SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2013, Vol. 56 012102:1–012102:24
doi: 10.1007/s11432-012-4697-1

A broker-based semantic agent for discovering Semantic Web services through process similarity matching and equivalence considering quality of service

ÇELİK Duygu[1*] & ELÇİ Atilla[2]

[1]*Computer Engineering Department, Istanbul Aydin University, Istanbul, Turkey;*
[2]*Electrical-Elecronics Engineering Department, Aksaray University, Aksaray, Turkey*

Received October 10, 2011; accepted May 13, 2012; published online October 10, 2012

Abstract The lack of semantic descriptions for "web service properties" makes it difficult to find suitable web services. Current solutions are mostly based on broker/mediator agent systems. However, these techniques are syntactical, rather than semantics oriented. This article presents a semantic matching approach for discovering Semantic Web services through a broker-based semantic agent (BSA). The BSA includes knowledge-bases and several processing steps. The BSA's knowledge-bases are concept, task, and process ontologies built to describe both functional and non-functional parameters of services. The BSA executes semantic-based matching algorithms to discover similar services through the semantic matching step, process equivalence task, and matching of quality of service parameters. Relevant services are ranked by client preferences utilizing the semantic descriptions of available services. Other matchmaker studies are reviewed and compared with the BSA. Performance of the BSA algorithm is compared with SAM using published data and an experimental setup. The results indicate that our approach is better and more effective in some respects.

Keywords Semantic Web, Semantic Web services, broker web applications, ontologies, semantic agents

Citation ÇELİK D, ELÇİ A. A broker-based semantic agent for discovering Semantic Web services through process similarity matching and equivalence considering quality of service. Sci China Inf Sci, 2013, 56: 012102(24), doi: 10.1007/s11432-012-4697-1

1 Introduction

The World Wide Web is still a platform for innovative application development. However, the lack of semantic descriptions for web services makes it difficult to find suitable web services when querying a service request by a user. The discovery process of web services entails matching semantic descriptions obtained from the client and the service provider. The well known semantic matchmaking algorithm proposed by Paolucci et al. [1] has been extended and is cited extensively in recent proposals [2–14].

In this article, we present a semantic matchmaking approach for a broker-based semantic agent (BSA), which is able to understand the meaning of a user request and retrieves semantically suitable web services through a Semantic Web. Novel aspects of the proposed BSA are the innovative semantic matching step

*Corresponding author (email: duygucelik@aydin.edu.tr)

(SMS), process equivalence task (PET), and matching of quality of service (QoS) parameters (MQP), which help to find the required suitable services through semantic matchmaking. These steps make BSA a cascading filtering mechanism that finds the best matched required process from a set of candidate web services. The SMS considers the concepts and performs semantic matching on focused concepts according to their meanings, similarities, and distance of concept relations. The related set captured from the candidate Semantic Web services (SWSs) for a particular query is filtered once more by the PET. As a final step in the BSA, the related set obtained after the PET is filtered for a third time according to the client's requested QoS metrics during the MQP. We demonstrate the SMS functionality using the scenario of a broker in a financial investment domain for services by product suppliers.

The web ontology language (OWL) [15,16] and web ontology language for services (OWL-S) [17] are used for the studied domains and SWSs, respectively.

We have implemented only the SMS and PET parts of the BSA algorithm using JBuilder. Two ontology RDF APIs, Jena 2.2 [18] and OWL-S API [19], are used to parse the required semantic information in the OWL and OWL-S files, respectively, for reasoning operations.

In summary, this article presents a framework for a BSA to discover appropriate services based on a client request. The following sections discuss the implementation mechanisms for the BSA approach, and present an innovative way of using ontologies for agent applications while considering e-commerce business cases.

The rest of the paper is organized as follows: Section 2 presents ontology knowledge-bases used by the BSA. Section 3 investigates embedding QoS requirements into the ontology knowledge-bases. In Section 4, the fundamentals of the BSA approach are explained. Section 5 describes the design aspects of process matching by means of the SMS and considers case studies. Section 6 discusses the PET. The step for MQP is described and discussed in Section 7, which concludes the discovery task. Section 8 surveys recently proposed solutions comparing them with the BSA approach. In Sections 9 and 10, we present our evaluation and conclusions, respectively.

2 Preparing ontologies for implementation of the BSA approach

The Semantic Web is an extension of the current web in which information has a well-defined meaning. This approach enables better cooperation between computers and people [20]. An ontology (expressed in OWL as the conceptual language of the Semantic Web) is a specification of a conceptualization of a knowledge domain. It is a controlled vocabulary that formally describes concepts and their relationships. According to W3C [16], OWL is a family of knowledge representation languages for reasoning through ontologies.

OWL-S is an OWL-based web services ontology that provides web service suppliers with a core set of markup languages, constructs for describing properties and functions, and the grounding information of web services in a computer-interpretable form. All structural information of web services can be described through the OWL-S ontology language. OWL-S includes three complimentary models, namely, profile, process, and grounding. The profile model describes what the service provides or does, while the process model is a functional description of how the service works. Finally, the grounding model describes how to access the service (providing a link between the OWL-S and web services description language (WSDL) descriptions of the service).

The inputs/outputs/preconditions/effects (I/O/P/E) are concepts of an atomic process in the OWL-S process model. Inputs and outputs specify data transformations produced by the process. Inputs describe the information that is requested from a user/agent before executing a process, while outputs describe the information that the process produces after its execution. Preconditions are conditions that are imposed on the inputs for a process prior to its invocation. Execution of a process may also result in changes to the state of the environment, i.e., effects. An effect can be defined as a proposition that becomes true after execution completes.

The BSA uses three ontology knowledge-bases, namely, concepts, processes, and tasks. Effective detailing of the knowledge-bases will aid understanding of the mechanics of the BSA and thus, these are

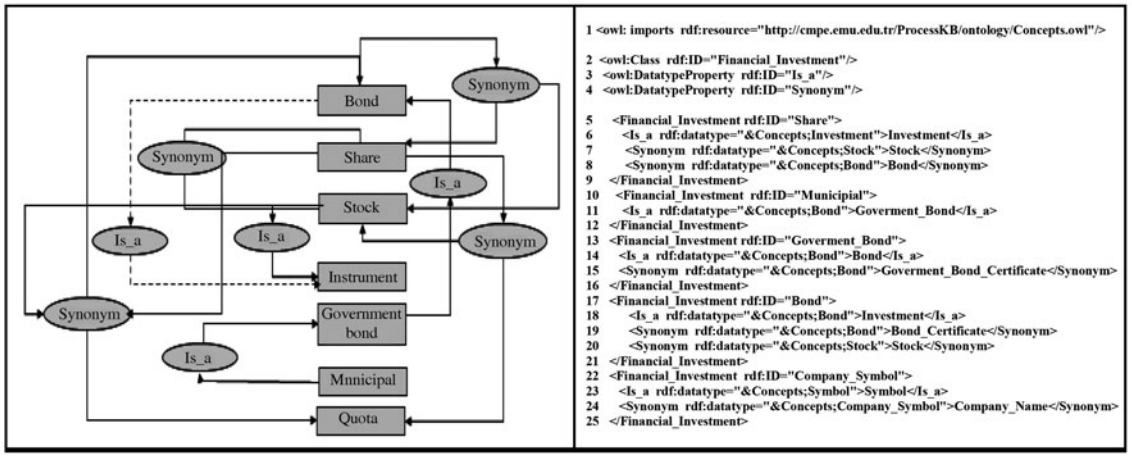

Figure 1 Part of the financial investment domain ontology.

explained in detail in the following subsections.

2.1 Concepts ontology

The BSA uses the concepts ontology (CO) to retrieve relationships among concepts. The ontology file stores metadata for the corresponding domains. The CO contains many concepts and relations that are likely to be common to several domains used in the e-commerce business field. The CO is a widely used concepts ontology found in a test collection created by the SemWebCentral community[1)]. In addition, several domain ontologies (Transport.owl, Financial_Investment.owl, Vehicle.owl, and Software.owl) have been created and incorporated in the CO by means of the OWL Plug-in 2.2, which is a Tab Widget of Protégé implemented by Stanford University [21].

Figure 1 shows a portion of the financial investment domain ontology, which includes financial concepts of the financial investment domain, such as share, stock, quote, and so on.

Two data type properties are used in the domain ontologies by means of the OWL tag (<owl:DatatypeProperty>), namely, Synonym and Is_a (lines 3 and 4 in Figure 1). For example, the concept "Bond" has two synonyms, "Bond_Certificate" and "Stock" in the domain ontology for Financial_Investment (lines 17–21 in Figure 1). In addition, "Municipial" has only one Is_a relation, that is, "Goverment_Bond", which is associated with the concept "Bond" in the ontology (lines 10–12 in Figure 1). To provide the association between the CO and the created domain ontologies, the URL of the CO should be inserted at the beginning of each domain ontology (as depicted in line 1 in Figure 1).

The defined concepts (in the CO or interest domain ontology) provide associations between I/O/P/E terms of the declared tasks in the tasks ontology (TO) and I/O/P/E terms of atomic processes of candidate SWSs in the knowledge-base for the processes ontology (PO). Details of their associations are given in Subsection 2.4.

2.2 Tasks ontology

We propose a TO as a common ontology containing capabilities corresponding to common functionalities provided by web services. The TO provides an extensive ontology of functions or tasks. Processes of common web services for a particular domain may be defined as classes of task instances that represent their capabilities based on OWL descriptions in the TO. Furthermore, the BSA uses the TO when necessary, to distinguish processes according to their functional semantics (for instance, buyBond(Company_Symbol,Price) and sellBond(Company_Symbol,Price)). The following ontology snippet presents an excerpt from the TO model. The TO is used by the BSA in matching service capabilities during the inference phase of the SMS. The semantic task contexts are described in the TO using OWL

1) http://projects.semwebcentral.org

```
<!—Datatype Properties for I/O/P/E Parameters of Tasks –>

<owl:imports rdf:resource="http://cmpe.emu.edu.tr/ProcessKB/ontology/Concepts.owl"/>
1 <owl:DatatypeProperty rdf:ID="Input_Parameter"/>
2 <owl:DatatypeProperty rdf:ID="Output_Parameter"/>
3 <owl:DatatypeProperty rdf:ID="Effect_Parameter"/>
4 <owl:DatatypeProperty rdf:ID="Precondition_Parameter"/>

5 <owl:Class rdf:ID="Tasks" />
6 <owl:Class rdf:ID="e_commerce_Service">
7     <rdfs:subClassOf rdf:resource="#Tasks"/>
8 </owl:Class>

9 <owl:Class rdf:ID="Information_Service">
10     <rdfs:subClassOf rdf:resource="#e_commerce_Service"/>
11 </owl:Class>

12 <owl:Class rdf:ID="Buying_Service">
13     <rdfs:subClassOf rdf:resource="#e_commerce_Service"/>
14 </owl:Class>

15 <owl:Class rdf:ID="Selling_Service">
16     <rdfs:subClassOf rdf:resource="#e_commerce_Service"/>
17 </owl:Class>

18 <owl:Class rdf:ID="SearchStockQuote">
19     <rdfs:subClassOf rdf:resource="# Information_Service"/>
20     <Input_Parameter rdf:datatype="&Concepts;Company_Symbol">Symbol</Input_Parameter>
21     <Output_Parameter rdf:datatype="&Concepts;Price">Price</Output_Parameter>
22 </owl:Class>
```

Figure 2 A small portion of the initialization of the TO.

semantic tags such as <owl:Class>, <rdfs:subClassOf>, <owl:DatatypeProperty>, and <owl:ObjectProperty>. For example, in Figure 2 semantic task contexts are used to explain the Tasks, the e_commerce_Service, the Information_Service, the Buying_Service, the Selling_Service, and the SearchStockQuote concepts. These are described as an <owl:Class>, which keeps semantic-based descriptions of commonly used task instances for the e-commerce field. Line 5 declares that there is a class called Tasks using the tag <owl:Class>, while line 6 states that there is another class named e_commerce_Service, which is a kind of Tasks using the tag <rdfs:subClassOf> in line 7. Other classes of the task instances are described similarly.

