A Graph-Based Approach to Web Service Matchmaking

Shang-Pin Ma
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung, Taiwan
Email: albert@ntou.edu.tw

Jonathan Lee
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan
Email: jlee@csie.ntu.edu.tw

Abstract—Web service discovery is the process of locating web services to meet requirements of service requesters, and is an important ingredient in building loosely-coupled SOA-based applications. In this paper, we propose a graph-based web service matchmaking approach based on WSDL without additional meta data or annotations. We divide the service matchmaking approach into two principal themes: constructing request graphs (RG) and service signature graphs (SSG) and calculating distance and similarity between a RG and an SSG. There are three main benefits in the proposed approach: (1) the structure of a service and involved datatypes are considered in the matching process; (2) the providers do not need to spend considerable effort annotating the service description using tags or ontology models; and (3) both the crisp similarity and fuzzy similarity can be produced automatically.

I. INTRODUCTION

Web service discovery is the process of locating web services to meet requirements of service requesters, and is an important ingredient in building loosely-coupled SOA-based applications. Usually, multiple published services could offer seemingly similar features but with some variations (e.g., different service interfaces, different attributes, and different quality-levels). A service requester can search, select and bind services that satisfy his or her intensions. Besides, service discovery mechanism is applied in service composition technology [6] [4] to select and bind component services and service management technology [5] to seek for failover services. Accordingly, how to fetch suitable web services from multiple service providers is always a significant issue to resolve [7]. In the past decade, a lot of solutions have been proposed to address this issue. These efforts can roughly be divided into two classifications: text-based methods [1] [10], which search services by employing information retrieval techniques, and semantics-based schemes [8] [11] [9] [2], which find services by semantic reasoning. However, in practice, both kinds of approaches are always unable to correctly find out the suitable web services. Text-based approaches treat the WSDL as plain text and does not consider the structure of the service and data types of I/O messages, it causes the matched WSDLs may not satisfy the users requirements. Ontology-based approaches are usually costly since service providers need to spend considerable effort annotating the service description according to prescribed domain ontology.

In this paper, we propose a graph-based web service matchmaking approach based on the open specification WSDL (Web Services Description Language) without additional meta data or annotations. The proposed approach extracts significant elements from a WSDL document and constructs Service Signature Graph (SSG) to represent the web service. Similar to SSG, the Request Graph (RG) is also built according to the user request. Based on these two kinds of graphs, the proposed approach focuses on both datatypes and keywords to calculate similarity between an RG and an SSG. We divide the matchmaking approach into two principal themes: constructing RG and SSG graphs and calculating distance and similarity between these graphs. Applying the proposed methods, the similarity can be produced automatically to help service requesters appropriately identify the appropriate web services suiting user requirements. Besides, the providers do not need to annotate the service description using tags or ontology models.

The remainder of this paper is organized as follows: Section 2 further describes the details of the proposed graph-based approach to web services matchmaking. Section 3 presents illustrative examples to show the effectiveness of the proposed approach. Conclusions and future work are shown in the final section.

II. GRAPH-BASED WEB SERVICE MATCHMAKER

To devise a service matchmaking mechanism considering input/output compatibility as well as textual similarity, we borrowed the notion proposed in [3] and treat graphs as the fundamental construct for service matchmaking. In this section, firstly, we introduce how to construct a graph that represents a service request or a web service. Secondly, we delineate how to calculate the similarity between a service request and a web service.

