Knowledge-Based Systems 23 (2010) 132–143

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

# Agent service matchmaking algorithm for autonomic element with semantic and QoS constraints

Zhang Kun [a,b,*], Xu Manwu [b], Zhang Hong [a], Xu Jian [a]

[a] School of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing 210094, China
[b] State Key Laboratory for Novel Software, Nanjing University, Nanjing 210093, China

ARTICLE INFO

*Article history:*
Received 17 October 2008
Accepted 10 July 2009
Available online 30 October 2009

*Keywords:*
Agent service matchmaking
Service description model
Semantic similar
QoS constraints
Autonomic element
Autonomic computing

ABSTRACT

Agent service description and matchmaking problem for autonomic element has been taken as one of the most important issue in the field of autonomic computing based on agent and multi-agent system. Considering the semantic and QoS for capability description of agent service are two important issues during matchmaking, the factors of semantic and QoS during matchmaking are considered together, and an agent service description model named ***ASDM_SQ*** is proposed. On basis of this model, a matchmaking algorithm with semantic and QoS constraints named ***ASMA_SQ*** is presented to find the agent service satisfied both the given semantic similarity threshold and optimal QoS performance. The proposed algorithm lies over two fundamental processes: semantic similarity matchmaking and QoS matchmaking. During QoS matchmaking, evaluation mechanism of confidence of individual QoS attributes, i.e. fidelity factor is introduced to overcome drawbacks such as subjectiveness and unfairness and improve the self-configuration capability for agent element. Simulation experiments demonstrate the effective and correction of our algorithm for agent service matchmaking, and perform better QoS performance than other existing algorithm, which can further show that our algorithm has better compromise between attribute quality and users' evaluation when selecting agent service.



## 1. Introduction

Advances in networking and computing technology and software tools have resulted in an explosive growth in networked applications and information services that cover all aspects of our life. These sophisticated applications and services are extremely complex, heterogeneous and dynamic. As a result, this complexity has increased the cost and errors of managing IT infrastructures. The skilled persons who manage these systems are expensive and cannot manage them in configuration, healing, optimization, protection, and maintenance. Moreover, IT managers look for ways to improve the return on investment (ROI) by reducing total cost of ownership (TCO), improving quality-of-services (QoS), and reducing the cost for managing of IT complexity. All these issues have motivated researchers to investigate a new idea to cope with the management of complexity in IT industry and self-management systems have been introduced. In mid-October 2001, IBM released a manifesto observing that the main obstacle to further progress in the IT industry is a looming software complexity crisis. Aimed at this problem, IBM Company innovatively proposed autonomic computing (AC) technology – computing systems that can manage themselves given high-level objectives from administrators. The goal of autonomic computing initiative is to help customers build more automated IT infrastructures to reduce costs, improve uptime, and make the most efficient use of increasingly scarce support skills. The essence of autonomic computing systems is self-management, composed of self-configuration, self-optimization, self-healing, and self-protection [1,2].

With the advent of agent and multi-agent system (MAS) technologies, autonomic computing based on agent has become research hotspot [3,4]. In these researches, authors aimed to achieve self-management of a distributed computing system via interactions amongst a population of autonomous agents called autonomic elements, i.e., the agent was regarded as autonomic element. Those researchers focused on how to utilize agent and multi-agent system to provide variously effective and efficient services for users, in order to make autonomic computing realize self-management capability by self-configuration. The most important issue in this field is description and matchmaking of agent services (i.e., service description and service matchmaking), that is, how to achieve the agent service matchmaking for autonomic element in self-configuration.

Many algorithms have been proposed for agent or agent service description and matchmaking [5–14]. However, there are three

* Corresponding author. Address: State Key Laboratory for Novel Software, Nanjing University, Nanjing 210093, PR China. Tel.: +86 25 84315982; fax: +86 25 84315960.
*E-mail address:* zhangkun@mail.njust.edu.cn (Z. Kun).

0950-7051/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2009.07.011

major difficulties in deployment and application of the existing algorithms to matchmaking of agent services.

Firstly, most of the existing algorithms are only concerned with functional matchmaking on agent service, and are used syntax or semantic matchmaking to obtain agent services satisfying with functional demands. Whereas, few of these algorithms focused on non-functional factors, such as service cost, time, reliability, satisfaction, i.e. quality-of-service (QoS) of agent service. Therefore, when more agents than one can provide a functional service, these algorithms just selected randomly one from these agents. The agent service requesters could not obtain optimal agent service with high QoS performance.

Secondly, only a few existing algorithms are considered QoS factors during agent service matchmaking. However, these algorithms were paid regard to QoS level as a whole, ignoring confidence of individual QoS attributes. Due to QoS data was often advertised by agent service providers, it suffered from the drawbacks such as subjectiveness and unfairness. Further, they took the attribute matchmaking level of agent service itself as the only target for measuring effect of agent service matchmaking, and did not consider the user's feeling and satisfaction. As a result, such subjectiveness and unfairness may restrain improvement of self-configuration capability for agent element to a certain extent.

The final difficulty is that, in recent years, semantic-based agent service matchmaking has become the focal point of agent service due to improvement of recall ratio and precision ratio in the matchmaking method based on syntax. Several semantic-based agent matchmaking algorithms were proposed. To the best of our knowledge, little work has been done on efficient combining with semantic matchmaking and QoS matchmaking. These authors focused on either semantic functional matchmaking [6,7,9,12], or QoS matchmaking [13,14]. As we know, the semantic and QoS for capability description of agent service are two important issues during matchmaking.

Aimed at overcoming the above difficulties, we consider together the matchmaking influenced by semantic and quality-of-service in this paper, and propose an agent service description model (***ASDM_SQ***) for autonomic element. On basis of this model, we put forward to a matchmaking algorithm named ***ASMA_SQ*** consisting of two phases, i.e. semantic matchmaking phase and QoS matchmaking phase, in order to find the agent service satisfying both the given semantic similarity threshold and optimal QoS performance. The fidelity factor for each QoS attribute is introduced into agent service matchmaking algorithm, which will improve the facility and fairness concerned with QoS attributes, and select optimal agent service with QoS performance objectively.

The rest of the paper is organized as follows. Section 2 briefly introduces concept of autonomic computing based on agent. Some of the existing work that is related to this paper is described in Section 3. Section 4 presents agent service description model named ***ASDM_SQ***. Section 5 gives our proposed agent service matchmaking algorithm in detail. Simulation results are presented in Section 6. Section 7 concludes the paper.

## 2. Autonomic computing based on agent

The autonomic computing has been inspired by the human autonomic nervous system. Its overarching goal is to realize computer and software systems and applications that can manage themselves in accordance with high-level guidance from humans. Autonomic systems will be interactive collections of autonomic elements (AEs) – individual system constituents that contain resources and deliver services to humans and other autonomic elements. Autonomic elements will manage their internal behavior and their relationships with other autonomic elements in accordance with policies that humans or other elements have established. So, we can know that autonomic elements are the basic building blocks of autonomic systems and their interactions produce self-managing behavior.

Autonomic elements will have complex life cycles, continually carrying on multiple threads of activity, and continually sensing and responding to the environment in which they are situated. Autonomy, proactivity, and goal-directed interactivity with their environment are distinguishing characteristics of software agents. Viewing autonomic elements as agents (agent services) and autonomic systems as multi-agent systems makes it clear that agent-oriented architectural concepts will be critically important [2,15].

On the basis of above assertion, we give the autonomic computing system architecture based on agent (or agent service), shown in Fig. 1.

The whole autonomic computing system (ACS) based on agent consists of three kinds of autonomic elements, i.e., agent service provider, agent service requester, and agent service broker (or middle agent service). These agent services communicate and cooperate with each other in order to implement the ACS's capability, which can show the characteristic of self-management for ACS. Agent service providers describe their capabilities, and publish these advertisements to requesters or brokers. Agent service requesters try to find their required agent service provider(s) according to their requests and published advertisements. Finally, a specific kind of AE is agent service broker (or called middle agent service). They support to cooperation between agent service providers and requesters, such as agent service negotiation, agent service composition, agent service collaboration, and agent service monitoring and so on.

Based on the above architecture, in order to efficiently accomplish the self-configuration and self-management between autonomic element agent services, the most important issue is how to describe agent services, and design efficient agent service matchmaking algorithm to find appropriate provider(s) for a requester according to the providers' agent service advertising description.