Lines 1–4 present the defined <owl:DatatypeProperty> properties, namely, Input_Parameter, Output_Parameter, Precondition_Parameter, and Effect_Parameter, which are used to describe the I/O/P/E parameters of task instances. Lines 18–22 present a SearchStockQuote task that provides the means to obtain a quote from the given company name or symbol. Therefore, it contains an Input_Parameter (in line 20) that specifies the association between the SearchStockQuote task and the Input_Parameter via <owl:DatatypeProperty>. This means that the SearchStockQuote task contains only one input which is the Symbol and its ParameterType is Company_Symbol. The parameter type is also a concept and contains a URI that points to a concept in the CO by means of the declaration "&Concepts;Company_Symbol" (line 20 of Figure 2).

All the other output, precondition, and effect parameters of the SearchStockQuote task are described in the same format (output is defined in line 21 and there are no precondition and effect parameters for

```
<owl:imports rdf:resource="http://cmpe.emu.edu.tr/ProcessKB/ontology/Concepts.owl"/>
1 <service:Service rdf:ID="StockQuoteCompany">
2     <service:presents rdf:resource="#StockQuoteCompany-Profile"/>
3     <service:describedBy rdf:resource="#StockQuoteCompany-Process"/>
4     <service:supports rdf:resource="#StockQuoteCompany-Grounding"/>
5 </service:Service>
```

Figure 3 Process, profile, and grounding models of the stock quote company SWS.

```
1 <profile:Profile rdf:ID="StockQuoteCompany-Profile">
2 <service:isPresentedBy rdf:resource="#StockQuoteCompany-Service"/>
3     <profile:hasInput rdf:resource="#Symbol"/>
4     <profile:hasOutput rdf:resource="#Price"/>
5 </profile:Profile>
```

Figure 4 Profile model of the stock quote company SWS.

```
1 <process:AtomicProcess rdf:ID="StockQuoteCompany-Process">
2     <process:hasInput rdf:resource="#Symbol"/>
3     <process:hasOutput rdf:resource="#Price"/>
4 </process:AtomicProcess>
5   <process:Input rdf:ID="Symbol">
6     <process:parameterType rdf:datatype="&Concepts;Company_Symbol"/>
7 </process:Input>
8 <process:Output rdf:ID="Price">
9     <process:parameterType rdf:datatype="&Concepts;Price"/>
10 </process:Output>
```

Figure 5 Process model of the stock quote company SWS.

the SearchStockQuote task). In this manner, various meanings related to the task can be described in the semantic task contexts.

2.3 Knowledge-base for processes ontology

In the BSA, OWL-S files of a candidate set of SWSs are collected in the PO knowledge-base. The set of OWL-S files (semantic descriptions) of the atomic processes (in the PO) is a widely used test collection that is available from the SemWebCentral website. OWL-S descriptions of the entire SWS collection in the PO are described next. Figure 3 gives the OWL-S service model of a StockQuoteCompany (StockQuoteCompany.owl) in the PO knowledge-base, which is an online company stock quote web service found in the SWS-TC on the aforementioned website[2)].

The service's process takes a company's symbol as input and returns the stock price information. Parts of the OWL-S file of the StockQuoteCompany web service are given in Figures 3–6. The three complimentary models of the service are introduced in lines 2–4 of Figure 3: StockQuoteCompany-Profile, StockQuoteCompany-Process, and StockQuoteCompany-Grounding, respectively.

Figure 4 shows the profile model of the StockQuoteCompany service, which provides the information needed for an application to discover the service in terms of its I/O/P/E parameters. For example, hasInput is an OWL-S tag that denotes that the service takes one input (Symbol) defined in line 3 of Figure 4. The definition of hasOutput is another OWL-S tag that denotes that the service also has one output (Price) as defined in line 4 of Figure 4.

The service's process model gives a detailed description of the atomic process of the service and is referred to as the StockQuoteCompany-Process. The input of the atomic process is tagged by process:Input, that is, Symbol (depicted in line 5 of Figure 5).

2) http://projects.semwebcentral.org/projects/sws-tc/

```
1 <grounding:WsdlGrounding rdf:ID="StockQuoteCompany-Grounding">
2     <service:supportedBy rdf:resource="#StockQuoteCompany-Service"/>
3 <grounding:WsdlDocumentrdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://finance.yahoo.com/search</grounding:WsdlDocument>
4 </grounding:WsdlGrounding>
```

Figure 6 Grounding model of the stock quote company SWS.

The input parameter Symbol has a parameterType and its rdf:datatype is a class that points to the concept Company_Symbol in the CO (line 6 of Figure 5). The output definition is declared in the same way (lines 8 and 9 of Figure 5).

OWL-S is an XML-based language and its process declarations and input/output (I/O) types fit well with the WSDL. It is easy to extend existing WSDL bindings for use with OWL-S, such as the SOAP (see related work in Section 8) binding. As expressed in Figure 6, WsdlGrounding shows how the atomic StockQuoteCompany-Grounding, specified in OWL-S, can be given a grounding using the WSDL (in WsdlDocument).

2.4 Association between the ontology knowledge-bases in the BSA

In the following example, we consider two OWL-S declarations for two input terms, "Bond Certificate" and "Bond", which are declared in the OWL-S process models of SearchBondRate and FindBondRate SWSs, respectively, as given below.

```
<service:Service rdf:ID="SearchBondRate">
  <process:Input rdf:ID="Bond">
    <process:parameterType rdf:resource=" http://..ontology/Concepts.owl#Bond"/>
  </process: Input>
</service:Service>

<service:Service rdf:ID="FindBondRate">
  <process: Input rdf: ID="Bond_Certificate">
    <process: parameterType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </process:Input>
</service:Service>
```

Both these processes take one input: the SearchBondRate SWS process has input="Bond", while the FindBondRate SWS process has input="Bond_Certificate". Clearly these inputs have the same meaning; however, they are declared differently by their providers. Furthermore, the process:parameterType property of the first is defined as Concepts.owl#Bond and that of the second as XMLSchema#string. As is shown in the ontology code snippets of the two SWSs above, a service with process:parameterType defined as a string may be compared with any other service process:parameterType that holds a specific concept. This makes proper discovery more difficult owing to the availability of unrelated services due to inadequate semantic descriptions. Unfortunately, the use of standard XMLschema data types such as string, integer, double, etc. in the properties of process:parameterType can mislead semantic matching to search for suitable processes of candidate SWSs during discovery. During the discovery phase, a software agent would not notice the input equivalence of these two processes from either the process:parameterType property or rdf:ID (I/O terms) concepts.

Therefore, during the discovery phase, task matching is needed, which can comprehend the similarity between the tasks or processes. To solve this problem, the BSA uses the Concepts.owl (CO) knowledge-base association with particular domain ontologies.

We assume that: 1) all parameter types of the predefined I/O/P/E parameters of task instances (in the TO) are assigned by an ontologist; 2) all parameter types of the predefined I/O/P/E parameters of the OWL-S-based process descriptions of candidate SWSs (in the PO knowledge-base) are assigned by

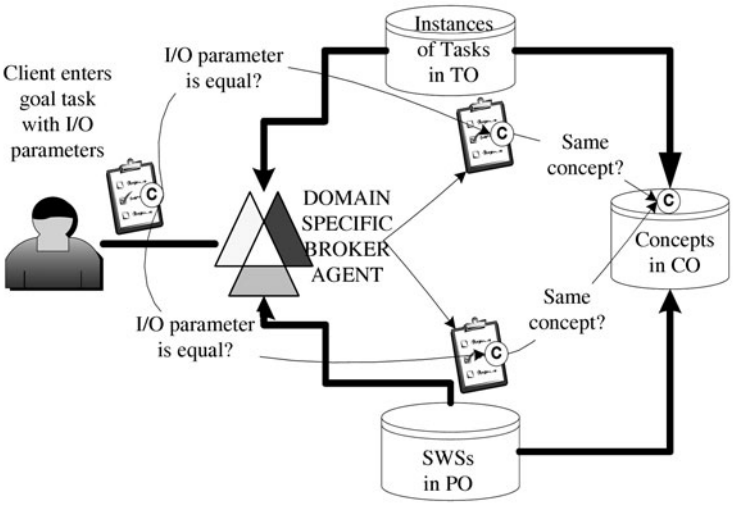

Figure 7 Discovering associations between the TO, PO, and CO in the BSA.

the service provider; and 3) each concept of the parameter types of these predefined I/O/P/E parameters (in the TO or PO) points to the corresponding concept (in the CO).

Briefly, both 1) and 2) indicate a predefined concept in the CO instead of the XMLschema data types (the concept is depicted as © in Figure 7). The association is necessary to recognize a probable functional similarity between a task instance (in the TO) and a process of the currently focused SWS (in the PO) during the matching step. Here, the matching supports reasoning to determine: "Does the task (in the TO) have the same purpose and parameters as the currently focused SWS?"

3 Embedding QoS requirements in the ontology

In the near future, web services will be required to adhere to standards and the QoS level will become an important selling point. Owing to the fast growth and large number of services on the Internet, QoS is becoming an important aspect in the discovery of web services and the assessment of service quality. A client is thus faced with the inevitability of selecting a suitable service supplier and may require a batch of different services. In such scenarios, knowledge of the quality of service is necessary to select an appropriate service and supplier. The BSA model is capable of providing web service clients with some measure of the quality of the discovered web services. To do this, we suggest that the service supplier define non-functional QoS parameters in the OWL-S document.

A QoS ontology model can be based on OWL semantics (i.e., classes, subclasses, data type properties, and object properties) since an upper ontology [22] defines and includes all the relations between QoS parameters, such as performance, cost, scalability, configuration, availability, accessibility, capacity, integrity, reliability, security, accuracy, and other attributes of interest. Figure 8 illustrates part of the ontology description of the throughput QoS parameter in OWL. Here, throughput is represented by a QoS class and has relations with other QoS parameter classes like Jitter, ErrorRate and Performance (defined in lines 2–10 of Figure 8).

The BSA uses the QoS profile definition to identify the best matched service, while the metric descriptions provide detailed measures of the service quality (a description of the metric for throughput is given in lines 11–18 of Figure 8 for the service). The metric for "ThroughputValue" is also given as 100000 in the ontology file of the service (line 13 of Figure 8).