A. Construction RG and SSG Graphs

In the proposed approach, the Request Graph (RG) is built by capturing user’s requirements for searching ser-
services, and the Service Signature Graph (SSG) is constructed through parsing a WSDL document. Both graphs are partitioned into three parts: operation part, input part, and output part, which refer to service operation, service input messages, and service output messages respectively. Nodes in the graph are also divided into three classifications: connector nodes, datatype nodes, and keyword nodes. Connector nodes, including Service, Operation, Input Message, Output Message, Primitive Datatype, Complex Datatype and Keyword, play the role of connectors to link multiple datatype nodes and keyword nodes. An example of Service Signature Graph is shown in the right-hand side of Figure 1. The illustrative example is a stock inquiry service, which accepts the stock ID as input and returns the price and descriptions of the designate stock. In the service signature graph, connector nodes are shown in an abbreviation form: S, OP, IM, OM, PDT, CDT and KW, standing for Service, Operation, Input Message, Output Message, Primitive Datatype, Complex Datatype and Keyword respectively. S, OP, PDT and CDT nodes can be linked to one or more keyword nodes, which denote keywords extracted from the service name, the operation name or names of primitive datatypes or complex datatypes. A PDT node can be connected with a single primitive datatype node, specifying the primitive datatype of an input message or an output message. A CDT node can be linked with multiple PDT nodes. Besides, both IM node and OM node appear twice since we will calculate the distance and the similarity between RG and SSG for each graph part (i.e. operation part, input part, and output part). IM nodes or OM nodes are equivalent even emerging in different graph part (so-called peer nodes). Finally, although the original WSDL has a lot of additional information, we only keep significant and core data in the SSG since unrelated information may become noises and cause negative effect during the service matching process.

On the other hand, the Request Graph (RG) is much similar to the Service Signature Graph (SSG). The difference between RG and SSG is that SSG is generated based on WSDL whereas the RG is produced via parsing a request form which is filled out by the user. The request form mechanism is specially devised to let users input their requirements of services by following a simplified WSDL-like structure, e.g. the user can input service name, operation name, input/output message, etc.. When the WSDL-like request form is filled out, the RG can be build accordingly. The generated RG is compared with multiple SSGs to find out which services can suit the service request. Figure 1 is an example including both an RG (the left-hand side) and an SSG (the right-hand side). The similarity between these two graphs can be calculated by following our proposed approach.

1) Building Datatype Distance Tables: Before matching services, a distance calculation function is necessary to devise firstly for determining the distance between two datatypes. In this study, we apply FloydWarshall algorithm for finding shortest paths between two datatype nodes and calculating the distance value according to predefined datatype trees. In other words, the semantic distance (defined in Definition 5) is determined by counting the path length between two datatype nodes in the tree. Notably, FloydWarshall algorithm is a graph analysis algorithm for finding shortest paths in a weighted graph. Thus, even if we want to assign weights on edges between two datatype nodes or allow directional edges, the FloydWarshall algorithm can effectively calculate all shortest paths as well.

The proposed datatype trees are established based on XML built-in datatype hierarchy [12] specified by W3C to organize the linkages among datatypes. Notably, the link between two types in the type trees means “derived” relation and serves as the basis for calculating the semantic distance between two datatypes. For obtaining reasonable semantic distances, we divide the overall type hierarchy into four categories (numerical, date/time, string, and misc.), and slightly adapt the original hierarchy for improving the reasonableness. Accordingly, multiple datatype trees are built in the proposed approach, including numerical datatype tree (see Fig. 2), date/time datatype tree, string datatype tree (see Fig. 3), and misc. datatype tree.

By applying FloydWarshall algorithm, four datatype distance tables are established to store the distance value.
between any two datatype nodes in each category. When performing the service matchmaking process, the distance values can be obtained through querying these loop-up tables directly.

2) Extracting Keywords: Prior to generating the request graph and the service signature graph, additional pre-process is required to extract the keywords from the service request form and the WSDL. The pre-process include two activities: (1) Splitting combined terms: the names of the operation and input/output parameters are usually specified according to naming rules such as Pascal Case or Camel Case. This step enables the retrieval of separate terms according to naming conventions, and also removes useless words. Notably, in the proposed approach, we do not remove common verbs, such as “get”, “store”, “delete”, and “cancel”, since verbs are significant characteristics describing a service. (2) Word normalizing: in addition to splitting process, we also perform word normalization to transform a morphological word into its original form to avoid treating the same concept as different ones.

Through performing the above steps, retrieved keywords are enabled to attach to the target RG or the target SSG.