## 3. Related work of matchmaking in agents

In recent years, much research has been devoted to the development of service matchmaking algorithms for agent or autonomic element [5–14]. Wickler [5] addressed the problem of capability brokering agent, and proposed a new capability description language (CDL) for the representation of agent capabilities. CDL was a decoupled action representation into which arbitrary state representations can be plugged, resulting in the expressiveness and flexibility needed for capability brokering. However, description for CDL was based on syntax rather than semantic. Sycara [6] defined a language called LARKS for agent advertisements and requests, and presented a flexible and efficient matchmaking process that used LARKS. LARKS performed both syntactic and semantic matching, and in addition allowed the specification of concepts (local ontologies) via ITL, a concept language. The establishment for semantic distance consumed much workload, so that their matching algorithm had limitations on practicability and reliability. Arisha et al. [9] brought forward to a simple agent service description language named SDL, and provided approximate service matchmaking by using semantic distance. Whereas, the algorithm could not support definition of data type and descript services efficiently. The authors in Refs. [11] and [12] focused on agent service matchmaking based on semantic. Shi et al [11] presented a description logic-based service matchmaking algorithm for multi-agent systems by using model-theoretic semantics and concept hierarchy of description logic. The authors proposed five kinds of

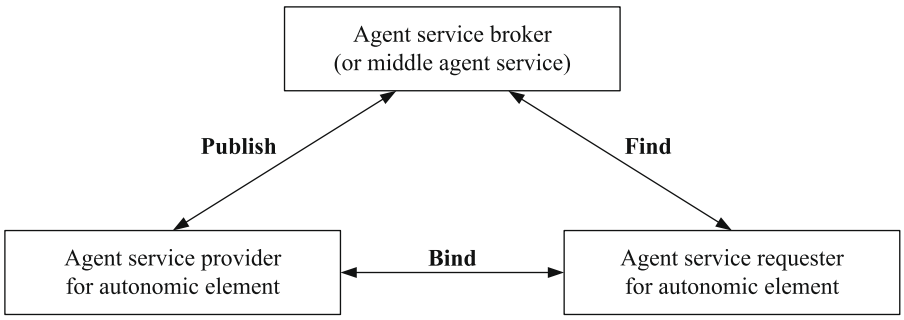


**Fig. 1.** Autonomic computing system architecture based on agent.

algorithms of agent service matchmaking, i.e., agent service matchmaking algorithm, strong service matchmaking algorithm, *k*-nearest service matchmaking algorithm, *k*-approximate service matchmaking algorithm, *m*-nearest and *n*-approximate service matchmaking algorithm. The proposed mechanism of service hierarchy could overcome the shortcoming of the method based on semantic distance. Hu et al. [12] defined domain concept language (DCL) to describe domain concepts and concept-taxonomy, and introduced the service semantic compatible degree to match agent service. The author proposed ontology based on compatible matchmaking mechanism for services request and services provider, which can rapidly find appropriate services provider for services request. From classifying matching to parameter matching filtration, till to the precise constraint matching at last, the matchmaking process had clear hierarchy and well matching efficiency. However, neither of the above algorithms considered the factors of quality-of-service attribution, such as service cost, time, and reliability. Zhang [13] and Jiang [14] analyzed the drawbacks in previous papers, i.e., the matchmaking was only based on the advertised capabilities of provider agents. They considered the matchmaking influenced by QoS of agent service, and individually presented agent service matchmaking algorithm based on QoS. In Ref. [13], it was argued that the practical performance of service provider agents had a significant impact on the matchmaking outcomes of middle agents. The authors' proposed algorithm can pick up the provider agents based on the history information in accomplishing similar tasks in the past rather than choosing randomly. At the launching of an agent system, the proposed algorithm can provide initial values of the track records. With agents' history information and the initial values of the track records, the quality of matchmaking algorithms can be improved significantly, and the returned results was more accurate and reasonable. Jiang [14] pointed out that there were two drawbacks using the track records in Ref. [13], i.e., the value of track records was too subjective, and track record model is too simple to judge agent services' performance more accurately. So, the authors in Ref. [14] presented model of agent service and QoS-driven matchmaking algorithm, which aimed at matching the best satisfying agent for user. It is pity for the above two algorithms that the authors did not considered semantic match issue, which could not represent the semantic correlation among agent services.

## 4. Autonomic element agent service description model

The first task is to realize service capability description during agent service matchmaking. In this paper, we give an agent service description model with semantic and QoS named ***ASDM_SQ*** as follows:

**Definition 1.** Agent service description model for autonomic element is a 3-tuple as follows:

$$AS = (general\_desc, functional\_desc, qos\_desc) \quad (1)$$

where *general_desc* denotes general description of agent service, i.e., elementary attributes, such as agent service name, agent service domain taxonomy, text description and so on. *functional_desc* represents functional description, including interface type and input/output parameters list. *qos_desc* denotes quality-of-service attribute for agent service, divided into two categories: one represents common QoS parameters, which are independent of domain specialty of agent service, such as execution time, execution cost, reliability, maintainability, and satisfaction; the other includes some quality parameters of domain specialty, for example business contents and business context. In general, domain specialty is given by domanial specialist or designer depended on their experience, or obtained through learning from domain ontology. For convenience, we do not consider the domain specialty of autonomic element agent service in this paper.

**Definition 2.** Agent service QoS model for autonomic element is a 6-tuple as follows:

$$qos\_desc = (time, cost, reliability, maintainability, reputation, fidelity) \quad (2)$$

where *time* is agent service response time from sending service request to receiving result, including service process time and service transmission delay time, i.e., $q_{time}(as) = T_{process}(as) + T_{trans}(as)$; *cost* is agent service cost, and represents the expenses paid by users to agent service provider; *reliability* is a metric of reliability, denoting the probability of agent service providing its registered services; *reliability* is a technical measure related to hardware and/or software configuration of agent services and the network connections between the agent service requesters and providers; *maintainability* denotes the probability of accurate maintenance when an exception occurs for an agent service; *reputation* is a measure of agent services' trustworthiness or satisfaction degree, and denotes users' (or requesters') satisfaction to agent service; *reputation* mainly depends on end user's experiences of using the agent service. Different end users may have different opinions or satisfaction degree on the same service. Usually, the end users or requesters are given a range to rank or score agent services, for example, in Amazon.com, the range is [0,5]. In this paper, the range is [0,1].

The first four QoS attributes are often published by agent service providers themselves, and describe the basic QoS performance of an agent service, so most of the authors take them as quality criteria for an agent service. However, such criterion is too subjective to reflect actual quality of agent service. In this paper, we introduce attribute "*reputation*" to measure users' satisfaction degree. On the other hand, *fidelity* vector is added to the QoS model to improve the impartiality and objectivity. Fidelity is treated as a vector composed of fidelity attributes. Each fidelity attribute refers to the confidence or fidelity of above first four QoS attributes, i.e.,

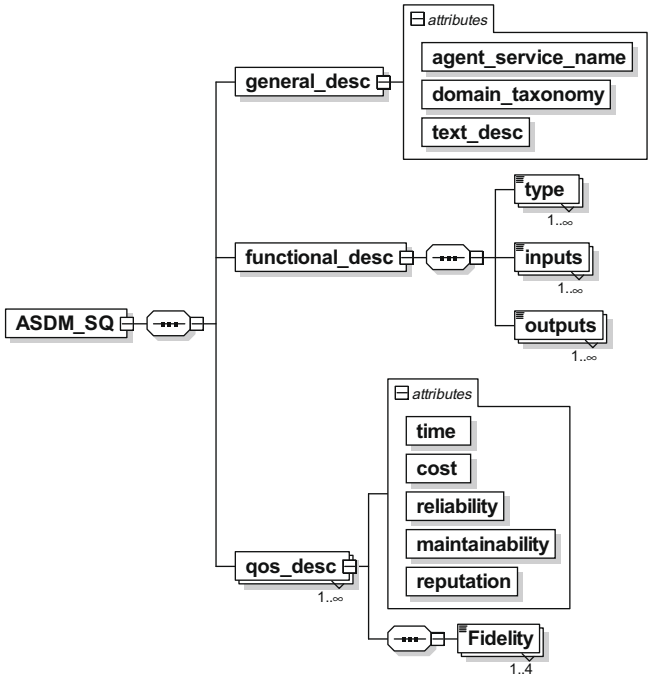


**Fig. 2.** The description schema of ***ASDM_SQ***.

$$fidelity = \langle fid_{time}, fid_{cost}, fid_{reliability}, fid_{maintainability} \rangle \quad (3)$$

The Fig. 2 shows agent service description schema for autonomic element based on ***ASDM_SQ***, followed by XML Schema.

Based on ***ASDM_SQ***, we consider an example for an agent service description of *Booking Airplane Ticket* as shown in Fig. 3.