Section 7 discusses the MQP task, which compares the queried and embedded values of the focused QoS parameters of both the client and candidate SWSs. In the next section, we present the details of the BSA approach.

```
1 < QoS Class rdf: ID="Throughput">
2  <hasRelation>
3    <Relation rdf:ID="Jitter">
4  </hasRelation>
5  <hasRelation>
6    <Relation rdf:ID="ErrorRate">
7  </hasRelation>
8  <hasRelation>
9    <Relation rdf:ID="Performance">
10  </hasRelation>
11  <hasMetric>
12  <Metric rdf:ID="ThroughputValue">
13    <Value rdf: Datatype=http://www.w3.org/2001/XMLSchema#String>100000
14    </Value>
15    <MetricType rdf: Datatype=http://www.w3.org/2001/XMLSchema#String>Double
16    </MetricType>
17  </Metric>
18  </hasMetric>
19 </QoS Class>
```

Figure 8 Part of the QoS parameter definition in OWL.

4 BSA mechanism

Ontologies can provide a semantic infrastructure for e-commerce [23–26]. Conceptual modeling may assist in gathering the information needed to connect web clients with service suppliers, through activities such as:

- Discovering an appropriate web service requested by a customer.
- Using product knowledge to search for a product required by the customer.

Briefly, this section explains the main steps carried out by the BSA in discovering an appropriate web service.

(i) Discovering a brokerage service. Through a simple query, the BSA provides a service to the customer via a domain-specific broker agent. The domain selection of the broker agent is carried out by the BSA according to the client's selection (e.g., the financial investment broker agent, referred to as an FIB), which has its own domain ontology, such as Financial_Investment.owl.

(ii) Discovering web services. As the next step, the client enters the I/O terms of the requested task, which are used in discovering the appropriate web services through the SMS. The SMS algorithm performs matching based on the requested I/O terms because a mapping is needed between the terms of the query and advertisements obtained in this phase. The outcome of the SMS for a focused atomic process (queried task) is an association between its I/O concepts and the concepts of all other processes (belonging to candidate SWSs). A similar technique can be used for precondition and effect matching by the SMS. Without loss of generality, we do not discuss precondition and effect matching in the rest of this paper. On conclusion of the SMS, the BSA may have retrieved several suitable processes from the set of candidate SWSs containing the same I/O concepts as the required I/O. The appropriate retrieved processes are ordered according to their SMS scores and then passed to the next filtering and scoring step of the BSA, that is, the PET. The SMS algorithm is explained in detail in Section 5.

(iii) Process equivalence task. This determines the equivalence of two focused processes according to the similarity scoring. The client goal process is compared against each appropriate SWS found during the SMS, to determine whether they have the same purpose. This step is explained in detail in Section 6.

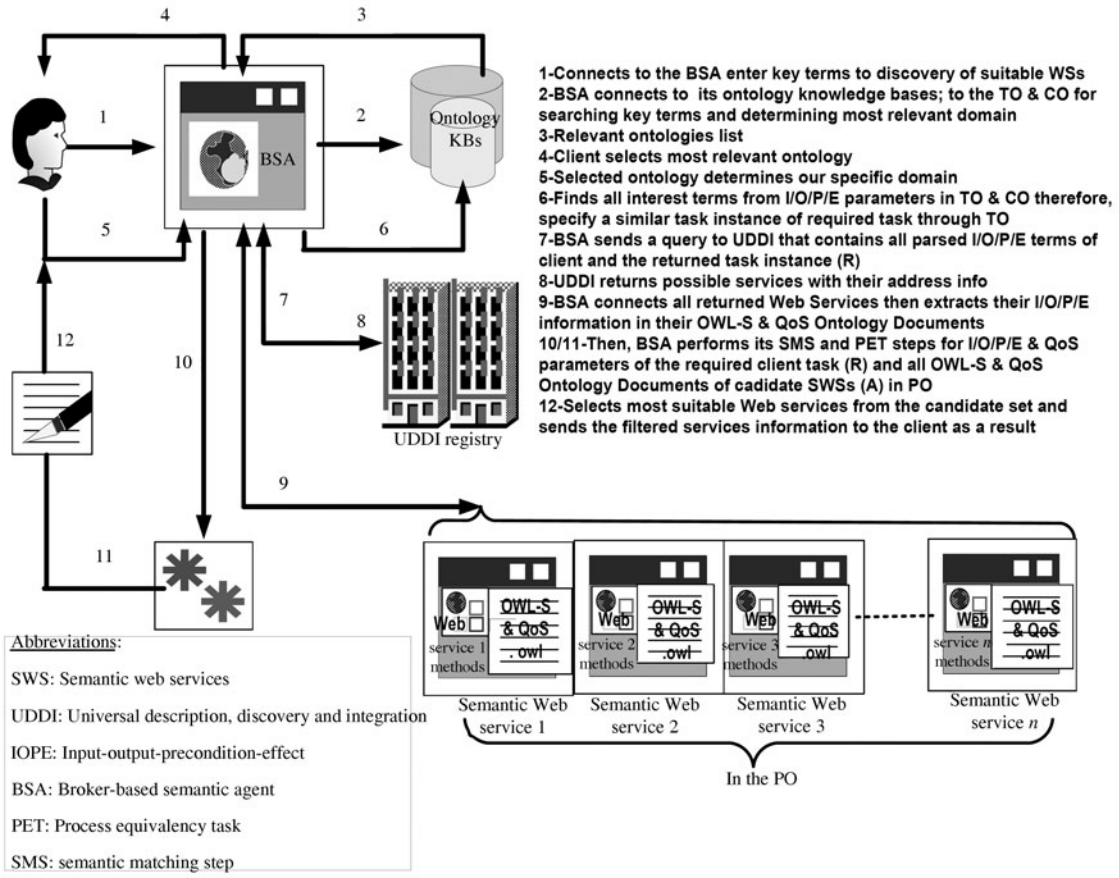

Figure 9 Diagram of the BSA workflow.

(iv) Matching of QoS task. After Step (iii), the QoS requirements are checked against the customer's requirements.

Before considering the details of the SMS, PET and MQP, we consider the actual mechanism of the BSA. Figure 9 illustrates the proposed BSA approach. Initially, the client enters the requested word(s) as the search parameter (ref. 1 in Figure 9). Then the BSA starts the search operation by specifying interest domains in the ontologies and task instances (ref. 2 in Figure 9). The agent retrieves all suitable domain ontologies and similar task instances (ref. 3 in Figure 9) and prepares an interface to ask the consumer which ontology and task instance are relevant to the intended search (ref. 4 in Figure 9). After the client has made a selection (ref. 5 in Figure 9), the agent fetches the selected ontology, from which it obtains all available synonym/is_a concepts for all client terms from the ontology knowledgebases (ref. 6 in Figure 9). Initially, all related concepts found (I/O concepts of both the client and the selected relevant task instance, R, in the TO) are sent to the universal description, discovery and integration (UDDI) repository through a query to retrieve all candidate web services (ref. 7 in Figure 9). Here, we assume that all ontology files (OWL-S) of the retrieved web services (candidates) after UDDI querying are collected in the PO knowledgebase (ref. 8 in Figure 9). Thereafter, the agent connects to the OWL-S files of all returned candidate web services from the UDDI and parses their profile and process information, including concepts of the I/O and QoS metric parameters (refs. 8 and 9 in Figure 9). The agent starts executing the SMS step, which matches the I/O concepts of the requested process (R) against those of the candidate SWSs (A) and computes a total score of the maximum similarity matching results. Additionally, it applies the PET to filter the suitable processes returned after the SMS step. The PET step is a second filtering mechanism that provides a checking function that eliminates unrelated web services from the result set of the SMS. The BSA retrieves the suitable SWSs returned in the result set of the PET matching and then parses their QoS metric values using their QoS ontology files to perform the MQP (refs. 9–11 in Figure 9). Finally, the agent displays all the matching web services to the client

as a results page (ref. 12 in Figure 9). In the workflow described above, Steps 9 and 11 deal explicitly with SMS, PET and MQP issues. The client requests the establishment of the binding by specifying a reference to a web service interface that also contains the required QoS.

In conclusion, the BSA incorporates Semantic Web technologies such as OWL and OWL-S ontology languages. Lack of semantic descriptions in the meaning of the I/O/P/E parameters of processes makes it impossible to understand the relations among processes, such as whether processes are equivalent. Thus, an agent cannot dynamically select atomic business processes and semantically discover them to accomplish the required process without human assistance.

5 Semantic matching step

As mentioned in Section 4, domain selection of the broker agent is done in step (i). For example, the client's selected domain ontology is financial investment (Financial_Investment.owl) in the case study (see Figure 1). Initially, the client enters I/O/P/E terms and then, the given I/O parameters are used to find similar task instance(s) in the TO, where the selected task instances have the same or similar I/O/P/E parameters and are the closest matching task(s) to the client goal process. Classes, subclasses, and properties of the task instances of a particular domain are expressed in the TO (as described in Section 2). Thus, the client is able to select the most similar one (let the requested task be R) from the similar task instances returned. This is necessary to distinguish processes according to their functional semantics, since different functions may have the same parameters. For example, the BSA needs to understand which process is required by the client: buyBond (Company_Symbol, Price) or sellBond (Company_Symbol, Price).

Additionally, the BSA takes into account the relationship between terms (such as is_a, synonym, etc.) of the selected domain before initiating SMS execution. So it parses all the process:parameterType and rdf:ID (I/O) concepts of the requested task instance R. It is required to parse the classes, super/subclasses, and all the is_a and synonym properties of the corresponding concepts in the selected domain ontology (e.g. Financial_Investment.owl). Besides this, all OWL-S advertisements of the candidate SWSs (let the set of SWSs be A) are stored in the PO knowledgebase as previously mentioned. Then, the agent accesses the PO and also parses the process:parameterType and rdf:ID (I/O) concepts of all available OWL-S advertisements of the candidate SWSs.

The inputs to the SMS are the requested process, R, and the set of OWL-S advertisements (or the semantic descriptions of atomic processes), A, of the candidate SWSs. The output is a set of matched OWL-S advertisements (of candidate SWSs) sorted according to similarity score.

The SMS focuses on these processes, which are symbolized as R and A above. The SMS uses the matching scores of the I/O concepts of these two processes, where Dissimilar=0, Subsume=0.5, Plugin=0.75, and Exact=1. The four degrees of similarity are related as follows: Dissimilar<Subsume<Plugin<Exact.

The SMS is explained below through three sub algorithms (Algorithm 1, Algorithm 2 and Algorithm 3), but first some of the logic symbols are defined in Table 1.

R_{I} : The set of input concepts of the focused task R.

R_{O} : The set of output concepts of the focused task R.

A_{I} : The set of input concepts of an OWL-S process, A (in the candidate SWSs).

A_{O} : The set of output concepts of an OWL-S process, A, (in the candidate SWSs).

D : The set of all concepts/classes defined in the selected domain ontology, the concepts ontology (CO) and tasks ontology (TO).

Exact relation: A 1-1 mapping $R_{\mathrm{I}} \rightarrow A_{\mathrm{I}}$ where $\forall x_{R_i} \in R_{\mathrm{I}}$ and $\exists x_{A_i} \in A_{\mathrm{I}}$ such that Concepts$(x_{R_i}) \equiv$ Concepts(x_{A_i}): any two focused concepts (R_i and A_i) of both processes with the same or equivalent concepts or having a hasSyn() relation. The same relation must hold on the output terms as well.

• For instance, if (R_i ="Bond" and A_i ="Bond") or (R_i = "Bond" and A_i ="Bond_Certificate") or (R_i ="Bond_Certificate" and A_i ="Bond") then an exact relation exists, giving a score of 1.

