dependence solution, then \( Ds \) is called the minimal cover dependence solution of the plan \( p \).

The number of elements contained in the minimal cover dependence solution is called the length of the minimal cover dependence solution. All the minimal cover dependence solutions of the plan are alternative solutions for fully matching the input parameters of the plan.

Once the user defines the service requirement, a user agent denoted as UserAgent that acts on behalf of the user is created. The key to DPAWSC is the alternative solution with smaller length has higher priority to be searched than one with larger length for both the user agent and the service agent. DPAWSC is based on the reasonable assumption that the plan workflow containing a smaller number of plans is better than one having more plans. This is because fewer plans in the plan workflow mean fewer participants in the composite Web service that undertakes to achieve the service requirement. Fewer participants also mean less interoperability and collaboration among service agents as well as the smaller management.

In order to find a feasible solution, both the user agent and the service agents always choose the alternative solution with the minimum length to search first. If the currently searching alternative solution is feasible, then the search is ended, otherwise, the next alternative with minimum length from the unsearched alternatives is chosen to continue the search until a feasible solution is found or all alternative solutions have been searched and no feasible solution is found.

Four types of messages are used in DPAWSC: CFP, PROPOSE, REQUEST and CONFIRM. These four types of messages are defined in table 2.

During the process of distributed Web service composition, the plan of a service agent can be one of three statuses: "Unexplored", "Exploring" and "Explored". The status of "Unexplored" means that the plan has not been searched, the status of "Exploring" means that the plan is being searched and the status of "Explored" means that the search for the plan has finished.

For a certain plan, if one of the minimal cover dependence solutions is feasible or its input parameters can be fully matched by the input parameters of the service requirement, then the plan is feasible for its
input parameters can be satisfied, otherwise, the plan is unfeasible.

DPAWSC includes three steps: initialization, backtrack search and forward decision. Fig. 4 shows the activity diagram of DPAWSC.

**TABLE 2**

<table>
<thead>
<tr>
<th>Message</th>
<th>Representation Forms</th>
<th>Sender</th>
<th>Receiver</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFP</td>
<td>CFP(UserAgent, r\textsuperscript{i}, r\textsuperscript{o})</td>
<td>UserAgent</td>
<td>All service agents</td>
<td>UserAgent notifies the service agents that a new task arrives.</td>
</tr>
<tr>
<td>PROPOSE</td>
<td>PROPOSE(sa\textsubscript{i}, p, x)</td>
<td>sa\textsubscript{i}</td>
<td>UserAgent</td>
<td>Service agent sa\textsubscript{i} can provide the parameters x for the UserAgent by the output parameters of plan p.</td>
</tr>
<tr>
<td>REQUEST</td>
<td>REQUEST(UserAgent, null, sa\textsubscript{i}, p, x)</td>
<td>UserAgent</td>
<td>sa\textsubscript{i}</td>
<td>UserAgent requires the service agent sa\textsubscript{i} to provide the parameters x by the output parameter of plan p.</td>
</tr>
<tr>
<td>REQUEST</td>
<td>REQUEST(sa\textsubscript{j}, p\textsuperscript{'} , sa\textsubscript{i}, p, x)</td>
<td>sa\textsubscript{j}</td>
<td>sa\textsubscript{i}</td>
<td>Service agent sa\textsubscript{j} requires the service agent sa\textsubscript{i} to provide the input parameters x for the plan p\textsuperscript{'} of service agent sa\textsubscript{j} by the output parameters of plan p.</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>CONFIRM(sa\textsubscript{i}, p, UserAgent, null, x, true/false, pw)</td>
<td>sa\textsubscript{i}</td>
<td>UserAgent</td>
<td>Service agent sa\textsubscript{i} can/can not provide the parameters x for the plan p\textsuperscript{'} of service agent sa\textsubscript{j} by the output parameters of plan workflow pw containing the plan p.</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>CONFIRM(sa\textsubscript{i}, p, sa\textsubscript{j}, p\textsuperscript{'} , x, true/false, pw)</td>
<td>sa\textsubscript{i}</td>
<td>sa\textsubscript{j}</td>
<td>Service agent sa\textsubscript{j} can/can not provide the inputs parameters x for the plan p\textsuperscript{'} of service agent sa\textsubscript{j} by the output parameters of plan workflow pw containing the plan p.</td>
</tr>
</tbody>
</table>