## 5. Our proposed matchmaking algorithm

### 5.1. Definitions and basic ideas

**Definition 3** (*Agent service matchmaking type*). Given an agent service provider *pas* and an agent service requester *ras*, if *pas* and *ras* are equivalent in terms of semantic, i.e., $ras \equiv pas$, this type of matching is called *exact match*, which is the most restrictive one; if $ras \subseteq pas$, that is, the agent whose capability description matches a given request can be "plugged into the place" where that request was raised, called *plug-in match*. The least accurate but most useful match is the so-called *relaxed match*. A relaxed match has a much weaker semantic interpretation than an exact match and plug-in match, and it determines how close the two descriptions are by returning just a numerical distance value. In this paper, only relaxed match is considered.

**Definition 4** (*Agent service matchmaking problem*). Given the set of providers $PAS = \{pas_1, pas_2, \ldots, pas_n\}$, an agent service requester *ras*, and semantic similarity threshold $\Delta$. Agent service matchmaking problem is to find agent service provider(s) such as the semantic similarity is satisfied $\Delta$ and that the QoS level is maximized.

**Definition 5** (*Semantic similarity of agent services*). Semantic similarity of agent services is the likeness of meaning (or semantic content) of concepts (or terms, or words) appear in agent services' description. The higher semantic similarity between agent services, the higher semantic matchmaking of them.

Aiming at the above problem in definition 4, an agent service matchmaking algorithm for autonomic element with both semantic and QoS constraints named *ASMA_SQ* is proposed in this paper. There are two phases in *ASMA_SQ*. In phase I, we will attempt to get the semantic similarity of general description and functional description for agent services through semantic similarity matchmaking algorithms (in Section 5.2). If the total semantic similarity is greater than or equal to given threshold $\Delta$, then those agent services satisfying condition will be selected as candidates. The second phase is QoS matchmaking phase (in Section 5.3), in which we generate an agent service whose total QoS value is maximal among all candidates. During QoS matchmaking, the fidelity factor is introduced to improve the impartiality and objectivity of quality performance. The pseudo code for *ASMA_SQ* is shown in Fig. 4.

### 5.2. Semantic similarity matchmaking phase

In semantic similarity matchmaking phase, the semantic similarity of general description and functional description for agent services are calculated through the following Algorithms 1 and 2, respectively. Then, the agent services whose total semantic similarity satisfies formula (4) are selected as candidate ones.

$$sim = \xi_1 \cdot sim_g + \xi_2 \cdot sim_f \geqslant \Delta \quad (4)$$

where, *sim* is total semantic similarity of agent services. $sim_g$ and $sim_f$ denote semantic similarity of general description and functional description for agent service, obtained by the Algorithms 1 and 2, respectively. $\xi_1$ and $\xi_2$ are the weight value of general and functional description, respectively, decided by users. In general, $\xi_1 + \xi_2 = 1, 0 < \xi_1, \xi_2 < 1$. $\Delta$ denotes given semantic similarity threshold.

**Algorithm 1** (*Semantic similarity matchmaking algorithm for general description*). The similarity value is calculated through getting concept semantic similarity of each general description in an ontology concept tree concerned with agent service. For convenience, we take ontology concept similarity of agent service name as whole semantic similarity of general description in this paper, and omit other attributes, such as domain taxonomy, text description and so on.

Now, the methods for calculating ontology concept similarity include geometrical distance-based, description logic-based, and information content-based. Considering drawbacks of inaccurate matchmaking in method based on description logic, complexity, and poor universality in method based on information content, we use simple distance-based method [16] to get concept semantic similarity in this paper. The authors in Ref. [16] combined the shortest path length as well as the depth of the subsumer, and proposed the following method to calculate semantic similarity between two concepts $c_1$ and $c_2$, that is,

$$sim(c_1, c_2) = e^{-\alpha \cdot l} \cdot \frac{e^{\beta \cdot h} - e^{-\beta \cdot h}}{e^{\beta \cdot h} + e^{-\beta \cdot h}} \quad (5)$$

where $l$ is the shortest path length between $c_1$ and $c_2$, $h$ is the depth of subsumer in the hierarchy ontology semantic tree, $\alpha \geqslant 0$ and $\beta \geqslant 0$ are parameters scaling the contribution of shortest path length and depth, respectively. Based on their experiments, the optimal parameters for the method are: $\alpha = 0.2$, $\beta = 0.6$.

For example, we consider a tourism service industry ontology as shown in Fig. 5, including several services concerned with Tourism Reception, such as Accommodation Reception, Restaurant Reception, and Ticket Service. According to (5),

*sim*(*Hotel Reservation Service*, *Motel Reservation Service*) = 0.360 ($l = 2$, $h = 1$);
*sim*(*Selling Subway Ticket Service*, *Booking Ticket Service*) = 0.295 ($l = 3$, $h = 1$);

```
<ASDM_SQ>
    <general_desc>
        <agent_service_name = "Booking Airplane Ticket Service">
        <domain_taxonomy = "Tourism Services Industry">
        <text_desc = "Booking airplane ticket from airlines">
    </general_desc>
    <functional_desc>
        <typename = "Date", type = "{year, month, day}">
        <typename = "Money", type = "double">
        <typename = "Ticket", type = "string">
        <input = "Your ID card number", type = "string">
        <input = "Booking date", type = "Date">
        <input = "Your destination", type = "string">
        <input = "Your money", type = "Money">
        <output = "Change money", type = "Money">
        <output = "Your ticket", type = "Ticket">
    </functional_desc>
    <qos_desc>
        <time = "37.56", unite = "ms">
        <cost = "25.00", unite = "$">
        <reliability = "0.875">
        <maintainability = "0.568">
        <reputation = "0.95">
        <fid_time = "0.75", fid_cost = "0.64", fid_reliability = "0.89", fid_maintainability = "0.95">
    </qos_desc>
</ASDM_SQ>
```

**Fig. 3.** An example for an agent service description.

**PROCEDURE ASMA_SQ** ($PAS = \{pas_1, pas_2, \ldots, pas_n\}, ras, \Delta$ )

1. $PAS' = \varnothing$ ; //agent provider candidates satisfying semantic similarity threshold
2. $qos_{max} = 0$; $index = 0$;
3. **for** (each agent service provider $pas_i \in PAS$) {
4. $sim_g = \textbf{simMatch}_{gerneral\_desc}(ras, pas_i)$; //semantic similarity matchmaking of general description
5. $sim_f = \textbf{simMatch}_{functional\_desc}(ras, pas_i)$; // semantic similarity matchmaking of functional description
6. **if** $(\xi_1 \bullet sim_g + \xi_2 \bullet sim_f) \geq \Delta$ **then** //weight value $\xi_1, \xi_2 \in [0,1], \xi_1 + \xi_2 = 1$.
7. $PAS' = PAS' \cup pas_i$ ;
8. }
9. **if** $PAS' \neq \varnothing$ **then** {
10. **for** (each agent service provider $pas'_j \in PAS'$ ) {
11. $qos_{current} = \textbf{qosMatch}(ras, pas'_j)$ //get whole qos value of current agent
12. if $qos_{current} \geq qos_{max}$ then {
13. $qos_{max} = qos_{current}$; $index = j$;
14. }
15. }
16. **return** $pas'_{index}$ ;
17. }
18. **else return FAILED**; //agent service provider satisfying request does not exist

**Fig. 4.** The pseudo code for ***ASMA_SQ***.

*sim*(*Booking Airplane Ticket Service*, *Booking Train Ticket Service*) = 0.559 ($l = 2$, $h = 2$).

**Algorithm 2** (*Semantic similarity matchmaking algorithm for functional description*). We computing semantic similarity of *typename* set, *input* set and *output* set in functional description, respectively. Then the following formula is used to compute the overall semantic similarity $sim_f$ for functional description:

$$sim_f = \lambda_1 \cdot sim_{f\_typename} + \lambda_2 \cdot sim_{f\_input} + \lambda_3 \cdot sim_{f\_output} \quad (6)$$

where $\lambda_1$, $\lambda_2$, $\lambda_3$ denote the weight value of each kind of functional description, and $\lambda_1$, $\lambda_2$, $\lambda_3 \in [0,1]$, $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Considering that functional description *typename*, *input* and *output* are all description sets, we use a semantic similarity method among entity classes from different ontologies [17] to determine similarity of each description set. The matching method in Ref. [17] was based on