Plugin relation: A 1-1 mapping from $R_{\mathrm{I}} \rightarrow A_{\mathrm{I}}$ where $\forall x_{R_i} \in R_{\mathrm{I}}$ and $\exists x_{A_i} \in A_{\mathrm{I}}$ such that Concepts (x_{R_i}) $\subset$Concepts(x_{A_i}): if any focused input concept (R_i) of the process R is a subset or subclass of the concept

Table 1 Table of various logic symbols (Y and Z are lists, each containing a set of concepts)

Logic symbol	Description	Formulation
$\subset$	Proper subset	$Y \subset Z$ means $Y \subseteq Z$ but $Y \neq Z$.
$\supset$	Proper superset	$Y \supset Z$ means $Y \supseteq Z$ but $Y \neq Z$.
$\models$	Entailment	$Y \models Z$ means the concepts/process Y canonically entails the concepts/process Z, which means in every model in which Y is true, Z is also true.
Concepts()	Also known as classes	Main entities of ontology. For example, the class 'Bond' in the Financial Investment ontology.
hasSyn(Y) $\equiv Z$	hasSynonym property	There exists a synonym concept of Y, which is Z; e.g., Bond's synonym is Bond Certificate or Stock.
hasIs_a(Y)	hasIs_a property	There exists an is_a concept of Y; e.g., Government Bond is a Bond.

of process A's input (A_i), or has a hasSyn() or hasIs_a() property, then a plugin relationship exists. At least the same relation must hold on the output terms as well.

• For instance, if (R_i = "Goverment_Bond" and A_i ="Bond") or (R_i ="Goverment_Bond_Certificate" and A_i ="Bond") or (R_i ="Goverment_Bond" and A_i ="Bond_Certificate") etc., then a plugin relation exists, giving a score of 0.75.

Subsume relation: A 1-1 mapping from $R_{\mathrm{I}} \rightarrow A_{\mathrm{I}}$ where $\forall x_{R_i} \in R_{\mathrm{I}}$ and $\exists x_{A_i} \in A_{\mathrm{I}}$ such that Concepts(x_{R_i}) $\supset$ Concepts(x_{A_i}): if any focused input concept (R_i) of the process R is a superclass of the concept of process A's input (A_i), or has a hasSyn(), or hasIs_a() property, then a subsume relationship exists. At least the same relation must hold on the output terms as well.

• For instance, if (R_i ="Bond" and A_i ="Goverment_Bond") or (R_i ="Bond_Certificate" and A_i = "Goverment_Bond") or (R_i ="Bond" and A_i ="Goverment_Bond_Certificate") etc., then a subsume relation exists, with a score of 0.5.

Dissimilar relation: A 1-1 mapping from $R_{\mathrm{I}} \rightarrow A_{\mathrm{I}}$ where $\forall x_{R_i} \in R_{\mathrm{I}}$ and $\exists x_{A_i} \in A_{\mathrm{I}}$ and Concepts(x_{R_i}) $\neq$ Concepts(x_{A_i}): if there is no relation between the inputs of the two processes, a dissimilar relationship exists.

• If (R_i ="Bond" and A_i ="Bondage") then a dissimilar relation exists, which is scored as 0.

Clearly the following relations hold among the I/O concepts of these processes $R_{\mathrm{I}} \subseteq D, R_{\mathrm{O}} \subseteq D, A_{\mathrm{I}} \subseteq D, A_{\mathrm{O}} \subseteq D$.

Calculating the degree of similarity of any two concepts in the I/O parameters of on-focus processes R and A is defined in Algorithm 1. All the relationship types are applied similarly to output terms of the processes being matched.

Algorithm 1 degreeOfProcessMatching (Concept R_i, Concept A_i)

1 **if** (($R_i \equiv A_i$) **or** (hasSyn(R_i) $\equiv A_i$) **or** ($R_i \equiv$hasSyn(A_i))) **then** return rel=**Exact**;

2 **if** (($R_i \subset A_i$) **or** (hasSyn(R_i) $\subset A_i$) **or** ($R_i \subset$hasSyn(A_i)) **or** (hasIs_a(R_i) $\equiv A_i$)) **then** return rel=**Plugin**;

3 **if** (($R_i \supset A_i$) **or** (hasSyn(R_i) $\supset A_i$) **or** ($R_i \supset$hasSyn(A_i)) **or** ($R_i \equiv$hasIs_a(A_i))) **then** return rel=**Subsume**;

4 **if** (($R_i \neq A_i$) **or** (hasSyn(R_i) $\neq A_i$) **or** ($R_i \neq$hasSyn(A_i))) **then** return rel=**Dissimilar**;

(Finding the degree of a match during the SMS, where R_i and A_i denote the input and output concepts, respectively, of two on-focus processes being tested for similarity. This algorithm compares input concepts to input concepts and output concepts to output concepts.)

As shown in Algorithm 2 search(R) takes as input the requested task R and iterates over every A in the PO repository to determine a match. R and A match if their outputs and inputs both match. The algorithm finds a set of matching advertisements with the similarity scores of the match, and returns a resulting list of processes after completing the PET depicted in line 9 in Algorithm 2.

As shown in Algorithm 3, getRelation takes two lists of input (or output) concepts of the $R_{\mathrm{I/O}}$ and a process $A_{\mathrm{I/O}}$ to find the total similarity score by the SMS. The first parameter is a list of input (or output) concepts of the requested task ($R_{\mathrm{I/O}} = \{R_1, R_2, \ldots, R_m\}$) while the second is a list of input (or output)

concepts of a Web service ($A_{\mathrm{I/O}} = \{A_1, A_2, \ldots, A_n\}$). During the SMS, the process:parameterType property or rdf:ID (I/O) concepts of the processes are analyzed by recovering their semantics, that is, meaning, similarities, differences, and relations, while carrying out Algorithm 1. Similarly, semantic distances of concepts, which offer similarity information between concepts, can be given by the ontology developer during the development phase.

Algorithm 2 search(R)
1 ResultList=null;
2 **for** each A in **PO KB do** – – for each process in PO
3 inMatchScore=getRelation(R_{in}, A_{in})
4 outMatchScore=getRelation(R_{out}, A_{out})
5 **if** (inMatchScore = or outMatchScore=0) **then goto** Step 2 and **take** next A
6 **else** ResultList ← (A inMatchScore, outMatchScore);
7 **end if**
8 **end for**
9 **return** (**PET**(R ResultList))

Algorithm 3 getRelation (RList, AList)
1 List3=null; res_score=0; n_R =size(RList); n_A =size(AList);
2 **for** $j = 1$ to n_A **do** – for each concept A_j in AList
3 **for** $i = 1$ to n_R **do** – for each concept R_i in RList
4 score=0;
5 degree=degreeOfProcessMatching (R_i, A_j)
6 **if** (degree≠Dissimilar) **then**
7 **if** (degree=Exact) **then**
score=1.00*d_{weight} (R_i, A_j)/distance(R_i, A_j); **end if**
8 **if** (degree=Plugin) **then**
score=0.75*d_{weight} (R_i, A_j)/distance(R_i, A_j); **end if**
9 **if** (degree=Subsume) **then**
score=0.5*d_{weight} (R_i, A_j)/distance(R_i, A_j); **end if**
10 **else** score=0;
11 **end if**
12 List3← score;
13 **end for**
14 score←Max(List3); – – returns max scored matched concepts from RList & AList
15 **if** (score≠0) **then**
16 **Find** int k ←index(Max(List3)); – – returns index of max scored matched concepts from the lists
17 DeleteRow(R_k); DeleteRow(A_j); – – delete concepts from RList & AList
18 $n_R \leftarrow n_R - 1; n_A \leftarrow n_A - 1$; – – decrease sizes of the AList & RList
19 $i \leftarrow 1; j \leftarrow 1$; – – start the for loops again for the next concept
20 Calculate res_score= res_score+score;
21 List3←null; **goto** Step 2, take next A_j
22 **else**
23 Calculate res_score= res_score+0; – – **goto** Step2, for next A_j
24 **end if**
25 **end for**
26 **if** (res_score= 0) **then** no matching exists; **return** 0;
27 **else return**{res_score=res_score/((size(AList)/size(RList))*d_W) }; – – d_W: Jaro_Winkler_Distance
28 **end if**

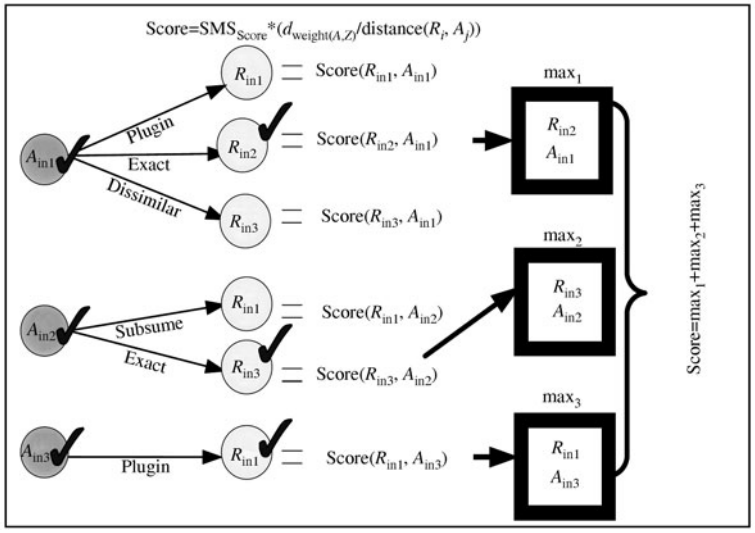

Figure 10 A simple example using Algorithm 3.

$$d_{\text{weight}A} = \frac{1}{\#\text{ofSubconceptsof}A}. \tag{1}$$

If semantic distances are not scored by the developer, all direct subconcepts of a parent concept will have the same distance weight (Bener et al. [1,34]) according to Eq (1).

The BSA finds the semantic distance weight $d_{\text{weight}(A,Z)}$ between any two concepts A and Z in a particular domain ontology as given by Eq. (2).

$$d_{\text{weight}(A,Z)} = d_{\text{weight}(A,B)} * d_{\text{weight}(B,C)} * \cdots * d_{\text{weight}(W,Y)} * d_{\text{weight}(Y,Z)}. \tag{2}$$

The applied SMS scoring method is a simple multiplicative weighting function. A given request R with input concepts $R_{\text{I}} = \{R_1, R_2, \ldots, R_m\}$ and a service A with input concepts $A_I = \{A_1, A_2, \ldots, A_m\}$ are matched and the total score $\text{Similarity}_{\text{Score}_{(R_{\text{I}}, A_{\text{I}})}}$ is calculated according to Eq. (3).

$$\begin{aligned}&\text{Similarity}_{\text{Score}_{(R_{\text{I}}, A_{\text{I}})}}\\ &\equiv \frac{\sum_{j=1}^{\max(n_{R_{\text{I}}}, n_{A_{\text{I}}})} \left(\max_{j,i} \left\{ \prod_{i=1}^{\max(n_{R_{\text{I}}}, n_{A_{\text{I}}})} \text{SMS}_{\text{Score}_{(R_i, A_j)}} * \left(\frac{d_{\text{weight}_{(R_i, A_j)}}}{\text{distance}(R_i, A_j)} \right) \right\}, \text{delete}R_i \& A_j \right)}{(n_{A_{\text{I}}}/n_{R_{\text{I}}}) * d_{\text{W}}},\\ &n_{R_{\text{I}}} = \text{size}(R_{\text{I}});\ \text{and}\ n_{A_{\text{I}}} = \text{size}(A_{\text{I}}); d_{\text{W}} = \text{Jaro} - \text{Winkler distance}.\end{aligned} \tag{3}$$

The above equation considers only input lists $\{R_{\text{I}}, A_{\text{I}}\}$ or $\{R_{\text{in}}, A_{\text{in}}\}$ of the R and A processes. Output lists of the R and A processes are similarly compared. In Eq (3), distance(R_i, A_j) shows the number of concepts (levels) between any two focused concepts, R_i and A_j, in the ontology.