Fig. 4. Activity diagram of DPAWSC

### 5.2 DPAWSC Steps

#### 5.2.1 Initialization

Once the user submits the service requirement, a user agent on behalf of the user is created and it sends the message CFP(UserAgent, r\textsuperscript{i}, r\textsuperscript{o}) to all service agents to notify the service agents that a new task arrives.

When the service agent sa\textsubscript{i} receives the message CFP(UserAgent, r\textsuperscript{i}, r\textsuperscript{o}), algorithm 1 is executed.

**Algorithm 1**: when service agent sa\textsubscript{i} receives message CFP(UserAgent, r\textsuperscript{i}, r\textsuperscript{o})

1. **foreach** plan p \( \in \text{GetPlans}(sa\textsubscript{i}) \) **do**
2. set \( p.\text{status} = "Unexplored"; \)
3. if \( \text{GetGoalOutputs}(p) \cap r\textsuperscript{o} \neq \phi \) then
4. send PROPOSE(sa\textsubscript{i}, p, GetGoalOutputs(p) \cap r\textsuperscript{o}) to UserAgent;
5. end
6. end

Where the function of GetPlans(sa\textsubscript{i}) returns the set of plans that the service agent sa\textsubscript{i} has. The service agent sa\textsubscript{i} checks whether there is a plan p whose output parameters can help the user agent to achieve the service requirement (Algorithm 1 line 3). If it is true, it sends the message PROPOSE(sa\textsubscript{i}, p, GetGoalOutputs(p) \cap r\textsuperscript{o}) to UserAgent.

#### 5.2.2 Backtrack Search

When the user agent receives all PROPOSE messages, it creates the set of minimal cover solutions denoted as MS and algorithm 2 is executed.

**Algorithm 2:**

1. if \( MS = \phi \) then
2. notify the user that requirement can not be achieved;
3. else
4. choose the minimal cover solution \( S \) with the minimum length from MS;
5. \( MS = MS - S \);
6. **foreach** \((sa\textsubscript{i}, p, x) \in S\) **do**
7. send REQUEST(UserAgent, null, sa\textsubscript{i}, p, x) to sa\textsubscript{j};
8. \( Sr = Sr \cup REQUEST(UserAgent, null, sa\textsubscript{j}, p, x) \);
9. end
10. end

The user agent checks whether the set of the minimal cover solutions is empty. If it is true, then notify the user that the requirement can not be achieved, otherwise, a minimal cover solution with the minimum length is chosen to search by sending REQUEST messages (Algorithm 2 line 6-9).

Once the service agent sa\textsubscript{i} receives the message REQUEST(s, p\textsuperscript{'} , sa\textsubscript{i}, p, x), algorithm 3 is executed.

\( Rr\textsubscript{p} \) is the set of REQUEST messages about the plan p that the service agent has received, \( Sr\textsubscript{p} \) is the set of REQUEST messages about the plan p that the service agent has sent and \( DSS\textsubscript{p} \) is the set of minimal cover dependence solutions of the plan p for the service requirement. According to the status of the plan p, two cases are distinguished:

1. The status of "Unexplored" (Algorithm 3 line 2), which means it is the first time that the service agent receives the REQUEST message about the plan p and the search for the plan has not been carried out before.
2. The status of "Explored" (Algorithm 3 line 18), which means that the search for the plan p has finished.
Algorithm 3: when the service agent $s_a_i$ receives message REQUEST($s_i, p_i, s_i, p, x$).