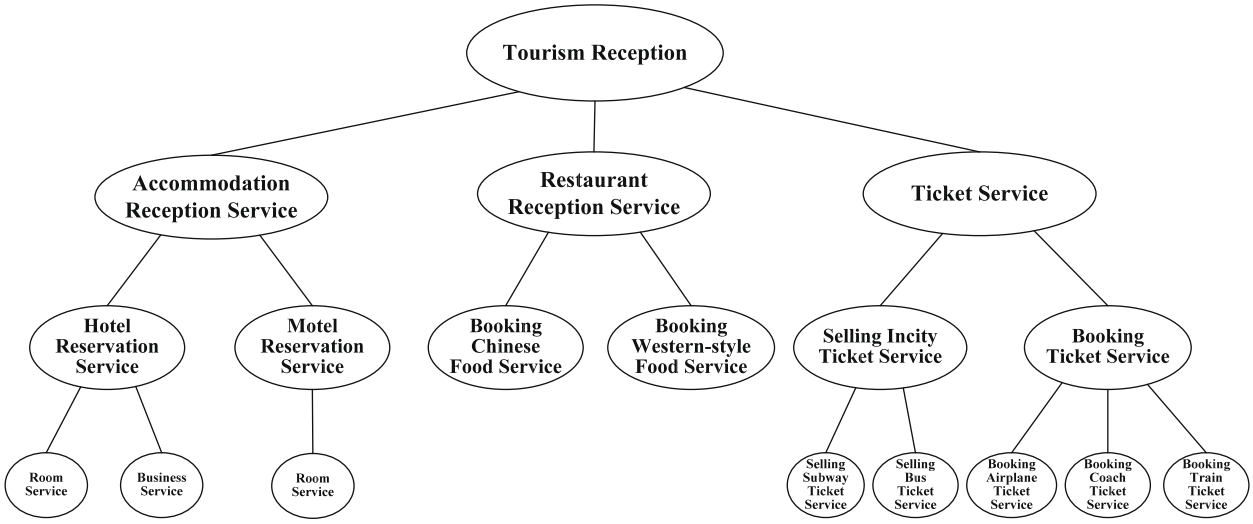


**Fig. 5.** An example of tourism services industry ontology.

the normalization of Tversky's model [18] and set-theory, and was used a process over synonym sets, semantic neighborhoods, and distinguishing features. The similarity model is as follows:

$$sim(a,b) = \frac{|A \cap B|}{|A \cap B| + \delta \cdot |A/B| + (1-\delta) \cdot |B/A|}, \quad \text{for } 0 \leqslant \delta \leqslant 1 \qquad (7)$$

where, $a$ and $b$ are entity classes needed to match (i.e., *typename*, *input* or *output*); $A$ and $B$ corresponds to description sets of $a$ and $b$ (i.e., synonym sets, set of distinguishing features, of set of entity classes in the semantic neighborhood). In this paper, $A$ and $B$ denote specific description items in each functional description set. $|\,|$ is the cardinality of a set; $|A \cap B|$ and $|A/B|$ is intersection and difference of a set, respectively; and $\delta$ is a parameter that defines the relative importance of the non-common characteristics, and is set to 0.45.

For instance, given two agent services $as_1$ and $as_2$, if *input* list set of $as_1$ is {*ID_number*, *Date*, *Destination*, *Money*} and *input* list set of $as_2$ is {*Destination*, *Money*}, then their semantic similarity of input description is:

$$\begin{aligned} sim_{f\_input}(as_1, as_2) &= \frac{\{Destination, Money\}}{\{Destination, Money\} + 0.45 \times \{ID\_number, Date\} + (1-0.45) \times \{\}} \\ &= \frac{2}{2 + 0.45 \times 2 + 0.55 \times 0} = \frac{2}{2.9} = 0.690 \end{aligned} \qquad (8)$$

For case of understanding overall semantic similarity matchmaking phase, we consider an example in which exists following three agent service providers coming from tourism reception ontologies in Fig. 5. They are *Selling Subway Ticket Service*, *Booking Train Ticket Service*, and *Booking Ticket Service*, and their descriptions are shown in Table 1.

Suppose that *Booking Airplane Ticket Service* shown in Fig. 3 is an agent service requester, the semantic similarity calculation result through Algorithms 1 and 2 between requester and the above three agent service provides is shown in Table 2, where $\alpha = 0.2$, $\beta = 0.6$, $\delta = 0.45$, $\lambda_1 = 0.2$, $\lambda_2 = 0.4$, $\lambda_3 = 0.4$, $\xi_1 = 0.5$, $\xi_2 = 0.5$.

It is easy to see from Table 2 that *Booking Ticket Service* has highest similarity (0.815) with agent service requester, and *Selling Sub-*

**Table 1**
Brief *general_desc* and *functional_desc* of three example agent services.

| No. | Agent service name | *general_desc* | *functional_desc* | | |
|---|---|---|---|---|---|
| | | | Typename | Input | Output |
| 1 | *Selling Subway Ticket Service* | *Selling Subway Ticket Service* | Money<br>Ticket | Your destination<br>Your money | Change money<br>Your ticket |
| 2 | *Booking Train Ticket Service* | *Booking Train Ticket Service* | Date<br>Money<br>Ticket | Booking date<br>Your destination<br>Your money | Change money<br>Your ticket |
| 3 | *Booking Ticket Service* | *Booking Ticket Service* | Date<br>Money<br>Ticket | Booking date<br>Your destination<br>Your money | Change money<br>Your ticket |

**Table 2**
The semantic similarity matchmaking result among example agent services.

| No. | $(as_1, as_2)$ | $simMatch_{generation\_desc}$ | | | $simMatch_{functional\_desc}$ | | | | *sim* |
|---|---|---|---|---|---|---|---|---|---|
| | | $l$ | $h$ | $sim_g$ | $sim_{f\_type}$ | $sim_{f\_input}$ | $sim_{f\_output}$ | $sim_f$ | |
| 1 | (*Booking Airplane Ticket Service*, *Selling Subway Ticket Service*) | 4 | 1 | 0.241 | 0.816 | 0.690 | 1.000 | 0.839 | 0.540 |
| 2 | (*Booking Airplane Ticket Service*, *Booking Train Ticket Service*) | 2 | 2 | 0.559 | 1.000 | 0.870 | 1.000 | 0.948 | 0.753 |
| 3 | (*Booking Airplane Ticket Service*, *Booking Ticket Service*) | 1 | 2 | 0.683 | 1.000 | 0.870 | 1.000 | 0.948 | 0.815 |

*way Ticket Service* has lowest similarity (0.540) with agent service requester. This result is correct and reasonable, which can be easily observed from Fig. 5 and Table 1. The example shows the correctness and effectiveness of our proposed semantic similarity matchmaking algorithm.

### 5.3. QoS matchmaking phase

The traditional agent service matchmaking algorithms were often paid attention to functional matchmaking while neglecting non-functional QoS factors. Therefore, when more agents than one can provide a functional service, these algorithms just selected randomly one from these agents. In this paper, we consider QoS attributes matchmaking of agent service after finishing semantic similarity matchmaking, which aims at obtaining optimal agent service with high QoS performance as Refs. [13] and [14]. Moreover, during QoS matchmaking phase, the drawbacks in Refs. [13] and [14] such as subjectiveness and unfairness are overcame through introducing evaluation mechanism of confidence of individual QoS attributes, i.e., fidelity factor for each attribute, which will improve the self-configuration capability for agent element.

The basic idea of QoS matchmaking phase is as follows: firstly, normalizing each attribute in QoS description to range [0,1] for each agent service in candidate set (i.e., agent services satisfying semantic threshold); then fidelity of each attribute is considered to evaluate QoS overall performance for every agent service comprehensively and objectively; finally, an agent service whose total QoS value is maximal is picked out from all candidates.

Concretely, suppose that there is a set of candidate agent services providing a certain service $PAS = \{pas_1, pas_2, \ldots pas_n\}$. By merging the quality vectors of all these candidates, a matrix $\mathbf{Q} = (Q_{i,j}, 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant 5)$ is built according with ***ASDM_SQ*** in Section 4, in which each row $Q_j$ corresponds to an agent service $pas_i$ while each column corresponds to a QoS attribute value. Let a matrix $\mathbf{F} = (fid_{i,j},\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant 4)$ denote confidence fidelity of the first four QoS attributes for agent service $pas_i$.

(1) Scaling phase.Some of the QoS attributes could be negative, i.e., the higher the value, the lower the quality, such as *time*, *cost*. Other QoS attributes are positive, i.e., the higher the value, the higher the quality, such as *reliability*, *maintainability*, and *reputation*. For negative attributes, values are scaled according to (9). For positive criteria, values are scaled according to (10).

$$M_{i,j} = \begin{cases} \frac{Q_j^{\max} - Q_{i,j}}{Q_j^{\max} - Q_j^{\min}} & \text{if } Q_j^{\max} - Q_j^{\min} \neq 0, \\ 1 & \text{if } Q_j^{\max} - Q_j^{\min} = 0, \end{cases} \quad \text{for } j = 1, 2 \tag{9}$$

$$M_{i,j} = \begin{cases} \frac{Q_{i,j} - Q_j^{\min}}{Q_j^{\max} - Q_j^{\min}} & \text{if } Q_j^{\max} - Q_j^{\min} \neq 0, \\ 1 & \text{if } Q_j^{\max} - Q_j^{\min} = 0, \end{cases} \quad \text{for } j = 3, 4, 5 \tag{10}$$

In the above equations, $Q_j^{\max}$ and $Q_j^{\min}$ are the maximal and minimal value of the $j$th QoS attribute, respectively. Let $\mathbf{M} = (M_{i,j},\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant 5)$ be normalizing matrix according to $\mathbf{Q}$, where $M_{i,j}$ denotes normalizing value of the $j$th QoS attribute for agent service provider $pas_i$.