Assume that the Bond concept has two sub-concepts, Goverment_Bond and Company_Bond. If the R_i =Goverment_Bond ith input concept of R and the A_j =Bond jth input concept of A are considered, then the $\text{SMS}_{\text{Score}_{(R_i, A_j)}} * d_{\text{weight}(R_i, A_j)}$ of the (Goverment_Bond, Bond) match is 0.75*0.5 = 0.375. Because (Goverment_Bond, Bond) has a Plugin relation, the subsumption score is 0.75 according to Algorithm 1 and the weight is 0.5 by Eq. (1). Moreover, there exists a subclass relation between (Goverment_Bond, Bond) and thus, {distance (Goverment_Bond, Bond)} is 1. The concepts used above are defined in the Financial_Investment ontology (Figure 1).

Figure 10 illustrates the basic step in the getRelation function by means of a simple example. The solid lines indicate the relationships inferred by the Pellet reasoner used in [27].

For example, the input sets of $R_{\text{in}} = \{R_1, R_2, R_3\}$ and $A_{\text{in}} = \{A_1, A_2, A_3\}$ are executed by the getRelation ($R\text{List}_{\text{I/O}} A\text{List}_{\text{I/O}}$) in the SMS. Pseudocode for the getRelation function of Algorithm 3 is given above.

The algorithm first attempts to compute a maximum match for A_{in1} (as depicted in Figure 10). Assume that the following matches are inferred, then the values of Score=$\text{SMS}_{\text{Score}}$*($d_{\text{weight}(A,Z)}$/distance($R_i, A_j$)) are calculated by dividing each parameter of R_{in} by the distance (R_i, A_j) value:

A_{in1} Subsumes R_{in1} →Plugin; A_{in1} is a Synonym of R_{in2} → Exact; A_{in1} does not match R_{in3} → Dissimilar; A_{in1} has a max match with R_{in2}. Thus, R_{in2} and A_{in1} are removed from their respective lists. The algorithm then attempts to match the next concept, A_{in2}. Assume that the following matches are inferred: R_{in1} Subsumes A_{in2} →Subsume; A_{in2} is a Synonym of R_{in3} →Exact.

Thus, A_{in2} is matched with R_{in3} and R_{in3} and A_{in2} are removed from their respective lists. The algorithm then attempts to find a match for A_{in3}. Assume that the following match is inferred for the last parameter in R_{in}: A_{in3} Subsumes R_{in1} →Plugin.

If the sizes of the R_{I} and A_{I} are not equal, we insert |size(R_{I})−size(A_{I})| null values into the smaller list. Thus, the function calculates the maximum values for each iteration and then finds their sum.

Note that Eq. (3) also takes into account the overall distance using the Jaro-Winkler distance [28, 29] and also the ratio of input concept sizes of the R and A processes to complete the calculation of the average $\text{Similarity}_{\text{Score}_{(R_{\text{I}},A_{\text{I}})}}$ for the given ($R_{\text{I}}, A_{\text{I}}$) query. The same equations and operations are also applied to the output lists $\{R_{\text{O}}, A_{\text{O}}\}$ of the R and A processes.

One of the key differences between syntactic/lexical and semantic matching is that in syntactic matching, when two concepts are matched we consider only the labels attached to them, independently of the position of the concepts in the ontology graph. On the other hand, when matching two concepts in semantic matching, the concepts we analyze depend not only on the concept attached to the node, but also on the position of the concept in the ontology.

Furthermore, we also consider the Jaro-Winkler distance, which was designed and is best suited to matching strings to measure the syntactic similarity between two terms. The Jaro-Winkler distance eliminates duplicate records in database tables and also normalizes the score, such that 0 denotes dissimilarity and 1 is an exact match. Therefore, the Jaro distance d_J is modified for use with the lists of I/O concepts of the R and A processes instead of being applied to two given strings. For semi-semantic based input matching among concepts, the formula based on the Jaro-Winkler distance approach is modified as given below:

$$d_{\text{J}} = \frac{1}{2}\left[\frac{m}{n_{R_{\text{I}}}} + \frac{m}{n_{A_{\text{I}}}}\right], \quad n_{R_{\text{I}}} = \text{size}(R_{\text{I}});\ n_{A_{\text{I}}} = \text{size}(A_{\text{I}}); \tag{4}$$

• m is a value based on the number of matching concepts multiplied by their particular degrees, for any theoretic relation including $\{\equiv, \cap, \subset, \supset\}$, but excluding Dissimilar $\{\neq\}$, where m = (#ofExact*1) + (#ofPlugin*0.75) + (#ofSubsume*0.5).

• l gives the length of the found relations of all the concepts.

• p is the constant scaling factor denoting how much the score has been adjusted upwards for common concepts. The standard value for this constant in Winkler's work is $p = 0.1$.

• The Winkler distance d_{W} is:

$$d_{\text{W}} = d_{\text{J}} + (l \cdot p \cdot (1 - d_{\text{J}})), \tag{5}$$

where d_{J} is the Jaro distance for all the concepts of R and A.

Note that Eq. (4) always yields a value between 0 and 1, since the total number of correspondences in m cannot be greater than the average size of the two sets R and A. Assume three given input lists, a client request (R) and two OWL-S advertisements (A_1 and A_2) in the candidate SWSs listed below

Request (R): Ordinary-Publisher, Novel, Paper-Back;

Service 1 (A_1): Publisher, ScienceFictionBook;

Service 2 (A_2): Book, Alternative-Publisher, Book-Type.

For service A_1: m=2 since there are 2 matched concepts, namely; {Publisher Subsumes Ordinary-Publisher → Plugin}; {ScienceFictionBook Subsumed by Novel → Subsume}; $n_{R_{\text{I}}} = 3$ and $n_{A_{\text{I}}} = 2$.

We calculate a Jaro score of:

$$d_{\text{J}} = \frac{1}{2}\left[\frac{0*1 + 1*0.75 + 1*0.5}{3} + \frac{0*1 + 1*0.75 + 1*0.5}{2}\right] = 0.5208.$$

```
ProcessEquivalenceTask (R_I/O, A_I/O)
1 if (R_I ≡ A_I) then
2     if (R_O ≡ A_O) then R ≡ A– –R ≡ A means is 'equivalent' and returns true
3     else if (R_O ⊂ A_O) then R ≡ A
4     else R! ≡ A–R! ≡ A means is 'not equivalent' and returns false
5     end if
6 else if (R_I ⊃ A_I) then
7     if (R_O ≡ A_O) then R ≡ A
8     else if (R_O ⊂ A_O) then R ≡ A
9     else R! ≡ A
10    end if
11 else R! ≡ A
12 end if
```

Figure 11 Process equivalence algorithm.

The Jaro-Winkler score using the standard weight $p = 0.1$ is given by $d_{\mathrm{W}} = 0.5208 + (2 * 0.1(1 - 0.5208)) = 0.6166$.

For service A_2: $m = 3$ since there are 3 matching concepts, namely; {Alternative-Publisher Subsumed by Ordinary-Publisher→ Subsume}; {Book Subsumes of Novel → Plugin}; {Book-Type Subsumes of Paper-Back → Plugin}; $n_{R_{\mathrm{I}}}$=3 and $n_{A_{\mathrm{I}}}$=3.

We calculate a Jaro score of

$$d_{\mathrm{J}} = \frac{1}{2}\left[\frac{0 * 1 + 2 * 0.75 + 1 * 0.5}{3} + \frac{0 * 1 + 2 * 0.75 + 1 * 0.5}{3}\right] = 0.6666.$$

The Jaro-Winkler score using the standard weight $p = 0.1$ is given by $d_{\mathrm{W}} = 0.6666 + (3 * 0.1(1 - -0.6666)) = 0.7666$.

Finally, note that Eq. (3) considers the overall distance via the Jaro-Winkler distance (d_{W}) by multiplying by the ratio of the I/O concept sizes of the R and A processes to compute the final average $\mathrm{Similarity}_{\mathrm{Score}_{(R_{\mathrm{I}}, A_{\mathrm{I}})}}$ for the given $(R_{\mathrm{I}}, A_{\mathrm{I}})$ query.

6 Process equivalence task

After the SMS step, the BSA performs a PET between the request process R and a similar found process A which is in the ResultList of the retrieved appropriate processes (see Algorithm 2, line 9) to identify whether these two processes have the same purpose (previously discussed in (iii) of Section 4). The BSA applies the PET only to the retrieved processes in the ResultList with the same concepts as R. Exact and hasSynonym relations are considered in the PET, which is carried out according to the algorithm given in Figure 11.

Let us trace the algorithm through an example. Assume that the I/O parameters of the request process R, and a process A in the ResultList are denoted, respectively, as follows: $R = \{R_{\mathrm{I}}, R_{\mathrm{O}}\}$ and $A = \{A_{\mathrm{I}}, A_{\mathrm{O}}\}$. Assume that: $R_{\mathrm{I}} \equiv$ {Bond[http://../ontology/Concepts.owl#Bond]} and, $R_{\mathrm{O}} \equiv$ {BondPrice[http://../ontology/Concepts.owl#Price]}; $A_{\mathrm{I}} \equiv$ {BondCertificate[http://../ontology/Concepts.owl#Bond]} and, $A_{\mathrm{O}} \equiv$ {Price[http://../ontology/Concepts.owl#Price]}.

Then, it is clear that $R_{\mathrm{I}} \equiv A_{\mathrm{I}}$ and $R_{\mathrm{O}} \equiv A_{\mathrm{O}}$, consequently A is equivalent to R according to their parameter types that are given in brackets "[]". In another case, if $R_{\mathrm{I}} \equiv A_{\mathrm{I}}$, but $R_{\mathrm{O}} \equiv$ {BondPrice[http://../ont-ology/Concepts.owl#Price]} and $A_{\mathrm{O}} \equiv$ {BondID[http://../ontology/Concepts.owl#ID], Price[http://../ ontology/Concepts.owl#Price]}, that is, $R_{\mathrm{O}} \subset A_{\mathrm{O}}$, then it can be stated that A subsumes the R process.

Finding the extra outputs (such as BondID given above) is acceptable, and therefore it can be concluded that A is equivalent to R. Input matching fails since A does not satisfy the client request R. In the latter

case, the BSA checks whether any unnecessary I/O is retrieved, and if so, stipulates whether it can be ignored in the above example.

Another scenario is that extra inputs in A_{I} may be unknown input parameters to the client if the R_{I} do not include these input parameters. Thus, the unknown input parameters could cause difficulty during the execution of a well scored process A, causing the scoring of the matching mechanism to be directed in the wrong way. The reason for this is that the matching mechanism could offer a smaller score to another better suited process than this type of process if the former has an exact relation. As an example, consider the input concepts of a request process and two services given, respectively, as $R_{\mathrm{I}} = \{a, b, c\}$, $S1_{\mathrm{I}} = \{a, b, c, d\}$, and $S2_{\mathrm{I}} = \{a', b', c\}$.