1. $Ra_p = Ra_p \cup REQUEST(s_i, p_i, s_i, p, x)$;
2. if $p.status = "Unexplored"$ then
   3. if $GetGoalInputs(p) \subseteq r^i$ and $DSS_p = \emptyset$ then
      4. set $p.status = "Explored"$; set $p.feasible = false$;
      5. send $CONFIRM(s_a_i, p, s_i, p, x, false, null)$ to $s$;
   6. else if $GetGoalInputs(p) \subseteq r^i$ then
      7. set $p.status = "Explored"$; set $p.feasible = true$;
      8. send $CONFIRM(s_a_i, p, s_i, p, x, true, pw)$ to $s$;
   9. else
      10. set $p.status = "Exploring"$;
      11. choose a minimal cover dependence solution $Ds$ with the minimum length from $DSS_p$;
      12. $DSS_p = DSS_p - Ds$;
      13. foreach $Dep_i(s_a_i, p, s_i, p, y) \in Ds$ do
         14. send $REQUEST(s_a_i, p, s_i, p, y)$ to $s_a_i$;
         15. $Sr_p = Sr_p \cup REQUEST(s_a_i, p, s_i, p, y)$;
      16. end
   17. else if $p.status = "Explored"$ then
      18. if $p.feasible = true$ then
         19. send $CONFIRM(s_a_i, p, s_i, p, x, true, pw')$ to $s$;
      20. else if $p.feasible = false$ then
         21. send $CONFIRM(s_a_i, p, s_i, p, x, false, null)$ to $s$;
      22. end
   23. end
24. end

If the status of the plan $p$ is "Unexplored", three conditions are considered:

1. The input parameters of plan $p$ can not be fully matched by $r^i$ and the set of minimal cover dependence solutions is empty (Algorithm 3 line 3), which means the plan $p$ is unfeasible for its input parameters can not be satisfied. Therefore, the search for the plan $p$ is ended and the message $CONFIRM(s_a_i, p, s_i, p, x, false, null)$ is sent to $s$.

2. The input parameters of the plan $p$ can be fully matched by $r^i$ (Algorithm 3 line 6). As the input parameters of the plan $p$ can be directly satisfied by $r^i$, the search for the plan $p$ is ended and the message $CONFIRM(s_a_i, p, s_i, p, x, true, pw)$ is sent to $s$.

3. The input parameters of the plan $p$ can not be fully matched by $r^i$ and the set of minimal cover dependence solutions is not empty. Set the status of the plan $p$ as "Exploring" and a minimal cover dependence solution with minimum length is chosen to search by sending REQUEST messages (Algorithm 3 line 10-16).

If the status of the plan $p$ is "Explored" (Algorithm 3 line 18), which means the message REQUEST($s_i, p_i, s_i, p, x$) is received after the search for the plan $p$ has finished, then the CONFIRM message is directly sent to $s$ according to whether the plan is feasible or unfeasible (Algorithm 3 line 19-23).

5.2.3 Forward Decision

Once the service agent $s_a_j$ receives the message CONFIRM($s_a_j, p''$, $s_j, p, x, flag, pw$), algorithm 4 is executed.

The service agent checks whether the size of $Ra_p$ is equal to the size of $Sr_p$ (Algorithm 4 line 2), where $Ra_p$ is the set of CONFIRM messages about the plan $p$ that service agent has received. If it is true, it means all the CONFIRM messages of currently searching minimal cover dependence solution of the plan $p$ have been received. Therefore, it is the time to determine whether the currently searching alternative solution is feasible or unfeasible. If all flag bits of the CONFIRM messages about the currently searching alternative solution in $Ra_p$ are true, it denotes that the currently searching alternative solution is feasible.

If the currently searching alternative solution is feasible, the search for the plan $p$ is ended and CONFIRM messages are sent to respond the received REQUEST messages about the plan $p$ (Algorithm 4 line 5-7).

If the currently searching alternative solution is not feasible, then the next minimal cover dependence solution with the minimum length is chosen from the unsearched alternatives to continue the search (Algorithm 4 line 9-16).

If all alternative solutions of the plan $p$ have been searched and no feasible solution is found, then the search for the plan $p$ is ended and CONFIRM messages are sent to respond the received REQUEST messages about the plan $p$ (Algorithm 4 line 19-21).