(2) Weighting phase. The following formula is used to calculate the overall quality score of agent service provider $pas_i$.

$$qos\_score(pas_i) = \sum_{j=1}^{4} (w_j \cdot M_{i,j} \cdot fid_{i,j}) + w_5 \cdot M_{i,5} \tag{11}$$

where $w_j$ is the weight value of the $j$th QoS attribute, $0 \leqslant w_j \leqslant 1, \sum_{j=1}^{5} w_j = 1$. End users express their preferences regarding QoS by providing values for the weight $w_j$.

A case study of QoS matchmaking phase will be given for explaining the effect of our proposed algorithm. The following experiment method will be used to select the most satisfactory agent service. Suppose the returned candidate providers through Algorithms 1 and 2 have 10 agent services, i.e., $PAS = \{pas_1, pas_2, \ldots, pas_{10}\}$. That means these ten agent services can provide the same or closely similar capabilities.

The QoS values of 10 agent services are generated through the following simulation. Assume there are 50 similar tasks (or agent service requests). For each request, we randomly delegate it to an agent service, $pas_i$, from *PAS*, and randomly generate QoS attribute values and their fidelity values, denoting QoS value of $pas_i$ in this request. In these attributes, *time* is randomly distributed between 70 and 100 ms; *cost* is uniformly distributed [10,100] $; *reliability*, *maintainability*, *reputation* is randomly generated in [0,1]. The fidelity for the first four attributes is in [0,1]. The QoS values of 10 agent services for 50 requests is shown in Table 3, where *pn* denotes provided service number of $pas_i$ for all requests.

The QoS values in matrix $\mathbf{Q}$ and the confidence fidelity matrix of the first four QoS attributes for agent service $pas_i$ are the mean value of the results produced by its provided service number, respectively. The QoS values matrix $\mathbf{Q}$, confidence fidelity matrix $\mathbf{F}$, and normalizing matrix $\mathbf{M}$ are as follows.

$$\mathbf{Q} = \begin{bmatrix} 81.44, 68.27, 0.40, 0.31, 0.36 \\ 84.19, 60.16, 0.56, 0.49, 0.59 \\ 93.58, 67.80, 0.60, 0.54, 0.68 \\ 76.41, 45.65, 0.64, 0.62, 0.58 \\ 86.27, 49.76, 0.47, 0.54, 0.43 \\ 84.70, 43.44, 0.62, 0.50, 0.16 \\ 89.57, 35.88, 0.48, 0.50, 0.46 \\ 84.47, 59.18, 0.46, 0.50, 0.58 \\ 81.34, 54.97, 0.51, 0.40, 0.58 \\ 79.47, 40.63, 0.64, 0.59, 0.56 \end{bmatrix},$$

$$\mathbf{F} = \begin{bmatrix} 0.34, 0.15, 0.41, 0.40 \\ 0.68, 0.47, 0.81, 0.36 \\ 0.54, 0.53, 0.41, 0.62 \\ 0.41, 0.76, 0.38, 0.26 \\ 0.51, 0.69, 0.43, 0.65 \\ 0.35, 0.30, 0.46, 0.48 \\ 0.60, 0.18, 0.66, 0.67 \\ 0.51, 0.42, 0.37, 0.46 \\ 0.42, 0.65, 0.50, 0.47 \\ 0.51, 0.65, 0.45, 0.47 \end{bmatrix},$$

$$\mathbf{M} = \begin{bmatrix} 0.71, 0.00, 0.00, 0.00, 0.37 \\ 0.55, 0.25, 0.65, 0.55, 0.83 \\ 0.00, 0.01, 0.83, 0.72, 1.00 \\ 1.00, 0.70, 0.99, 1.00, 0.81 \\ 0.43, 0.57, 0.31, 0.73, 0.51 \\ 0.52, 0.77, 0.90, 0.60, 0.00 \\ 0.23, 1.00, 0.34, 0.60, 0.56 \\ 0.53, 0.28, 0.26, 0.61, 0.81 \\ 0.71, 0.41, 0.46, 0.29, 0.79 \\ 0.82, 0.85, 1.00, 0.89, 0.77 \end{bmatrix} \tag{12}$$

Suppose that weight value of each QoS attribute defined by users is 0.15, 0.2, 0.2, 0.15 and 0.3, respectively. The overall quality scores of each agent service provider by using formula (11) are:

**Table 3**
The QoS values of 10 agent services for 50 request.

| *pas* | *pn* | QoS attribute value. (*time*, *cost*, *reliability*, *maintainability*, *reputation*) |
|---|---|---|
| $pas_1$ | 4 | (81.364, 94.46, 0.069, 0.614, 0.575), (73.451, 35.37, 0.838, 0.042, 0.187), (71.636, 99.96, 0.304, 0.106, 0.656), (99.294, 43.29, 0.384, 0.493, 0.011) |
| $pas_2$ | 5 | (84.663, 23.28, 0.740, 0.456, 0.716), (79.636, 70.50, 0.746, 0.767, 0.616), (91.45, 26.392, 0.427, 0.235, 0.245), (91.586, 92.89, 0.124, 0.888, 0.587), (73.615, 87.73, 0.738, 0.082, 0.804) |
| $pas_3$ | 3 | (99.547, 94.69, 0.690, 0.593, 0.930), (90.360, 86.49, 0.422, 0.235, 0.356), (90.825, 22.23, 0.678, 0.784, 0.763) |
| $pas_4$ | 4 | (77.087, 90.03, 0.947, 0.393, 0.697), (74.888, 44.23, 0.978, 0.834, 0.624), (80.114, 12.45, 0.502, 0.948, 0.990), (73.533, 35.89, 0.114, 0.325, 0.026) |
| $pas_5$ | 9 | (82.862, 32.69, 0.708, 0.043, 0.986), (73.595, 59.18, 0.611, 0.816, 0.219), (97.835, 69.63, 0.150, 0.017, 0.724), (91.110, 11.66, 0.540, 0.945, 0.221), (97.034, 54.32, 0.205, 0.832, 0.493), (93.738, 58.43, 0.659, 0.478, 0.182), (70.386, 36.56, 0.674, 0.919, 0.364), (70.894, 60.31, 0.288, 0.810, 0.637), (98.963, 65.05, 0.412, 0.013, 0.052) |
| $pas_6$ | 4 | (73.856, 41.60, 0.411, 0.647, 0.385), (89.247, 25.84, 0.409, 0.542, 0.114), (87.707, 16.07, 0.649, 0.706, 0.112), (88.005, 90.24, 0.991, 0.102, 0.045) |
| $pas_7$ | 5 | (91.256, 34.28, 0.566, 0.295, 0.087), (91.973, 61.65, 0.708, 0.659, 0.263), (89.541, 24.46, 0.096, 0.515, 0.247), (89.445, 27.47, 0.342, 0.484, 0.978), (85.628, 31.53, 0.692, 0.551, 0.706) |
| $pas_8$ | 8 | (82.750, 60.07, 0.224, 0.790, 0.624), (97.355, 89.42, 0.283, 0.307, 0.567), (71.577, 88.84, 0.643, 0.443, 0.804), (80.630, 51.27, 0.360, 0.116, 0.666), (97.848, 46.98, 0.243, 0.958, 0.529), (82.601, 50.14, 0.681, 0.785, 0.296), (75.847, 14.88, 0.266, 0.178, 0.547), (87.174, 71.81, 0.983, 0.453, 0.635) |
| $pas_9$ | 4 | (93.761, 60.72, 0.447, 0.177, 0.424), (75.342, 47.91, 0.730, 0.548, 0.783), (77.398, 70.89, 0.231, 0.295, 0.638), (78.867, 40.35, 0.624, 0.599, 0.457) |
| $pas_{10}$ | 4 | (88.814, 56.19, 0.883, 0.563, 0.029), (80.738, 64.78, 0.728, 0.648, 0.872), (75.470, 20.82, 0.637, 0.401, 0.752), (72.877, 20.72, 0.305, 0.750, 0.601) |

$$\begin{aligned} &qos\_score(pas_1) = 0.147, \quad qos\_score(pas_2) = 0.463,\\ &qos\_score(pas_3) = 0.437, \quad qos\_score(pas_4) = 0.525,\\ &qos\_score(pas_5) = 0.363, \quad qos\_score(pas_6) = 0.200,\\ &qos\_score(pas_7) = 0.332, \quad qos\_score(pas_8) = 0.368,\\ &qos\_score(pas_9) = 0.402, \quad qos\_score(pas_{10}) = 0.558 \end{aligned} \tag{13}$$

It is easy to see from (13) that $pas_{10}$ has maximal QoS score among all agent services, and this agent service will be selected as optimal one with QoS performance.