Assume that the output lists of the services are exactly the same as that of the requested process and their similarity scores are: $S1_{\mathrm{I}}\&R_{\mathrm{I}} = 3$ and $S2_{\mathrm{I}}\&R_{\mathrm{I}} = 2.5$. Here $S1_{\mathrm{I}}$ contains three Exact relations and $S2_{\mathrm{I}}$ two Plugin and one Exact relation. Consequently, $S1$ is found to be the most similar service since it has three exact parameters: 'a', 'b', and 'c'.

In fact, $S1$ would be unsuitable since it contains an extra parameter 'd' that may be unknown to the client. Therefore, it is necessary to take into account such cases during the final calculation of the matching scores. So, $S1_{\mathrm{I}}$ is considered by the PET to specify its final score. All other possible conditions of process equivalence are given in Figure 11.

Consequently, the BSA uses the scores ($\mathrm{SimilarityScore_{in}}$, $\mathrm{SimilarityScore_{out}}$) from the SMS and then multiplies them with the corresponding values of ($\mathrm{PETin_{Score}}$, $\mathrm{PETout_{Score}}$) from the PET (line 5 of Algorithm 4) before calling the Matching of QoS procedure (line 8 of Algorithm 4).

Algorithm 4 PET(R, ResultList)

1 **for** each A in ResultList **do**

2 $\mathrm{PETin_{Score}}$=ProcessEquivalenceTask($\mathrm{R_{in}}$, $\mathrm{A_{in}}$);– – see Figure 5

3 $\mathrm{PETout_{Score}}$ =ProcessEquivalenceTask(R_{out}, A_{out});– – Figure 5

4 **if**($R\neg \equiv A$)=true **then goto** Step1 and **take** next A

5 **else** ResultList ← (A, $\mathrm{SimilarityScore_{in}}$* $\mathrm{PETin_{Score}}$, $\mathrm{SimilarityScore_{out}}$ * $\mathrm{PETout_{Score}}$);

6 **end if**

7 **end for**

8 ResultList←(Matching of QoS (R_{QoS}, $\mathrm{ResultList_{QoS}}$))

9 return (ResultList)

The PET score also depends on the sizes of R and A, with their ratio, n, defined as in Eq. (6).

$$\begin{aligned} &n_{R_{\mathrm{I/O}}} = \mathrm{size}(R_{\mathrm{I/O}}) \text{ and } n_{A_{\mathrm{I/O}}} = \mathrm{size}(A_{\mathrm{I/O}}),\\ &n = \mathrm{Ratio}_{\mathrm{OfSizes}^{(\mathrm{I/O})}} = n_{R_{\mathrm{I/O}}}/n_{A_{\mathrm{I/O}}}. \end{aligned} \tag{6}$$

Eq. (7) defines the multiplication and final similarity score after the PET step.

$$\mathrm{Similarity}_{\mathrm{Score}^{(\mathrm{I/O})}} = \mathrm{Similarity}_{\mathrm{Score}^{(\mathrm{I/O})}} * \mathrm{PET}_{\mathrm{Score}^{(\mathrm{I/O})}}. \tag{7}$$

Parameters for $\mathrm{PETin_{Score}}$ and $\mathrm{PETout_{Score}}$ are given as shown in Table 2.

The value '1' does not affect the order of services in the ResultList after the SMS. However, according to our example, the similarity scores of $S1$ are multiplied by $\mathrm{PETin_{Score}} = n$, since $R_{\mathrm{I}} \subset S1_{\mathrm{I}}$ (see the 4th row and 2nd column of Table 2) according to the PET, that is, $S1_{\mathrm{I}}\&R_{\mathrm{I}}$=3*(3/4) =2.25. This causes $S1$ to be placed after $S2$ in the ResultList since its score is smaller. Following the SMS and PET, the BSA aims to increase the quality of the returned result set (ResultList) using the predetermined QoS metric information for each returned SWS in the ResultList as introduced in Section 3.

7 Matching of QoS task

The BSA compares the QoS metric values of each found SWS in the ResultList with that of the client request. This process, called MQP, compares the values of the on-focus services with that of the client.

Table 2 Assigned scores for $\text{PETin}_{\text{Score}}$ and $\text{PETout}_{\text{Score}}$

I/O	$R_{\text{O}} \equiv A_{\text{O}}$	$R_{\text{O}} \subset A_{\text{O}}$	$R_{\text{O}} \supset A_{\text{O}}$	Otherwise
$R_i \equiv A_i$	$\text{PETin}_{\text{Score}}$=1, $\text{PETout}_{\text{Score}}$=1	$\text{PETin}_{\text{Score}}$=1, $\text{PETout}_{\text{Score}}$=1	$\text{PETin}_{\text{Score}}$=1, $\text{PETout}_{\text{Score}}$=1/$n$	
$R_i \supset A_i$	$\text{PETin}_{\text{Score}}$=1, $\text{PETout}_{\text{Score}}$=1	$\text{PETin}_{\text{Score}}$=1, $\text{PETout}_{\text{Score}}$=1	$\text{PETin}_{\text{Score}}$=1, $\text{PETout}_{\text{Score}}$=1/$n$	$\text{PETin}_{\text{Score}}$=1, $\text{PETout}_{\text{Score}}$=1
$R_i \subset A_i$	$\text{PETin}_{\text{Score}} = n$, $\text{PETout}_{\text{Score}}$=1	$\text{PETin}_{\text{Score}} = n$, $\text{PETout}_{\text{Score}}$=1	$\text{PETin}_{\text{Score}} = n$, $\text{PETout}_{\text{Score}}$=1/$n$	
Otherwise	$\text{PETin}_{\text{Score}}$=1, $\text{PETout}_{\text{Score}}$=1			

The role of the QoS step is to find the degree of quality ($\text{DegreeOfQoS}_{\text{Score}}$) of the filtered candidate Web services after completing the SMS and PET through their QoS metric information and that of the client. The QoS metric information for each service is provided by its provider in its QoS ontology file. Thus, the BSA is able to calculate the final score ($\text{Total}_{\text{Score}}$) as a simple multiplication of $\text{Similarity}_{\text{Score}}$ (from the PET) and $\text{DegreeOfQoS}_{\text{Score}}$ (from the MQP) using Eq. (8).

$$\text{Total}_{\text{Score}} = \text{Similarity}_{\text{Score}} * \text{DegreeOfQoS}_{\text{Score}}. \tag{8}$$

The total scored list of web services is then ordered (in descending order according to $\text{Total}_{\text{Score}}$) and presented to the user as a results page. To-date, we have implemented two scoring modules: SMS and PET scoring. We were unable to implement the MQP scoring module owing to the unavailability of an OWL-S based SWS test collection with QoS parameters. However, the SMS and PET evaluation results for the test runs executed on the prototype system we implemented, are presented in Section 9.

8 Related works

Web services use certain standards to communicate between machines and have interfaces that are described in a machine readable format. The WSDL [30] is an XML-based language used to describe Web services. It provides an easy means for service suppliers to explain the basic format of requests to their systems, regardless of the underlying access protocol. UDDI [31], which stands for Universal Description, Discovery and Integration, is a platform-independent registry for web services around the world. In addition, the SOAP protocol [32] provides for the exchange of parameters and results through XML-based messages over computer networks using HTTP [33].

DAML+OIL is a markup language for the Semantic Web [34]. Its revision, OWL [16] has been developed by the W3C as an international standard. The extension-language of OWL, OWL-S [17,35], is used to describe the semantics of web services (i.e., Semantic Web services), supporting automated service discovery, invocation, and composition.

In this section, we discuss recent research on the discovery of SWSs based on semantic matching methods. Several semantic matching algorithms are based on the matching of I/O/P/E values in the Service Profiles or Process Model. One such algorithm, which was initially proposed by Paolucci et al. [1,7,8,11–13], has been cited extensively in recent literature and several expanded approaches [2–14] are based thereon. Bener et al. [1,3,4,10] discussed enhancing the well-known matching algorithm according to the depth of the captured semantic similarity. They also introduced a new approach [3,4] that considers semantic distances based on subsumption-based similarity, property-level similarity, similarity distance annotations and WordNet-based similarity. Instead of classifying the candidate web services on a discrete scale, the matching agent applies a scoring scheme to rank candidate web services according to their relevance to the request.

In our study, the BSA considers the semantic distance based on subsumption-based similarity, similarity (hasSynonym, hasIsa) annotations, PET similarity and QoS-based similarity scoring instead of property-level similarity, similarity distance annotations, and WordNet similarity.

Subsumption-based similarity is performed on rdf:ID (I/O) concepts of candidate SWSs while also considering the ParameterTypes of the concepts during the matching phase. The reason for this has been explained in detail in the previous sections.

The (hasSynonym, hasIsa) annotations are based on the is_a and synonym properties between the concepts. The hasSynonym property of an on-focus concept signifies that there exists an exact relation with another concept. The hasIsa property denotes that there exists a plugin or subsume relation between concepts. Although WordNet is used in several approaches [2,3,4,36] and also in our previous studies [37–42], we do not use WordNet-based similarity in the BSA since focused concepts may not be contained within the limited database of WordNet.

Bener et al. argued that in matching, it is important to include both properties and their associated range in measuring the degree of the match, so that, if two concepts have similar properties (properties with a subclass relation) and their range classes are similar, this improves their level of similarity [3,4]. Thus, this provides property-level similarity which is very useful for I/O/P/E matching via ontologies. Additionally, the method is also applicable to currently cited partitioned-based similarity matching architectures (such as the BSA) to improve the rankings of SWSs during the discovery process.

Ilhan et al. [3,4] and Şenvar et al [10] created their own similarity distance ontologies to find the distance between objects. However, a similarity distance ontology is supposed to contain proper similarity scores through the assignment of concept-similarity ratings of all the concepts in the ontology by the ontologist or a similarity ranking mechanism. Additionally, there are some critical points that should be considered by the ontologists or the ranking system when assigning similarity scores between concepts for such a similarity matching ontology. The critical points may be dependent on some statistical information of frequently consumed concepts for a particular domain, or the distance of the relation between concepts and their domain(s). Instead of using such a similarity matching ontology, the similarity distance between two concepts is better calculated using Eqs. (1) and (2) for any ontology since these give equivalent similarity rates for all the concepts that do not change the expected result. In addition, the overall similarity distance, the Jaro-Winkler distance, of the two lists containing the I/O concepts of the R and A processes is calculated using Eqs. (4) and (5) for any ontology. Therefore, we propose a task ontology (TO) that helps to distinguish the functionality of processes and also some domain ontologies with domain specific concepts and relations because some processes may have the same I/O/P/E parameters but different functionalities. Different from SAM, the BSA can understand the service categorization of the requested task through the TO. While carrying out the steps above, it also takes into account the maximum similarity scores like SAM from the captured results through association with its own knowledgebases (TO, Domain and CO).

Another difference from SAM is that we take into account the PET scoring. In finding the total similarity score between a request process (R) and each returned advertisement (A_i) in the result set of SWSs after the SMS, there may be two processes with the same I/O concepts, but with different total similarity scores due to the availability of extra input concept(s) in the A_i or A_j processes. This situation causes one (A_i or A_j) or both to be an unrequested process(es) since the unknown input parameters may exceed the client's knowledge. On the other hand, the lack of expected suitable outputs in any of the returned advertisements (A_i) in the result set of SWSs may not satisfy the client's request.