3) Transforming Complex Type and Array into Flattened Sub-Graph: Considering the complex type which is comprising of multiple primitive types is always a significant issue to address in service matchmaking process. Due to a complex type contains arbitrary levels of data, applying structural graph matching mechanism is difficult to identify two similar complex types if they are similar in content but different in structure. Accordingly, to eliminate the negative effect caused by structure, we transform the complex type into a “flattened” graph as Figure 4. Proposed transformation rules include:

- Whether how many levels reside in the complex type, retrieve underlying primitive datatypes only.
- Append corresponding PDT (primitive datatype) nodes to an CDT (complex datatype) node directly. Accordingly, only two levels will be kept in the graph.
- Append keywords describing the primitive type to the PDT node.
- Append keywords describing the complex type to the CDT node.
- Append keywords in middle levels to the PDT node.
- Remove overlapping keywords from the PDT level.

As the example shown in Figure 4, original three-level complex type is transformed as a two-level "flattened" complex type.

![Fig. 3. String Datatypes](Image)

![Fig. 4. Generating flattened sub-graph for the complex type](Image)
Definition 2: (The Most Specific Common Supertype of Two Primitive Datatypes.) Let the most specific common supertype of two datatypes \( dt_1 \) and \( dt_2 \) in the same type category be denoted by \( G(dt_1, dt_2) \). We have

\[
G(dt_1, dt_2) = \{ g \mid \text{general}(g, dt_1) \land \text{general}(g, dt_2) \land (\forall x \neq g)(\text{general}(x, dt_1) \land \text{general}(x, dt_2)) \Rightarrow \text{general}(g, y) \}
\]

where \( \text{general}(x, y) \) is a predicate to indicate that \( x \) is more general than \( y \) (i.e. \( x \) is a supertype of \( y \)).

Definition 3: (Unadjusted Distance between Two Primitive Datatypes). Let the unadjusted distance between two datatypes \( dt_1 \) and \( dt_2 \) in the same category be denoted by \( UD^T(dt_1, dt_2) \). \( UD^T(dt_1, dt_2) \) is defined as the distance of the shortest path from \( dt_1 \) to \( dt_2 \) in a type hierarchy. We have

\[
UD^T(dt_1, dt_2) = UD^T(dt_1, G(dt_1, dt_2)) + UD^T(dt_2, G(dt_1, dt_2))
\]

\[
UD^T(dt_1, dt_2) = 0 \text{ if } dt_1 = dt_2
\]

Definition 4: (Criticality of a Node). The criticality (i.e., the degree of importance) of a node is quantified by length. The length indicates the maximal admissible path through which a node can match other types in the type hierarchy. The higher the criticality of a node, the larger the set of matchable nodes and the propositional weight of this node in the query graph. The criticality of a node belonging to node type \( nt \) is denoted by \( \text{Cri}(nt) \).

In the proposed approach, the system administrator can assign different criticality value for keyword nodes, datatype nodes, and connector nodes for determining the importance of each node type in the service matchmaking process.

Definition 5: (Distance between Two Primitive Datatypes). Let the adjusted distance between two datatypes \( dt_1 \) and \( dt_2 \) be denoted by \( D^T(dt_1, dt_2) \). We have

\[
D^T(dt_1, dt_2) = \frac{UD^T(dt_1, dt_2) \times \text{Cri}(Datatype)}{\text{Cri}(Datatype)} + \frac{UD^T(dt_2, G(dt_1, dt_2)) \times \text{Cri}(Datatype)}{\text{Cri}(Datatype)}
\]

\[
D^T(dt_1, dt_2) = 0 \text{ if } dt_1 = dt_2
\]

where \( DTC(dt_i) \) is the datatype category for a given datatype \( dt_i \), \( \text{Cri}(dt_{i,j}) \) is a given criticality value for a given datatype category \( dt_{i,j} \) and is always assigned as the length of the longest path in the type tree for the datatype category \( dt_{i,j} \), \( \text{Cri}(Datatype) \) is a given value to indicate the importance of datatype nodes for the whole matching process, and \( \text{Root}(dt_i) \) is a predicate to indicate the root node for the category in which the \( dt_i \) resides.

For example, the unadjusted distance of long type and int type \( UD^T(\text{long}, \text{int}) \) is 1 (refer to Fig. 2). If \( \text{Cri}(\text{NumericalType}) \) is assigned as 8 (the length of the longest path), and \( \text{Cri}(\text{Datatype}) \) is assigned as 6, the adjusted distance \( D^T(\text{long}, \text{int}) \) is calculated as \( 1 \times 6 / 8 = 0.75 \). The adjusted distance between two datatype nodes is utilized in the further similarity calculation process.