In traditional agent service matchmaking algorithm, the first agent service is usually selected as match result, i.e., $pas_1$. In Zhang's algorithm [13], the track records concept is equivalent to the fifth QoS attribute in our algorithm, i.e., reputation. So, the corresponding evaluation matrix in Ref. [13] by using Zhang's algorithm is as follows:

$$R = \begin{array}{c} \begin{matrix} A_1 & & & & & \cdots & & & & A_{10} \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 2 & 2 & 0 & 2 & 1 & 0 & 1 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 3 & 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 3 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} strong\ satisfaction \\ satisfaction \\ weak\ satisfaction \\ neutral \\ weak\ unsatisfaction \\ unsatisfaction \\ strong\ unsatisfaction \end{array} \tag{14}$$

where each row denotes numbers of different satisfactory degree (strong satisfaction, satisfaction, etc.) for each agent. The result QoS scores of each agent are: −1, 1.33, 1.33, 0.67, −1.33, −3.33, −1, 1.33, 0.67, 1. As a result, $pas_2$, or $pas_3$, or $pas_8$ with maximal value (1.33) will be selected. Finally, Jiang's algorithm [14] will be compared with our proposed algorithm, and fidelity factor is not considered by Jiang's algorithm. The overall QoS scores of each agent service by using their algorithm are: 0.218, 0.594, 0.576, 0.880, 0.503, 0.501, 0.562, 0.521, 0.562, 0.858. Obviously, $pas_4$ (score is 0.880) is optimal agent service.

We can see from normalizing matrix **M** in formula (12), QoS performance (0.82, 0.85, 1.00, 0.89, 0.77) of $pas_{10}$ gives slightly worse than performance (1.00, 0.70, 0.99, 1.00, 0.81) of $pas_4$. But, fidelity performance of QoS attributes of $pas_{10}$ (0.51, 0.65, 0.45, 0.47) is evidently better than that of $pas_4$ (0.41, 0.76, 0.38, 0.26). This phenomenon shows that some service providers usually claim their higher QoS attribute performance like $pas_4$, but if the confidence fidelity or users' feeling degree is also considered when selecting agent service, the overall quality or performance for these services is not necessarily optimal. This point accords with the practical situation on Internet, such as e-business, online shopping, in which, people have always compromised selection need service between quality and users' evaluation. Therefore, our proposed QoS matchmaking algorithm is effective and reasonable.

## 6. Simulation experiments on QoS matchmaking

Effectiveness of our proposed QoS matchmaking algorithm has been shown in Section 5.3. Next, we will further evaluate QoS matchmaking algorithm through simulation experiments.

### 6.1. Impact of fidelity factor on matchmaking

One of important characteristics in our proposed algorithm is introducing of fidelity factor for QoS attributes. To evaluate the impact of fidelity on the final matchmaking results, we conduct the following simulations and compare the matchmaking results of our algorithm with Jiang's algorithm [14].

For the sake of simple, suppose that there are 10 agent services satisfying semantic similarity constraint to be selected. At each experiment, we randomly generate QoS attributes and fidelity val-

ues of each agent service according to the following strategy: *time* is randomly distributed between 10 and 100 ms; *cost* is uniformly distributed [70,100] $; *reliability*, *maintainability*, *reputation* is randomly generated in [0,1]. The fidelity for the first four attributes is in [0,1]. At each experiment point, we record average values $\overline{M}$ of normalizing matrix **M** and average fidelity values $\overline{F}$ of matrix **F** of selected agent service with highest score by using Jiang's algorithm and our algorithm, respectively, i.e.,

$$\overline{M}(i) = \frac{\sum_{j=1}^{5} M_{ij}}{5}, \overline{F}(i) = \frac{\sum_{j=1}^{4} fid_{ij}}{4} \tag{15}$$

where $i$ is numbering of agent service with highest QoS performance selected by using Jiang's algorithm or our algorithm. Let $Q_{ei}(i)$ in formula (16) denote QoS evaluation index for current algorithm, in order to measure final QoS evaluation of selected agent service. The higher the $Q_{ei}$, the greater the comprehensive quality performance. In which, $\sigma_1$ and $\sigma_2$ are evaluation weight for QoS attribute and fidelity factor.

$$Q_{ei}(i) = \sigma_1 \cdot \overline{M}(i) + \sigma_2 \cdot \overline{F}(i), \sigma_1 = \sigma_2 = 0.5 \tag{16}$$

Simulations are divided into three groups for different weight values of the QoS attributes in formula (11) considering different users' quality preference. *Group 1:* $w_1 = w_2 = w_3 = w_4 = w_5 = 0.2$, denoting unbiasedness; *Group 2:* $w_1 = w_2 = 0.35, w_3 = w_4 = w_5 = 0.1$, showing preference for *time* and *cost*; *Group 3:* $w_1 = w_2 = 0.125, w_3 = w_4 = w_5 = 0.25$ showing preference for other attributes. For each group, we run the experiments 100 times. The simulation results are shown in Fig. 6.

It can be seen from three subfigures in Fig. 6 that our algorithm gives slightly higher performance than Jiang's algorithm. In some experiment points, same agent service is selected by two algorithms. The results demonstrate better compromised selection between quality and users' evaluation in our algorithm.

### 6.2. Impact of attribute weight values on matchmaking

To show the impact of weight values of QoS attributes on matchmaking, a similar simulation method to example in Section 5.3 will be adopted. For each experiment point, assume there are 100 similar tasks (or agent service requests) and 15 candidate agent service providers. For each request, we randomly delegate a request to an agent service, and randomly generate QoS attribute values and its fidelity values. In these attributes, *time* is randomly distributed between 10 and 100 ms; *cost* is uniformly distributed [10,150] $; *reliability*, *maintainability*, *reputation* is randomly generated in [0,1]. The fidelity for the first four attributes is in [0.5,1]. The maximal and minimal values of each attribute for 15 agent services are recorded in each point.

Simulations are divided into three groups for different weight values according to different users' quality preference as following.