As a final step in the BSA, unsuitable candidate SWSs are eliminated according to the required QoS parameters of the client. This step allows retrieval of better quality SWSs in the result set. Thus, through the use of the SMS, PET, and MQP algorithms, the chance of missing suitable processes is reduced in the discovery task.

Currently, we have implemented only two of the scoring modules: the SMS and PET. Additionally, task and domain ontologies have been developed and used during the SMS. The Pellet reasoner [27] is used to obtain the associations between the concepts during inferencing through the TO, Domain, and CO ontologies. The BSA was developed in Java with JBuilder 2006. Java was chosen for the implementation owing to its power in the open source community and existing projects on the Semantic Web.

Table 3 gives a featurewise comparison of some recent matchmaker architectures [3,4] augmented with the values for the BSA. In addition, the criteria rows for (hasSynonym, hasIsA) annotations, process

Table 3 Comparison of BSA and other matchmaker algorithms

Feature/Matchmaker	SAM Ilhan et al. [3,4]	MSMatchmaker Şenvar et al. [10]	Larks Sycara et al. [6,7]	Li and Harrocks et al. [43]	Wu and Wu et al. [36, 44]	OSDL Kuang et al. [44]	BSA
Subsumption-based	+	+	+	+	+	+	+
Service interface	+	+	+	+	+	+	+
Matching constraints	−	−	+	−	−	−	
Property level matching	+	−	−	−	−	+	+
Similarity distance	+	+	−	−	−	−	−
(hasSynonym, hasIsa) annotations	−	−	−	−	−	−	+
Process equivalence matching	−	−	−	−	−	−	+
QoS-based or nonfunctional matching	−	−	−	−	+	+	+
Service category matching	−	+	+	+	+	−	+
WordNet	+	−	−	−	+	−	−
OWL-S	+	−	−	−	−	−	+

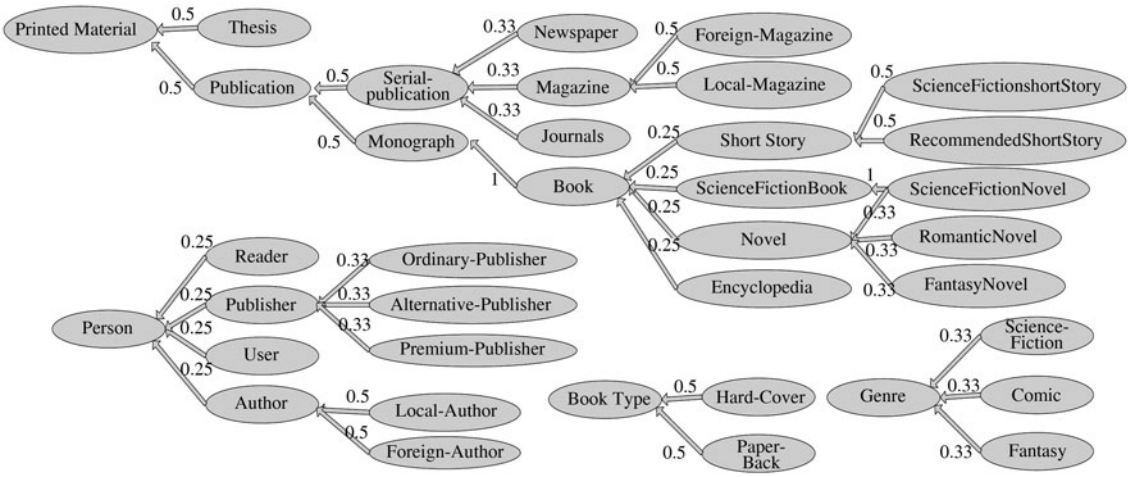

Figure 12 Book, Person and Printed Material ontology section [SAM, 200] showing similarity distances.

equivalence matching, and QoS-based matching have been inserted to better highlight the success of the BSA approach.

9 Evaluation

To evaluate the performance of our proposed BSA algorithm, we compared it with an advanced matchmaker, the SAM architecture of Ilhan et al. [3,4]. We used their extended research on a book ontology in the OWL-S service retrieval test collection (OWL-S TC) available from the SemWebCentral Website[3)] In addition, we also used their modified related request and service definitions [45]. As shown in Figure 12, they added two subclasses of Magazine, namely the Foreign-Magazine and Local-Magazine classes. They also introduced subclasses for the Publisher concept, namely the Author and Newspaper concepts.

Their ontology contains information on printed material classification and related concepts such as authors, publishers, publishing intervals in terms of time and date and several other concepts. We cal-

3) http://projects.semwebcentral.org/frs/?group_id=89

Table 4 Comparison of BSA and SAM based on input/output parameter matching [34][a)]

Service name	Inputs	Outputs	Service score of SAM			SMS scores of BSA		
			Input scores	Output scores	Total scores	Input scores	Output scores	Total scores
Request	Ordinary-Publisher, Novel, Paper-Back	Local-Author, Genre						
Service 1	Publisher, ScienceFictionBook	Author, Price	0.35964(2)	0.12229(3)	0.21723	0.640(2)	0.8571(2)	0.7485(2)
Service 2	Book, Alternative-Publisher, Book-Type	Publisher, Price, Date	0.4388(1)	*0.01447(4)	0.27771	0.77(1)	0.012(5)	0.391(5)
Service 3	FantasyNovel, Author	Price, Comic	0.18026 (5)	*0.17033 (2)	0.08078	0.47 (5)	0.5076 (4)	0.4888(4)
Service 4	Newspaper, Book-Type, Person	Review, Fantasy	0.23636 (4)	*0.12229 (3)	0.69465	0.5321(4)	0.5076(3)	0.5198 (3)
Service 5	Publication, Book-Type, Reader	Genre, r Publishe	0.31718 (3)	*1.00018 (1)	0.20024	0.575 (3)	1.2565 (1)	0.9157 (1)

a)The order of similarity matching scores is given in parenthesis ( ).

*These figures reflect corrected entries. Please see discussion in the text below.

culated the similarity distances using Eq. (1) for the whole path of the ontology as shown above in Figure 12.

The input concepts for the Request process are Ordinary-Publisher, Novel, and Paper-Back, which are given (in the first row and second column) in Table 4. The output parameters for the Request process are Local-Author and Genre (in the first row and third column) in Table 4. To demonstrate the value-added features of our proposed BSA matchmaker, we present a test case between Request and Service 2 for input matching. The input parameters for Service 2 are Book, Alternative-Publisher, and Book-Type (in the Service 2 row and second column) in Table 4.

In Table 4, the '*' denotes a corrected value. In the original reference the authors mistakenly swopped Service 2 with Service 3 and also Service 4 with Service 5 in the output matching. For the best understanding of the evaluation results of their case study, we re-arranged and placed them in the correct order in the output matching column. On the other hand, the scores for input matching were given in the correct order by the authors (see the column for Input scores of SAM in Table 3). However, the total scores were calculated with the incorrect outputs. Thus, using the incorrect output scores (see the column for Total scores of SAM in Table 3) makes it difficult to compare the results of SAM and the BSA To better compare the two algorithms, we calculated the correct values according to their system mechanics although we could not ascertain some of the annotated similarity scores of their ontology for focused concepts in their case study. Thus, we consider only the input matching scores of SAM compared with those of the BSA for the same case study.

Figure 13 presents the steps in the calculation of only the SMS score for input matching between Request and Service 2 (using Eqs. (2) to (5)). Service 2 is found to be the most similar to Request since it has the highest score for input matching of all the other classes. Results of the input-output parameter matching of all services in the test case are listed in Table 4. We do not consider the PET scoring here since the authors' case study does not consider exact match cases. This is not suitable for PET since the closest parameters are considered instead of the exact parameters in the SAM example.

Matching of Request and Service 2 gives the highest score based on input matching since it has one subsume and two plugin relations. All applied calculations for the SMS are shown in Figure 13. In fact, both the BSA and SAM found this to be the best matched service in input matching with a score of 0.77 by BSA (ranked as (1) in the input column for the BSA and SAM in Table 3).

$$\{\text{Ordinary-Publisher} \rightarrow \text{Alternative-Publisher}\},$$
$$\{\text{Novel} \rightarrow \text{Book}\},$$

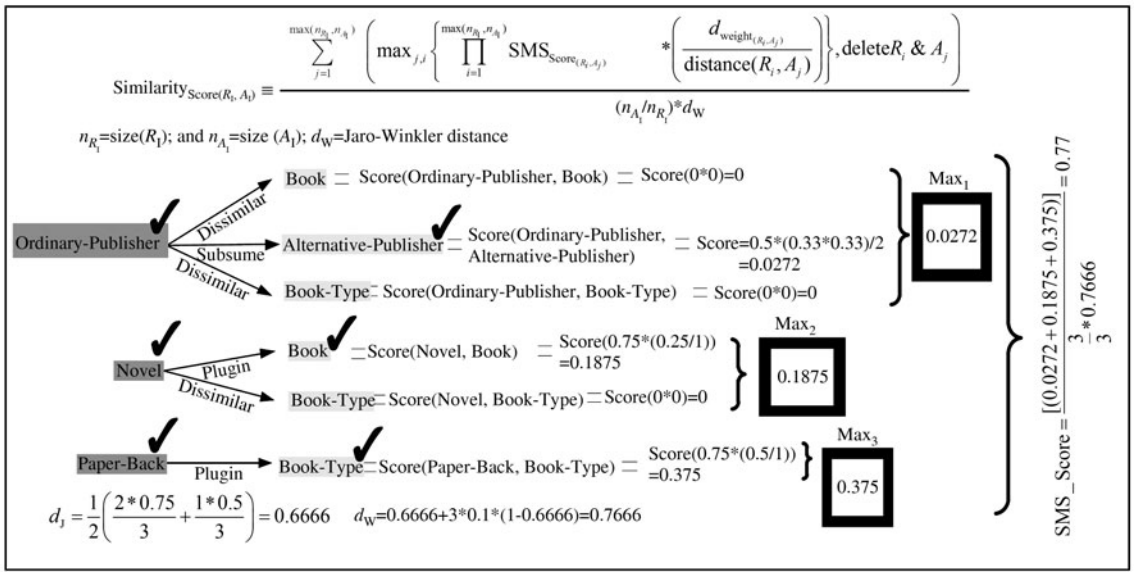

Figure 13 BSA calculations of SMS score for input matching of Request and Service 2.

{Paper-Back → Book-Type}.

Furthermore, the researchers explained that Service 4 has a great score advantage considering the exact match for the Genre parameter in the outputs between Request and Service 4. However there is no such common concept between the outputs of Request and Service 4 since Request contains Local-Author and Genre concepts whereas Service 4 contains Review and Fantasy.

Matching {Local-Author→ Review} yields no similarity and thus its score is zero. Similarly, {Local-Author → Fantasy} has no similarity, resulting in a score of zero as well. The matching mechanism could not find any relation for the Local-Author concept of Request.

Comparing {Genre→Review} yields no similarity, so its score is zero as well. However, {Genre→ Fantasy} has only a subsumption relation and its score is calculated as 0.12229 by the authors (Service 4 row and the output column of SAM in Table 4). The matching mechanism found only a subsumption relation between the Genre and Fantasy concepts in all the matching steps considering Request and Service 4

On the other hand matching Request and Service 5 should give the highest score according to output matching since {Genre→Genre} has an exact relation. The BSA found this to be the best matched service for output matching and scored it as 1.2565 (ranked as (1) in the output column of BSA in Table 4).