2) Determining the Distance between Keyword Nodes: Using WordNet, we can classify the relationships between two keywords into four classifications: (1) the same concept; (2) synonym; (3) existing lexical relation but not synonym; and (4) antonym. Thus, we design a set of keyword distance determining rules (definition is shown in Definition 6) to determine the distance level between two keywords:

1) If two keywords belong to the same concept, the distance is zero.
2) If two keywords are synonyms, the distance is one.
3) If two keyword are antonyms, the distance is assigned four directly.
4) If there is a path connecting two keywords in wordNet, the distance is assigned two; otherwise, the distance is assigned three.

Definition 6: The distance between two keywords \( kw_1 \) and \( kw_2 \) calculated by the keyword distance determining algorithm is denoted by \( D^K(kw_1, kw_2) \rightarrow [0, 1, 2, 3] \).

3) Aligning the Query Graph and the Candidate Graph: Due to node sequence in the same level for both query graph and candidate graph can be changed without affecting semantics. Thus, how to align graphs is another challenge for graph matching problem. In this study, we propose an alignment procedure for two complex types or two input message (IM) including multiple input elements, as follows:

- Exhaustedly match all possible keyword mapping pairs between two sub-graphs, and find the best matchup, i.e. the matchup with smallest calculated distance value. An illustrative example is shown in Fig. 5, the best matchup is \( (aW_1, bW_1), (aW_3, bW_2), (aW_2, bW_3) \) with smallest distance value 5.
- Change the node sequence according to best matchup. Following the previous example, the node sequence in the candidate graph should be changed according to the above matching result with distance value 5.

4) Calculating Overall Similarity and Fuzzy Similarity: Based on the above calculation algorithms, we can calculate the distance between two connector nodes or two datatypes or two keywords. In this section, we explain the method for calculating the overall similarity and a set of uncertain similarities.
Definition 7: (Distance between Two Nodes). Let \( n_1 \) and \( n_2 \) be nodes in the query graph and the candidate graph, respectively. The distance between \( n_1 \) and \( n_2 \) is denoted by \( D(n_1, n_2) \). We have

- \( D(n_1, n_2) = \min\{D^T(n_1, n_2), \text{Cri}(\text{Datatype})\} \) if both \( n_1 \) and \( n_2 \) are datatype nodes.
- \( D(n_1, n_2) = \min\{D^K(n_1, n_2), \text{Cri}(\text{Keyword})\} \) if both \( n_1 \) and \( n_2 \) are keyword nodes.
- \( D(n_1, n_2) = \min\{D^\ell(n_1, n_2), \text{Cri}(\text{Connector})\} \) if both \( n_1 \) and \( n_2 \) are connector nodes.
- \( D(n_1, n_2) = 0 \) if \( n_1 \) and \( n_2 \) do not belong to the same node type.

where \( \text{Cri}(\text{Datatype}), \text{Cri}(\text{Keyword}), \) and \( \text{Cri}(\text{Connector}) \) are given values to indicate the importance of datatype nodes, keyword nodes, and connector nodes, respectively.

For calculating the similarity between two graphs, we formally define the concept of a compatible set in Definition 8 to model the concept of the correspondence of all nodes in these two graphs. Definition 9 - Definition 12 describe the detailed guidelines to compute the similarity of two graphs.

Definition 8: (The Compatible Set between Two Graphs). Let \( g_q \) be a query graph that contains nodes \( n_i \) where \( i = 1...p \), and \( g_c \) be a candidate graph that contains nodes \( m_j \) where \( j = 1...q \). The compatible set \( CS(q, c) \) between two graphs \( g_q \) and \( g_c \) contains \( p \) pairs of nodes to indicate the correspondence of all nodes in \( g_q \) and \( g_c \) after performing the alignment procedure.