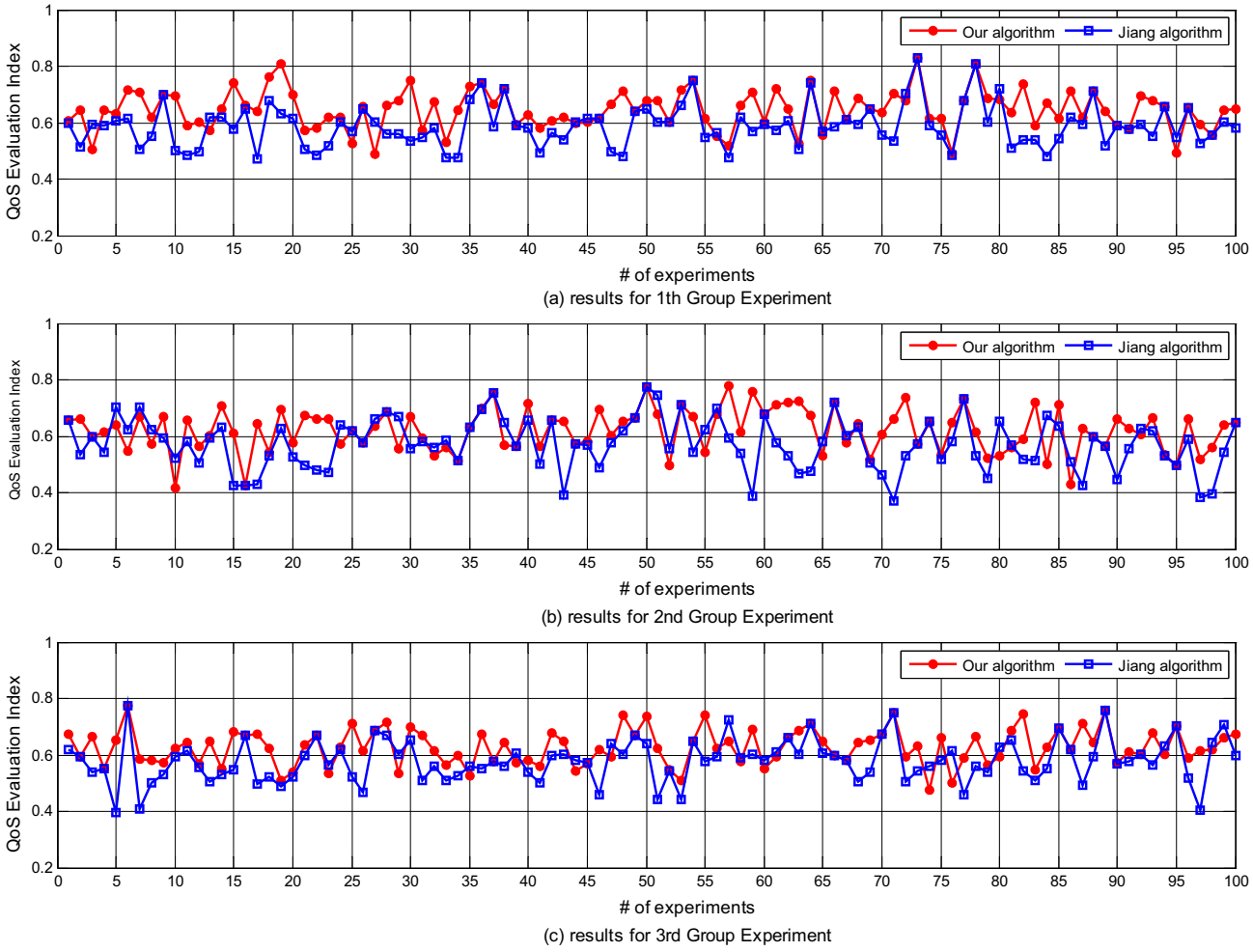


**Fig. 6.** A comparison on QoS evaluation index of selected optimal agent service for three different group.

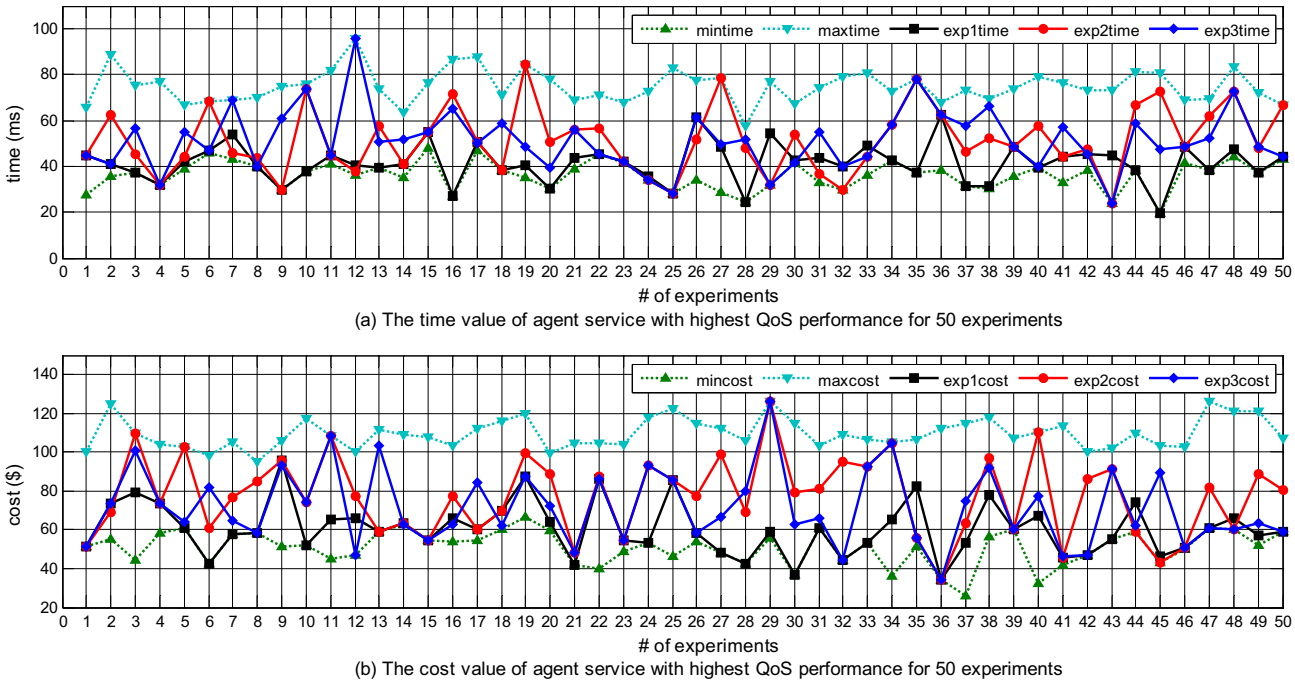


**Fig. 7.** The time and cost values of agent service with highest QoS performance.

*Group 1:* $w_1 = w_2 = 0.35, w_3 = w_4 = w_5 = 0.1$, showing preference for *time* and *cost*;

*Group 2:* $w_1 = w_2 = 0.1, w_3 = w_4 = 0.35, w_5 = 0.1$, showing preference for *reliability* and *maintainability*;

*Group 3:* $w_1 = w_2 = w_3 = w_4 = 0.15, w_5 = 0.4$, showing preference for *reputation*.

For each group, we run the experiments 50 times and select an agent service with highest score by using our algorithm for each time. Then we observe the relationship between maximal/minimal values and its attribute value.

Fig. 7 shows the *time* and *cost* values of selected agent service with highest QoS performance. It is easy to see from two subfigures that the experiment results in *Group 1* are most close to their minimal values due to negative attributes, i.e., black curves.

Fig. 8 shows the *reliability* and *maintainability* values of selected agent service with highest score. We can see that the experiment results in *Group 2* are most close to their respective maximal val-