Furthermore, the authors mentioned that Service 3 has the weakest match for both inputs and outputs, so this places it last in the rankings. Finally, the authors evaluated the input matching of Service 3 as the weakest service, which was also found as unrelated and scored as 0.47 by the BSA algorithm. However, Service 3 is not the weakest match in output matching and this ranking is probably due to the small mistake made in the output matching of SAM. Both the BSA and SAM found that Service 3 and Service 4 have the same output matching scores as depicted in the output columns of the BSA and SAM in Table 4. Thus, for Service 3 and Service 4, we order the results according to the maximum value of input scores instead of using a random selection as for SAM. In conclusion, the authors found the order of input matching to be:

Service 2 > Service 1 > Service 5 > Service 4 > Service 3.

The BSA also found the same order for input matching. The results reveal that Service 2 is more closely matched to the Request process with respect to input matching. In addition, the BSA found Service 1 to be more related to the request process than either Service 5 or Service 4. Although Service 5 and Service 4 have two plugin and one subsume relations, which is more than the available relations

for Service 1, the similarity of the compared concepts for the Request process and Service 1 is closer (or more related) than that for Service 5 or Service 4. The meaning of similarity here is the similarity distance, which is measured by the number of levels and total weight between the compared concepts. Finally, the BSA found that Service 5 has the highest total score (0.9157) considering both input and output matching (see the column for Total scores of BSA in Table 4).

10 Conclusion

In this article, we introduced the BSA approach to discover appropriate web services. The BSA uses several knowledge-bases to obtain semantic annotations of web services: OWL-S files of candidate SWSs, and the tasks, domain, and concept ontologies. To parse the semantic annotations of web services, we employed some widely used ontology APIs such as Jena and the OWL-S API. The Jena OWL API is used for parsing necessary information from OWL files. Additionally, the OWL-S API is used to parse necessary process and profile models from OWL-S files of the candidate SWSs.

For the evaluation of the BSA approach, an OWL-S service retrieval test collection found on the SemWebCentral website was used. The BSA uses its knowledge-bases while executing the SMS, PET, and MQP algorithms and obtains a total similarity score that expresses the degree of semantic similarity for each process of the candidate SWSs. In calculating the score, each web service in the test collection is compared against the requested process of a client. All required details of the proposed semantic-based similarity scoring mechanism are formulated and presented in the algorithms used in the BSA prototype. Additionally, we presented a financial investment brokerage scenario with three different parties: a customer, service supplier, and broker agent.

In this article, we also presented the results of a comparison of the BSA and some recently published approaches (SAM, MSMatchmaker, Larks, etc.) and preliminary experiments to evaluate the effectiveness of our approach. In the future we aim to work on matching product metadata, such as in supply chain management for example, to find the required information relationships between a broker and a supplier's product metadata using domain specific product ontologies at either end.

Acknowledgements

This work was supported in full by Eastern Mediterranean University (EMU) Scientific Research Project (Grant No. BAP-A-07-20), Turkey. The second author was also employed at EMU.

References

1 Paolucci M. Semantic matching of web service capabilities. In: Horrocks I, Hendler J A, eds. International Semantic Web Conference. London: Springer-Verlag, 2002. 333–347
2 Bener A B, Özadali V, Ilhan E S A. Semantic matchmaker with precondition and effect matching using SWRL. Expert Syst Appl, 2009, 36: 9371–9377
3 Ilhan E S, Bener A B. Improved service ranking and scoring: Semantic advanced matchmaker (SAM) architecture. In: Evaluation of Novel Approaches to Software Engineering (ENASE 2007), Barcelona, 2007
4 Ilhan E S, Akkus G B, Bener A B. SAM: Semantic advanced matchmaker. In: 19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007), Boston, 2007. 698–703
5 Wang H, Li Z. Capability matchmaking of Semantic Web services with preconditions and effects. In: 3rd Chinese Semantic Web Symposium (CSWS 2009), Nanjing, 2009
6 Sycara K, Klusch M, Widoff S, et al. Dynamic service matchmaking among agents in open information environments SIGMOD Rec, 1999, 28: 28–35
7 Sycara K, Paolucci M, Ankolekar A, et al. Automated discovery, interaction and composition of Semantic Web services. J Web Semant , 2003, 1: 27–46
8 Paolucci M, Kawamura T, Payne T R, et al. Importing the Semantic Web in UDDI. In: Horrocks I, Hendler J A, eds. Proceedings of Web Services E-business and Semantic Web. London: Springer-Verlag, 2002. 225–236

9 Rambold M, Kasinger H, Lautenbacher F, et al. Towards autonomic service discovery: A survey and comparison. In: IEEE SCC 2009 International Conference on Services Computing, Bangalore, 2009. 192–201

10 Şenvar M, Bener A B. Matchmaking of Semantic Web services using semantic- distance information. In: Yakhno T, Neubold E, eds. Fourth Biennial International Conference on Advances in Information Systems LNCS4243. Berlin: Springer-Verlag, 2006. 177–186

11 Srinivasan N, Paolucci M, Sycara K. Adding OWL-S to UDDI, implementation and throughput. In: First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), San Diego, 2004

12 Srinivasan N, Paolucci M, Sycara K. An efficient algorithm for OWL-S based semantic search UDDI. In: Cardoso J, Sheth A P, eds. First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC'04). Berlin: Springer-Verlag 2004. 96–110

13 Kawamura T, De Blasio J A, Hasegawa T, et al. A preliminary report of a public experiment of a semantic service matchmaker combined with a UDDI business registry. In: Orlowska M E, Weerawarana S, Papazoglou M P, et al, eds. First International Conference on Service Oriented Computing (ICSOC), Trento, 2003. 208–224

14 Bellur U, Vadodaria H, Gupta A. Semantic matchmaking algorithms. In: Bednorz W, ed. Advances in Greedy Algorithms. Vienna: IN-TECH Publications, 2008. 481–502

15 McGuiness D L, Harmelen F V. OWL Web ontology language overview technical report. World Wide Web Consortium, 2004. Available from: http://www.w3.org/TR/2004/REC-owl-features-20040210

16 World Wide Web Consortium. OWL Web ontology language overview. 2001, accessed November 2009. Available from: http://www.w3.org/TR/owl-features/

17 Dean M. OWL-S: semantic markup for Web services. 2004. Available from: http://www.daml.org/services/owl-s/1.0/owl-s.pdf

18 Hewlett-Packard Development Company. Jena RDF API. 2001, accessed November 2009. Availabel from: http://www.hpl.hp.com/semweb/jena.htm

19 Mindswap-Maryland Information and Network Dynamics Lab. Semantic Web agents project: OWL-S Java API. 2004, accessed November 2009. Available from: http://www.mindswap.org/2004/owl-s/api/index.shtml

20 Berners-Lee T, Hendler J, Lassila O. The Semantic Web. Sci Am, 2001, 284: 34–43

21 OWL-S Ontology Editor CS/AI Department University of Malta. Protégé. 2004, accessed November 2009. Available from: http://owlseditor.semwebcentral.org/

22 IEEE Standard Upper Ontology (SUO). November 2009. Available from: http://suo.ieee.org/refs.html

23 Obrst L, Liu H, Wray R. Ontologies for corporate Web applications. AI Mag, 2003, 24: 49–62

24 Korotkiy M. On the Effect of ontologies on Web application development effort. In: TechRepublic Online Journal. Available from: http://whitepapers.techrepublic.com.com/whitepaper.aspx?docid=258949

25 Lo M, Gandon F. Integrating Dynamic Resources in Corporate Semantic Web: An Approach to Enterprise Application Integration Using Semantic Web Service. INRIA Research Report RR-5663, 2005. Available from: http://www.inria.fr/rrrt/rr-5663.html.

26 Garcia S F, Valencia G R, Martinez B R. An integrated approach for developing e-commerce applications. Expert Syst Appl, 2005, 28: 223–235

27 Sirin E, Parsia B. PELLET: An OWL DL Reasoner. In: International Workshop on Description Logics (DL2004), Whistler, 2004

28 Winkler W E. The state of record linkage and current research problems. Technical Report, U.S. Census Bureau, Washington, D.C, 1999

29 Winkler W E, Thibaudeau Y. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. Technical report, U.S. Bureau of the Census, Washington, D.C: Statistical Research Report Series RR91/09, 1991

30 World Wide Web Consortium. WSDL 1.1 Recommendation. 2001, accessed November 2009. Available from: http://www.w3.org/TR/WSDL

31 OASIS. UDDI Specification. 2002, accessed November 2009. Available from: http://www.uddi.org

32 World Wide Web Consortium. SOAP Recommendation. 2001, accessed November 2009. Available from: http://www.w3.org/TR/SOAP

33 World Wide Web Consortium. HTTP, Hypertext Transfer Protocol. Accessed November 2009. Available from: http://www. w3.org/Protocols/

34 Horrocks I, Harmelen F. Reference description of the DAML+OIL ontology markup language. 2001. Available from: http://www.daml.org/2001/03/reference.html

35 OWL-S Coalition. OWL-S specification. 2004, accessed November 2009. Available from: http://www.daml.org/services/ owl-s/1.1/

36 Wu J, Wu Z. Similarity-based Web service matchmaking. In: Proceedings of the IEEE International Conference on

Services Computing, Miami, 2005

37 Çelik D, Elçi A. A Semantic search agent approach: Finding appropriate Semantic Web services based on user request term(s). In: ITI 3rd International Conference on Information & Communication Technology (ICICT 2005). Cairo: IEEE Publ, 2005. 675–689

38 Çelik D, Elçi A. A semantic search agent discovers suitable Web services according to an e-learning client demand. In: 6th International Educational Technology Conference (IETC 2006). Gazimagusa, 2006. 416–424

39 Çelik D, Elçi A. Akilli anlamsal ağ Servisi arayicisi. In: Turkish Informatics Foundation Journal of Computer Science and Engineering, Istanbul, 2006. 131–142

40 Çelik D, Elçi A. Discovery and scoring of Semantic Web services based on client requirement(s) through a semantic search agent. In: International Workshop on Engineering Semantic Agent Systems ESAS 2006. In conjunction with 30th IEEE Annual International Computer Software and Applications Conference (COMPSAC 2006), Chicago, 2006. 273–278

41 Çelik D, Elçi A. Provision of Semantic Web Services through an Intelligent Semantic Web Service Finder. Multiagent Grid Syst, 2008, 4: 315–334

42 Çelik D, Elçi A. Searching Semantic Web services: An intelligent agent approach using semantic enhancement of client terms and matchmaker algorithm. In: International Conference on Intelligent Agents Web Technologies and Internet Commerce (IAWTIC 2005), Vienna, 2005. 28–30

43 Li L, Harrocks I. A software framework for matchmaking based on Semantic Web technology. In: Proceedings of the 12th International World Wide Web Conference, Budapest, 2003

44 Kuang L, Wu J, Deng S, et al. Exploring semantic technologies in service matchmaking. In: Proceedings of the 3rd European Conference on Web Services. 2005, accessed November 2009. Available from: http://suo.ieee.org/refs.html

45 Khalid M A, Fries B, Kapahnke P. OWL-S Service Retrieval Test Collection Version 2.1, Deutsches Forschungszentrum für Künstliche Intelligenz. 2006