When the user agent receives the message CONFIRM($s_a_j, p''$, UserAgent, null, flag, pw), algorithm 5 is executed, which has similar logics with algorithm 4.
Algorithm 5: When the user agent receives the message CONFIRM(sa_i, p'^i', UserAgent, null, flag, pw)

1. Ra = Ra ∪ CONFIRM(sa_i, p'^i', UserAgent, null, flag, pw);
2. if |Ra| = |Sr| then
   3. if the currently searching minimal cover solution is feasible then
      4. notify the user that the solution has been found;
   5. else if MS ≠ Ø then
      6. choose the minimal cover solution S with the minimum length from MS;
      7. MS = MS - S;
      8. foreach (sa_j, p, y) ∈ S do
         9. if REQUEST(UserAgent, null, sa_j, p, y) ∉ Sr then
            10. send REQUEST(UserAgent, null, sa_j, p, y) to sa_j;
            11. Sr = Sr ∪ REQUEST(UserAgent, null, sa_j, p, y);
      12. end
      13. end
   14. else
      15. notify the user that service requirement can not be achieved;
   16. end
17. end

5.3 DPAWSC Complexity Analysis

The number of messages needed is used to evaluate the time complexity of DPAWSC. For a dependence graph with n service agents and r relation dependencies, the number of messages needed can be estimated by the following steps:

1. The number of CFP message is n. Generally, since there are a few plans whose output parameters can partially or fully match the output parameters of the service requirement, the number of PROPOSE messages is far less than that of the CFP messages. Therefore, the number of the messages needed in the initialization stage is about n.

2. In the worst case, all the alternative solutions will be searched. Therefore, the number of REQUEST messages is r and the number of corresponding CONFIRM messages is r.

From the above analysis, the estimation of the messages number is about n + 2r in the worst case.

The service agent community is an open multi-agent system where service agent may leave or enter dynamically. As a consequence, as the set of service agents belonging to the agent community can not be known a priori, the dependence relations must be established dynamically. Whenever a new service agent enters the agent community, he must present himself to the others by sending them the information about its abilities for future cooperation. Meanwhile, the others should send him the information about their abilities as well. In a similar way, the service agents must tell others when they are leaving the community.

6 EXPERIMENTAL EVALUATION

A group of experiments have been designed and performed to evaluate the performance of DPAWSC. The simulation experiments are executed on JADE platform [7]. Firstly, the experiment setting is introduced.

It is assumed that each service agent owns only one plan. Four parameters are used to define the dependence graph, which are defined in table 3.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>number of plans</td>
</tr>
<tr>
<td>d</td>
<td>diameter of the dependency graph</td>
</tr>
<tr>
<td>k</td>
<td>number of dependence relations of plan</td>
</tr>
<tr>
<td>η</td>
<td>number of minimal cover dependence solutions of plan</td>
</tr>
</tbody>
</table>

Given n, d, k, η, the dependency graph is generated according to the following steps.

Step 1: All service agents are divided into d groups denoted as Sa_1, Sa_2, . . . , Sa_d.

The number of service agents in Sa_i (1 ≤ i ≤ d) is equal to [n/d] and the number of the service agents in Sa_d is equal to n − ∑_{i=1}^{d-1}|Sa_i|, where [n/d] is the maximal integer less than or equal to n/d.

Step 2: Construct the dependence relation.

The number of dependence relations of plans in Sa_i (2 ≤ i ≤ d) is denoted as k. Except for the first service agents group, the dependence relations of the plans in Sa_i (2 ≤ i ≤ d) are randomly constructed with the plans contained the service agent groups from Sa_1 to Sa_{i-1}.

Step 3: Generate the minimal cover dependence solutions.

The number of minimal cover dependence solutions of plans in Sa_i (2 ≤ i ≤ d) is denoted as η and the set of minimal cover dependence solutions is randomly selected from the subsets of its dependence relations.