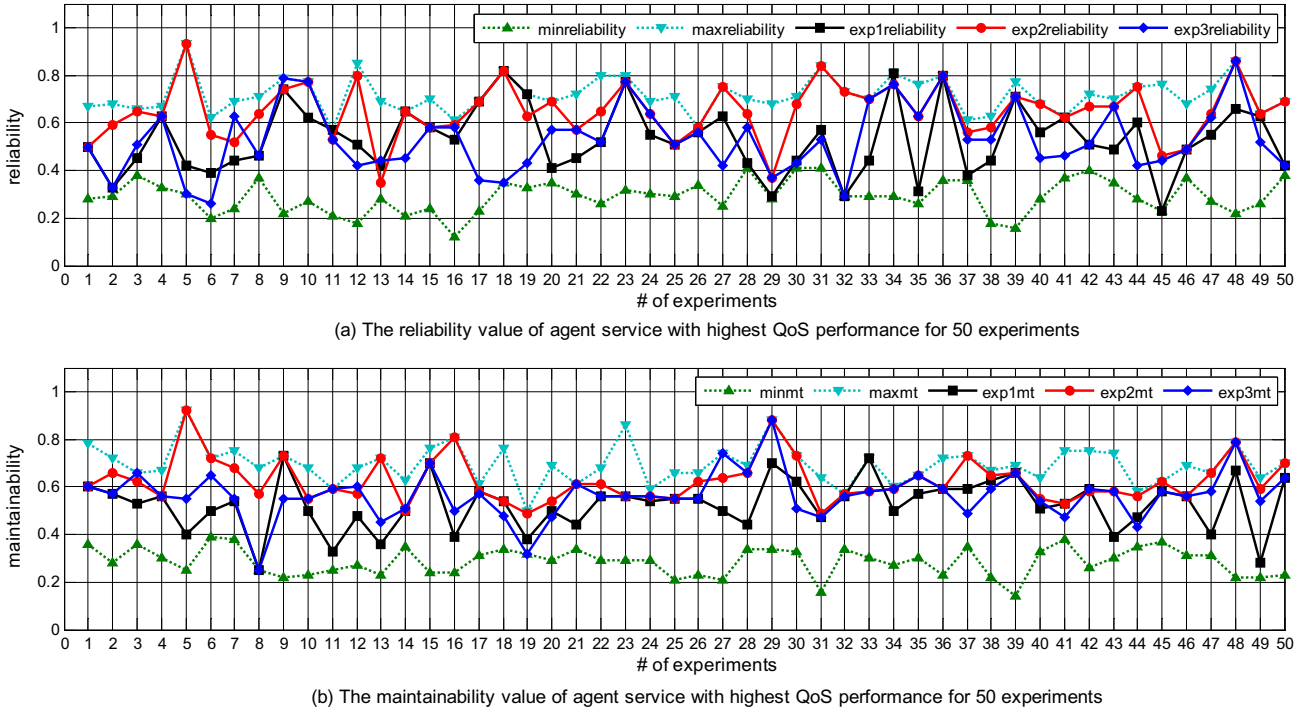


**Fig. 8.** The reliability and maintainability values of agent service with highest QoS performance.

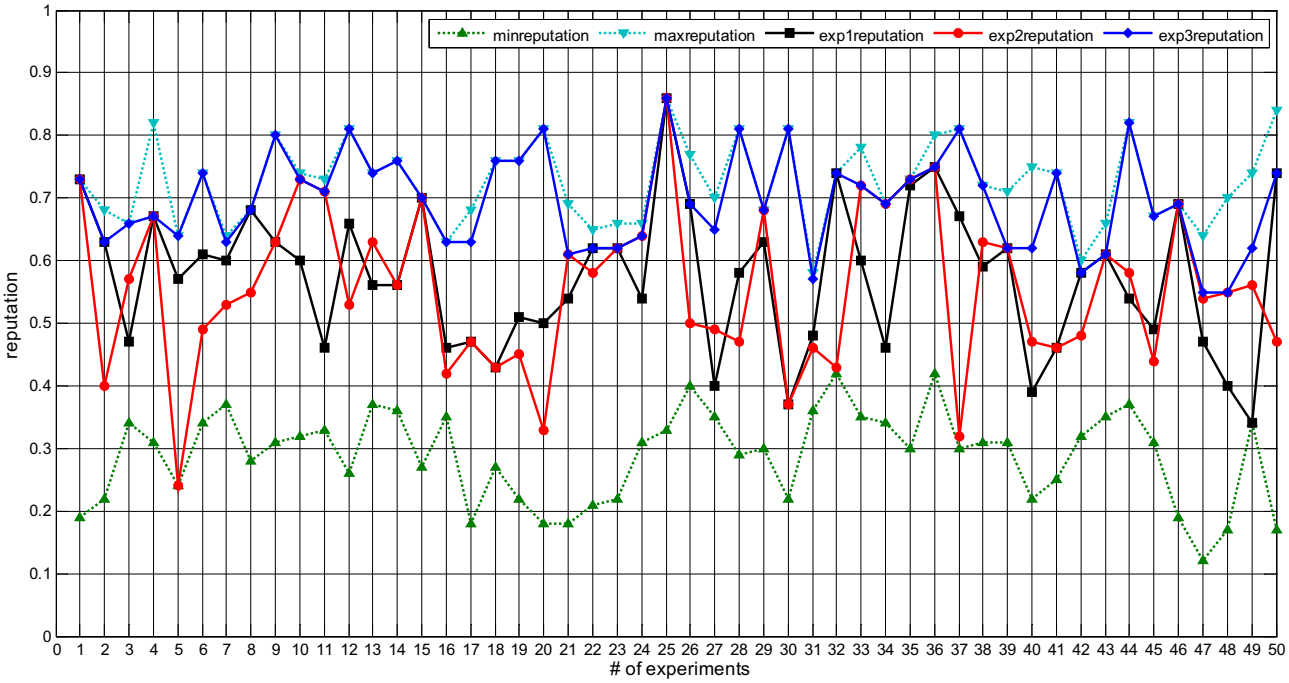


**Fig. 9.** The reputation values of agent service with highest QoS performance.

ues, i.e., two red curves. This phenomenon is expected because they belong to positive attributes, the higher value, the better performance.

Finally, the *reputation* values of selected agent service with highest score are shown in Fig. 9. Due to same reason for Fig. 8 and preference for *reputation* on weight value, the experiment results in *Group 3* are most close to maximal values, i.e., blue curve.

## 7. Conclusion

In this paper, we discuss the agent service matchmaking problem for autonomic element, which has been taken as one of the most important issue in field of autonomic computing. We consider together the factors of semantic and QoS, and propose an agent service description model named ***ASDM_SQ***. Further, a matchmaking algorithm so-called ***ASMA_SQ*** with semantic and QoS constraints is presented to find the agent service satisfying both the given semantic similarity threshold and optimal QoS performance. The whole algorithm consists of two phases, one is semantic similarity matchmaking, and the other is QoS similarity matchmaking.

***ASMA_SQ*** algorithm has following features:

(1) The semantic and QoS factors are considered synthetically, which can both overcome the shortcoming of heterogeneity of service description based on traditional syntax, increase precision ratio, and satisfy different needs came from agent service requestor to service performance;
(2) During matchmaking for semantic similar matchmaking, the distance-based and set-theory methods are used to compute similar, respectively. These methods are simple and effective. Moreover, they can be used to overcome the shortcoming that the conceptions are difficult to be distinguished on equal conditions, and the matchmaking result is not perfect;
(3) The final characteristic is that we introduce evaluation mechanism of confidence of individual QoS attributes during QoS matchmaking, i.e., fidelity factor for each attribute, which will overcome drawbacks such as subjectiveness and unfairness and improve the self-configuration capability for agent element. Simulation experiments demonstrate the effective and correction of our algorithm for agent service matchmaking, and perform better performance in terms of QoS than other existing algorithm. Simulations show also that our algorithm has better compromise between attribute quality and users' evaluation when selecting agent service.

## Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No. 90718021.

## References

[1] M. Parashar, S. Hariri, Autonomic computing: an overview, Lecture Notes in Computer Science 3566 (2005) 257–269.
[2] J. Kephart, D. Chess, The vision of autonomic computing, IEEE Computer Society (1) (2003) 41–59.
[3] T. Gerald, M.C. David, E.W. William, D. Rajarshi, S. Alla, W. Ian, O.K. Jeffrey, R.W. Steve, A multi-agent systems approach to autonomic computing, in: AAMAS'04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, USA, 2004, pp. 464–471.
[4] D.W. Tom, H. Tom, Towards autonomic computing: agent-based modeling, dynamical systems analysis, and decentralized control, in: INDIN'03: Proceedings of the First IEEE International Conference on Industrial Informatics, Banff, Alberta, Canada, 2003, pp. 470–479.
[5] G.J. Wickler, Using Expressive and Flexible Action Representations to Reason about Capabilities for Intelligent Agent Cooperation, Ph.D. Thesis, University of Edinburgh, Edinburgh, UK, 1999.
[6] K. Sycara, S. Widoff, M. Klusch, J.G. Lu, LARKS: dynamic matchmaking among heterogenous software agents in cyberspace, Autonomous Agents and Multi-Agent Systems 5 (2) (2002) 173–203.
[7] K. Sycara, M. Klusch, J.G. Lu, S. Widoff, Matchmaking among heterogeneous agents on the Internet, in: Proceedings of AAAI Spring Symposium on Intelligent Agents in Cyberspace, Stanford, USA, 1999, pp. 40–46.
[8] M. Klusch, K. Sycara, Brokering and matchmaking for coordination of agent societies: a survey, in: A. Omicini et al. (Eds.), Coordination of Internet Agents, Springer, 2001.
[9] K. Arisha, S. Kraus, F. Ozcan, R. Ross, V.S. Subrahmanian, T. Eiter, S. Kraus, Impact: a platform for collaborating agents, IEEE Intelligent Systems 14 (2) (1999) 64–72.
[10] H.E. Lu, Agent services description and matching, in: PDPTA'03: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA, 2003, pp. 1040–1048.

[11] Z.Z. Shi, Y.C. Jiang, H.J. Zhang, M.K. Dong, Agent service matchmaking based on description logic, Chinese Journal of Computers 27 (5) (2004) 625–635.
[12] J. Hu, J. Gao, B. Zhou, B.S. Liao, J.J. Chen, Ontology based agent services compatible matchmaking mechanism, in: ICMLC'03: Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, PR China, 2004, pp. 111–116.
[13] Z.L. Zhang, C.Q. Zhang, An improvement to matchmaking algorithms for middle agents, in: AAMAS'02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, Bologna, Italy, 2002, pp. 1340–1347.
[14] Y.C. Jiang, Z.Z. Shi, Quality of service driven agent service matchmaking, Mini–Micro Systems 26 (4) (2005) 687–692.
[15] N.R. Jennings, On agent-based software engineering, Artificial Intelligence 177 (2) (2000) 277–296.
[16] Y.H. Li, Z.A. Bandar, D. McLean, An approach for measuring semantic similarity using multiple information sources, IEEE Transactions on Knowledge and Data Engineering 15 (4) (2003) 871–882.
[17] M.A. Rodriguez, M.J. Egenhofer, Determining semantic similarity among entity classes from different ontologies, IEEE Transactions on Knowledge and Data Engineering 15 (2) (2003) 442–456.
[18] A. Tversky, Features of similarity, Psychological Review 84 (4) (1977) 327–352.