Definition 9: (Distance between Two Graphs). Let \( g_q \) be a query graph that contains nodes \( n_i \) where \( i = 1...p \), and \( g_c \) be a candidate graph that contains nodes \( m_j \) where \( j = 1...q \). The distance between two graphs \( g_q \) and \( g_c \) is denoted by \( D(g_q, g_c) \). We have \( D(g_q, g_c) = \sum_{(n_i, m_j) \in CS(g_q, g_c)} D(n_i, m_j) \).

Definition 10: (Criticality of the Query Graph). Let \( g \) be a query graph that contains nodes \( n_i \) where \( i = 1...p \). The criticality of the query graph \( g \) is denoted by \( \text{Cri}(g) \). We have \( \text{Cri}(g) = \sum_{n_i \in g} \text{Cri}(n) \).

Definition 11: (Relative Criticality of a Candidate Graph) Let \( g_q \) be a query graph and \( g_c \) be a candidate graph. The relative criticality of the candidate graph \( g_c \) is denoted by \( \text{Cri}_r(g_c|g_q) \). We have \( \text{Cri}_r(g_c|g_q) = \text{Cri}(g_c) - D(q, g_c) \).

Definition 12: (Similarity between Two Graphs) Let \( g_q \) be a query graph and \( g_c \) be a candidate graph. The similarity between two graphs \( g_q \) and \( g_c \) is denoted by \( S(g_q, g_c) \). We have \( S(g_q, g_c) = \frac{\text{Cri}(g_q)}{\text{Cri}(g_c)} \).

Based on the proposed similarity calculation process, we also define additional calculation rules to compute distance among complex types and arrays:

- If both the query graph and the candidate graph are either primitive type or complex type, not array, the graph distance can be calculated as usual.
- If the query graph is a primitive datatype \( \text{PDT}_1 \) and the candidate graph is a primitive-type array \( \text{PDT}_2 \), the distance between \( \text{PDT}_1 \) and \( \text{PDT}_2 \) depends on the distance between the \( \text{PDT}_1 \) and the element datatype in \( \text{PDT}_2 \).
- If the query graph is a complex datatype \( \text{CDT}_1 \) and the candidate graph is a complex-type array \( \text{CDT}_2 \), the distance between \( \text{CDT}_1 \) and \( \text{CDT}_2 \) also depends on the distance between the \( \text{CDT}_1 \) and the element datatype of \( \text{CDT}_2 \).
- If the query graph is a primitive datatype \( \text{PDT}_1 \) and the candidate graph is a complex-type array \( \text{CDT}_2 \), the distance between \( \text{PDT}_1 \) and \( \text{CDT}_2 \) also depends on the distance between the \( \text{PDT}_1 \) and any PDT which resides in the CDT of \( \text{CDT}_2 \).
- If the query graph is a complex datatype \( \text{CDT}_1 \) and the candidate graph is a primitive-type array \( \text{PDT}_2 \), the distance between \( \text{CDT}_1 \) and \( \text{PDT}_2 \) is directly assigned as the maximal distance.
- If the query graph is a primitive-type array \( \text{PDT}_1 \) or a complex-type array \( \text{CDT}_1 \) and the candidate graph is a primitive datatype \( \text{PDT}_2 \) or a complex datatype \( \text{CDT}_2 \), the distance between \( \text{PDT}_1 \) and \( \text{CDT}_2 \) is assigned as the distance of the element datatype of \( \text{PDT}_1 \) and \( \text{CDT}_2 \) plus half of the graph criticality of the query graph.

This rule intentionally emphasizes the mismatch between a normal datatype and an array.

Definition 13: (Service Similarity between a Service Signature Graph and a Request Graph) Let \( r_g \) be a request graph where \( \text{opp}_r \), \( \text{ip}_r \), \( \text{op}_r \) are sub-graphs of \( r_g \) representing the operation part, input part, and output part of \( r_g \) respectively, and \( s_{ssg} \) be a service signature graph where \( \text{opp}_{ssg} \), \( \text{ip}_{ssg} \), \( \text{op}_{ssg} \) are sub-graphs of \( s_{ssg} \) representing the operation part, input part, and output part of \( s_{ssg} \) respectively. The service similarity between \( r_g \) and \( s_{ssg} \) is denoted by \( S_{SS}(r_g, s_{ssg}) \) and fuzzy similarity between \( r_g \) and \( s_{ssg} \) is denoted by \( FSS_{SS}(r_g, s_{ssg}) \) We have:

- \( S_{SS}(r_g, s_{ssg}) = \frac{1}{3} \sum_{i, p} \text{opp}_{ssg}(r_g, s_{ssg}), \text{ip}_{ssg}(r_g, s_{ssg}), \text{op}_{ssg}(r_g, s_{ssg}) \)
- \( FSS_{SS}(r_g, s_{ssg}) = \left| S_{SS_{min}}(r_g, s_{ssg}), S_{SS_{max}}(r_g, s_{ssg}) \right| \)
- \( S_{SS_{min}}(r_g, s_{ssg}) = \min\{S_{opp_{ssg}}, S_{ip_{ssg}}, S_{op_{ssg}}\} \)
- \( S_{SS_{max}}(r_g, s_{ssg}) = \max\{S_{opp_{ssg}}, S_{ip_{ssg}}, S_{op_{ssg}}\} \)

Notably, the input part of \( s_{ssg} \) is treated as the query graph and the input part of \( r_g \) is treated as the candidate graph since the user’s input must cover the service’s input for successfully invoking the service.

In summary, when a user issues a request, the proposed approach will calculate the similarity scores of all web services in the repository, and return services with top-N scores to the user.

III. ILLUSTRATIVE EXAMPLES

In order to illustrate the proposed service matchmaking approach, an example of stock inquiry service is offered.
For the sake of clarity, the example is simplified in order to explain how our methodology can be utilized.

The example is shown in Figure 6. The left-hand side of the figure is the request graph (rg), which captures the user’s requirement, and the right-hand side of figure is the service signature graph (ssg) which extracts important features of a web service. Notably, the ssg is already adapted by performing the alignment process for the sake of simplicity. In this example, the user wants to search for a stock inquiry service which can return the price of a designate stock id or stock name, i.e. the user can provide the id and the name of a stock. On the other hand, the web service is a stock inquiry service which can return the value and descriptions of a designate stock. Following the proposed approach, we can calculate the similarity of operation part, input part, and output part as follows:

- \( S_g(opp^*_s, opp^*_g) = \frac{1}{1.4} = 0.72 \)
- \( S_g(ipp^*_s, ip^*_g) = \frac{1}{1.6} = 0.625 \)
- \( S_g(op^*_s, op^*_g) = \frac{1.3}{3.7} = 0.351 \)

Note that

1. the distance of the keyword “price” and the keyword “value” is 1 since “price” and “value” are synonyms; and
2. the distance of the datatype “float” and the datatype “string” is calculated as 2.7 by following Definition 5: \( D_f(\text{float}, \text{string}) = \frac{2 \times 6 + 1 \times 5}{2 + 1} = 1.5 + 1.2 = 2.7 \)

According to the separate similarity value of each part, we can calculate the overall similarity and the fuzzy similarity:

1. the overall similarity \( S_{op}(rg, ssg) = \frac{0.93 + 1.00 + 0.93 + 1.00 + 0.72}{5} = 0.88 \); and
2. the fuzzy similarity \( F_{SS}(rg, ssg) = [SS_{min}(rg, ssg), SS_{max}(rg, ssg)] = [0.72, 1.00] \).

Fig. 6. Illustrative Example: Stock Service

IV. CONCLUSION

In this study, we propose a graph-based web service matchmaking approach based on the open specification WSDL to extract various information, such as datatypes and keywords, to construct service signature graphs (SSG) and request graphs (RG) for capturing significant features of services and requests, and to calculate similarity values between an RG and an SSG.

Comparing with typical text-based and ontology-based methods, three main benefits are offered by the proposed approach: (1) in addition to textual similarity, the structure of a service and involved datatypes are considered in the matching process; (2) the providers do not need to spent considerable effort annotating the service description using tags or ontology models; and (3) both the crisp similarity and fuzzy similarity can be produced automatically to help service requesters appropriately identify the appropriate web services suiting user requirements.

Our further work will focus on establishing a web service search engine based on the proposed approach to let users search web services more conveniently just like searching web pages by Google.

REFERENCES