Based on the above generation process of dependence graph, finding a feasible plan workflow whose output parameters can fully match the input parameters of a selected plan is set as the desired output of the service requirement and the complexity of the service requirement is characterized by two parameters:

1. λ(0 ≤ λ ≤ 100%): the percentage that the input parameters r^i can fully match the input parameters of the plans in Sa_1.

2. hop count: the service agent group Sa_i that the selected plan belongs to.

The algorithm’s performance is measured in term of three parameters. The first is the number of messages needed, the second is the number of plans contained in the plan workflow and the third is the number of data flows among plans contained in the plan workflow.

DPAWSC is compared with two other alternative searching strategies that might be used in Web service composition:

1. Random walks. an alternative solution is randomly chosen for searching.
(2) Exhaustive. All alternative solutions are searched and the result is the plan workflow with the minimal number of plans.

6.1 Number of Messages

As far as the number of messages is concerned, only the REQUEST and CONFIRM messages are considered for these three search strategies have equal number of CFP messages and PROPOSE messages. For \( n = 150, d = 15 \), the comparison results of messages number are illustrated in Fig. 5.

![Fig. 5. Comparison of messages number](image)

The horizontal axis shows the value of hop count and the vertical axis shows the number of messages. It can be seen from Fig. 5 that the number of messages naturally increases as the value of hop count increases for these three searching strategies. This is because, as the value of hop count increases, finding a feasible solution will search more plans. Since the exhaustive strategy searches all the alternative solutions, the messages number grows far more rapidly than those of the other two strategies. During the process of Web service composition, the alternative solution with smaller length has higher priority to be searched than one with larger length for DPAWSC. Therefore, the growth rate of DPAWSC is less than that of the random walk strategy. According to the above analysis, it can be known that DPAWSC has better scalability.

Fig. 6 shows how messages number varies with \( \lambda \). The horizontal axis shows the value of hop count and the vertical axis shows the number of messages.

![Fig. 6. Messages number variation with \( \lambda \)](image)

It can be seen from Fig. 6 that the number of messages increases as the value of \( \lambda \) decreases from 100% to 80% for both DPAWSC and the random walk strategy. This is because, as the decrease of the percentage that the input parameters \( r \) can fully match the input parameters of the plans in \( Sa^1 \), more alternative solutions will be tried in order to find a feasible solution. However, both the growth rate and absolute growth value of DPAWSC are much less than that of the random walk strategy.

Fig. 7 shows how messages number varies with \( k \). The horizontal axis shows the value of hop count and the vertical axis shows the number of messages.

It can be seen from the Fig. 7 that the number of messages have a slight increase as the value of \( k \) increases from 6 to 8. This is because, as the increase of dependence relations number, the average length of the minimal cover dependence solutions will increase. To be specific, the average length of the minimal cover
dependence solutions increases from about 3 to 4 as the number of dependence relations increases from 6 to 8.

6.2 Number of Plans

For \( n = 150, d = 15 \), the comparison results of plans number are shown in Fig. 8. The horizontal axis shows the value of hop count and the vertical axis shows number of plans.

Since exhaustive strategy searches all the alternative solutions and the result is the plan workflow with the minimal number of plans. Therefore, the result of exhaustive strategy is optimal. It can be known from Fig. 8 that the number of plans contained in the plan workflow found by DPAWSC is quietly close to the optimal results of the exhaustive search strategy and is much less than that of the random walk strategy. Fewer plans in the plan workflow mean fewer participants in the composite Web service that undertakes to achieve the service requirement. Fewer participants also mean less interoperability and collaboration among service agents as well as the simpler management.

For the given \( n, d, k, \lambda \), it can be seen from Fig. 8(a) and Fig. 8(b) that the difference of these three search strategies becomes less evident as the value of \( \eta \) decreases from 6 to 4. This is because, as the the number of minimal cover dependence solutions decreases, the number of feasible plan workflows for the service requirement decreases. When there is only one feasible solution, these three search strategies will have the same results. We can arrive the same conclusion from Fig. 8(c) and Fig. 8(d).

It can be seen from Fig. 8(a) and Fig. 8(c) that the difference of these three search strategies becomes less evident as the number of dependence relations decreases from 8 to 6. This is because, as the number of dependence relations decreases, the difference of length of the minimal cover dependence solutions decreases. When \( k = 8 \), the length of the minimal cover dependence solution varies from 1 to 8. While \( k = 6 \), the length of the minimal cover dependence solution varies from 1 to 6. When the minimal cover dependence solutions
of each plan have the same length, these three search results contains the same number of plans. We can arrive the same conclusion from Fig. 8(b) and Fig. 8(d).

Fig. 9 shows how plans number in plan workflow varies with \( \lambda \).

For the given \( n, d, k, \eta \), It can be seen from Fig. 9 that the number of plans contained in the plan workflow found by DPAWSC keeps relatively stable as the percentage that the input parameters \( r^k \) can fully match the input parameters of the plans in \( S_0 \) decreases from 100% to 80%. There are two reasons for this phenomenon. One is the average length of the minimal cover dependence solutions is static for the given dependence relations number \( k \). The other is the minimal cover dependence solution with less length has higher probability to be feasible when the value of \( \lambda \) is less than 100%.

### 6.3 Number of Data Flows among Plans

For \( n = 150, d = 15 \), the comparison results of data flows number among plans are shown in Fig. 10.

![Fig. 10. Number of Data Flows](image)

Fig. 10. Number of Data Flows

For the given \( n, d, k, \eta, \lambda \), it can be known from Fig. 10 that the number of data flows among plans contained in the plan workflow of DPAWSC is quietly close to the results of the exhaustive search strategy and is much less than that of the random walk strategy. Fewer data flows in the plan workflow mean the fewer communications among service agents and less execution time.

According to the above analysis, it can be concluded that DPAWSC is effective for its ability to produce high quality solution at a low cost of communication.

### 7 Prototype System and A Case Study

The system implementation and a case study are introduced to show how the algorithm presented in this paper works. The prototype system includes a group of service agent modelling tools and a Web portal to help users to create composite Web services.

![Fig. 11. Service agent modelling environment](image)

Fig. 11. Service agent modelling environment

The service agent modelling tools are developed using Eclipse 3.2 on WindowXP and JADE platform serves as the communication middleware. Fig. 11 shows the service agent modelling tool for the prototype system. This tool is flexible and easy to use. It is composed of a navigation tree to show the hierarchical structure of all the entities defined in the service agent model. The plan model tool supports the drag and drop operations, which facilitates to define the data flows and control flows.

![Fig. 12. The outcome of DPAWSC](image)

Fig. 12. The outcome of DPAWSC

The Web portal denoted as Fig. 12 is a Web interface for users, which is developed using JAVA JSP language and deployed with Jakarta Tomcat 5.0. Through the Web portal, the users submit their service requirement. For the example illustrated in section 3. If one student want to attend the conference INFOCOM2009 held in Rio
de Janeiro of Brazil, he defines the service requirement (Table 4) and submit it through the portal.

Once the service requirement is submitted, a user agent on behalf the user is created. According to the principle of DPAWSC, the user agent sends $\text{CFP}$ messages to all service agents:

1. $\text{CFP(}\text{UserAgent, (conferenceName, departureCity), tripArrangement)}$

Receiving the $\text{CFP}$ message, the service agents creates the minimal cover dependence solutions set:

$$DSS_{\text{ReserveTrip}} = \{\text{Dep}_a(\text{TripReservationAgent, ReserveTrip, TripAssistant, PlanTrip, (flightInfo, hotelInfo)}), \{\text{Dep}_a(\text{TripReservationAgent, ReserveTrip, FlightConsultant, GetFlight, flightInfo), Dep}_a(\text{TripReservationAgent, ReserveTrip, HotelConsultant, GetHotel, hotelInfo})\}$$

$$DSS_{\text{TripAssistant}} = \phi$$

$$DSS_{\text{GetFlight}} = \{\text{Dep}_a(\text{FlightConsultant, GetFlight, ConferenceAgent, GetConferenceInfo, (conferenceDate, conferenceAddress)})\}$$

$$DSS_{\text{ConferenceAgent}} = \phi$$

Since the plan ReserveTrip of TripReservationAgent can provide the outputs parameters for the user agent, TripReservationAgent sends the following message to user agent:

2. $\text{PROPOSE(}\text{TripReservationAgent, ReserveTrip, tripArrangement)}$

According to the $\text{PROPOSE}$ messages, the user agent creates the minimal cover solutions set:

$$MS = \{\text{[(TripReservationAgent, ReserveTrip, tripArrangement)]}\}$$

In the backtrack search stage, the user agent choose the minimal cover solution with minimum length to search by sending $\text{REQUEST}$ message.

(3) $\text{REQUEST(}\text{UserAgent, null, TripReservationAgent, ReserveTrip, tripArrangement)}$

Since the plan ReserveTrip of TripReservationAgent has two minimal cover dependence solutions. The first is chosen to search and following message is sent to TripAssistant:

(4) $\text{REQUEST(}\text{TripReservationAgent, ReserveTrip, TripAssistant, PlanTrip, (flightInfo, hotelInfo)})$

Since the minimal cover dependence solutions set of plan PlanTrip is null and its input parameters can not be fully matched by the input parameters of the service requirement. Therefore, the following message is sent to TripReservationAgent:

(5) $\text{CONFIRM(}\text{TripAssistant, PlanTrip, TripReservationAgent, ReserveTrip, (flightInfo, hotelInfo), false, null)}$

Since the current searching minimal dependence cover solution of plan ReserveTrip is not feasible, the next alternative is chosen to search and following messages are sent.

(6) $\text{REQUEST(}\text{TripReservationAgent, ReserveTrip, FlightConsultant, GetFlight, flightInfo)}$

(7) $\text{REQUEST(}\text{TripReservationAgent, ReserveTrip, HotelConsultant, GetHotel, hotelInfo)}$

Similarity, following messages are sent.

(8) $\text{REQUEST(}\text{FlightConsultant, GetFlight, ConferenceAgent, GetConferenceInfo, (conferenceDate, conferenceAddress)})$

(9) $\text{REQUEST(}\text{HotelConsultant, GetHotel, ConferenceAgent, GetConferenceInfo, (conferenceDate, conferenceAddress)})$

(10) $\text{CONFIRM(}\text{ConferenceAgent, GetConferenceInfo, FlightConsultant, GetFlight, (conferenceDate, conferenceAddress), true, (GetConferenceInfo)})$

(11) $\text{CONFIRM(}\text{ConferenceAgent, GetConferenceInfo, HotelConsultant, GetHotel, (conferenceDate, conferenceAddress), true, (GetConferenceInfo)})$

(12) $\text{CONFIRM(}\text{FlightConsultant, GetFlight, TripReservationAgent, ReserveTrip, flightInfo, true, (GetConferenceInfo, GetFlight)})$

(13) $\text{CONFIRM(}\text{HotelConsultant, GetHotel, TripReservationAgent, ReserveTrip, hotelInfo, true, (GetConferenceInfo, GetHotel)})$

(14) $\text{CONFIRM(}\text{TripReservationAgent, ReserveTrip, UserAgent, null, TripArrangement, true, (GetConferenceInfo, GetFlight, GetHotel, ReserveTrip)})$

The sniffer agent can be used to track the messages exchanged among service agent. As the name itself points out, the sniffer agent allows to track messages exchanged in a JADE agent platform. When the user decides to sniff an agent, or a group of agents, every message directed to or coming from that agent, or group of, is tracked and displayed in the sniffer window.

8 Related Work

Web service composition has received a lot of attention. Most of the work about the Web service composition can be broadly classified into three categories: manual composition, semi-automatic and automatic composition.
In manual composition, the composite service is modelled manually by a structure of sub-services using a service flow language such as BPEL4WS [37]. Examples of this approach can be found in [2][8][20][23][26]. The manual Web service composition requires the user to have the domain knowledge. Meanwhile, it is a labor-intensive and error-prone task, thus it is not appropriate for the large scale Web service composition.

Some semi-automatic composition techniques have been proposed [14][17][27]. Although, these semi-automatic composition techniques solve some of the problems of manual composition, they are still unsuitable as the filtering process may provide numerous services for the user to select from.

Many research efforts on automatic Web service composition via AI planning have been reported [4][15][32]. These automatic Web service composition approaches take the input parameters and the desired output parameters as the initial state and goal state respectively, and the available Web services are the set of actions that the planner can perform in attempting to change one state to another state [16]. However, these centralized Web service composition approaches suffer from performance bottlenecks and single points of failure.

The topic of agent-based service composition has recently received great attention. McIlraith and son [30] proposed an agent-based Web service composition framework. Wang et al. [35] proposed an agent-based Web service workflow model for inter-enterprise collaboration. Maamar et al. [41] presented an agent-based and context-oriented approach that supports the composition of Web Services. During service composition process, software agents engage in conversations with their peers to agree on the Web services that participate in this process. However, the agents in the above mentioned approaches are not used to achieve the automatic Web service composition but to the coordination and enactment of the composite Web services.

The approach of service level agreements for the service composition through agent negotiation is presented in [19]. An agent-enabled semantic Web service composition approach is proposed by Ermolayev et al. [40], where the middle agent layer is introduced to conduct service requirement to task transformation, agent-enabled cooperative task decomposition and performance. However, integration of Web service technologies and software agent technologies is not achieved.

The motivation of agent-based coalition formation for service composition [11] is in a sense similar to our approach, which focuses on repetitive coalition formation for service composition according to private preferences. However, the coalition algorithm has low scalability. Our approach differs from existing approaches in two aspects: (1) A formal service agent model is proposed, which integrates the software agent technologies and Web service technologies into a cohesive entity. (2) A distributed algorithm for automatic Web service composition is proposed, which supports the distributed nature of Web service composition.

9 Conclusion and Future Work

Aiming to exploit agent’s capabilities to enhance Web service’s behaviors, a service agent model is proposed in this paper. Based on the service agent model, a distribute planning algorithm for Web service composition named as DPAWS is proposed. The simulation experiments demonstrate the effectiveness of DPAWS.

Our ongoing work includes distributed quality optimization for composite Web service and exception handling mechanism for the distributed execution of the composite Web service as well as information exchange mechanisms among service agents.

Acknowledgment

This work is supported by National Science Foundation of China under grant 60873230, National High-Tech Research and Development Plan of China under grant 2007AA01Z137 and 2006AA01A124, Major Program for the Fundamental Research of Shanghai Committee of Science and Technology under grant 08JC1411700, Program for New Century Excellent Talents in University under grand NCET-08-0347, National Basic Research Program of China under grant 2007CB310702.

References


Hongxia Tong received his Ph.D degree from Shanghai Jiao Tong University in 2009. He currently works as a post-doctor at the Institute of Computing Technology, Chinese Academy of Sciences. His main research topics include service computing, multi-agent system and network measurement.

Jian Cao received his Ph.D degree from Nanjing University of Science and Technology in 2000. He is currently a professor of the Department of Computer Science and Engineering at Shanghai Jiaotong University. His main research topics include service computing, cooperative information system and software engineering. He has published more than fifty papers.

Shensheng Zhang received his Ph.D degree from Stanford University in 1988. He is currently a professor of the Department of Computer Science and Engineering at Shanghai Jiaotong University. His main research topics include distributed computing, agent, virtual reality and software engineering.

Minglu Li received his Ph.D degree from Shanghai Jiaotong University in 1996. He is currently a professor of Department of Computer Science and Engineering at Shanghai Jiaotong University. His main research topics include grid computing, image processing, and e-commerce. He is the Director of IBM-SJTU Grid Research Center at Shanghai Jiaotong University.