Self-Adaptive Service Organization for Pragmatics-Aware Service Discovery


Published in:
Proceedings - 2017 IEEE 14th International Conference on Services Computing, SCC 2017

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
Copyright 2017 IEEE. This work is made available online in accordance with the publisher’s policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Download date: 09. Aug. 2019
Self-adaptive Service Organization for Pragmatics-aware Service Discovery

Dionysis Athanasopoulos
School of Engineering & Computer Science
Victoria University of Wellington, New Zealand
dionysis.athanasopoulos@ecs.vuw.ac.nz

Abstract—Traditional service registries form groups of functionally related services based on service descriptions, without taking into account past service-usage from the consumers’ perspective, a.k.a. pragmatics. Thus, we propose a self-adaptive service-organization mechanism that follows an iterative and evolutionary life-cycle for autonomically evolving service groups by the arrival of pragmatics. Since pragmatics are not available beforehand, we propose an on-line service-organization algorithm that leverages the highly accumulated number of pragmatics. We evaluate the efficiency and effectiveness of our mechanism on the services of a publicly available benchmark and the results show that the effectiveness of the traditional service-organization is improved, while a low number of pragmatics is greedily stored.

Keywords—service organization; self-adaptive; on-line hierarchical clustering; pragmatics.

I. INTRODUCTION

Whereas famous vendors (e.g. Google, Amazon) provide access to their resources via adopting the Service-oriented Architecture (SoA) style, the majority of the available Web services has not been discovered or invoked [1]. The process for discovering available services typically includes the submission of service requests to registries, which return a set of service candidates by using matching mechanisms. Registries form organization schemes that are consisted of groups of functionally similar services\(^1\). Services’ functionality is specified based on the providers’ perspective using syntactic (e.g. WSDL) descriptions, usually enriched with semantics (e.g. WSDL-S, OWL-S). Registries may further form classification schemes, consisted of classes of groups. Each class is usually characterized by a label [2]. However, traditional registries do not organize/classify services considering past service usage from the consumers’ perspective, a.k.a. pragmatics [3].

Pragmatics are either explicit, i.e. consumers specify them (e.g. XML-based queries [4]), or implicit, i.e. past service usage. Since explicit pragmatics requires an incentive for consumers to provide them, we consider only the case of implicit pragmatics. While the importance of pragmatics has been underlined [5], a few early approaches\(^2\) have been proposed that offer pragmatics-aware searching facilities [4–7], but not service organization/classification. The restriction of the approaches is that they compare pragmatics-aware queries to traditional service descriptions. To overcome this restriction, service-organization mechanisms should dynamically enhance service descriptions with pragmatics and accordingly evolve organization schemes.

Contribution. We propose a mechanism that organizes services into groups based on both service interfaces and implicit pragmatics. As an indicative aspect of service pragmatics, we consider previously used service instances, given as input to operation invocations\(^3\). To be responsive in dynamically arrived pragmatics, the mechanism enters into an iterative and evolutionary life-cycle, during which it autonomically evolves its organization scheme. In particular, we propose a self-adaptive mechanism [8] that goes through an off-line phase to produce an initial organization scheme and an on-line phase to evolve the scheme based on pragmatics.

During the on-line phase, our mechanism initially enhances service interfaces with newly arrived schema instances. To do so, it uses an extended conceptual model for service-interfaces that permits such an enhancement. The mechanism also re-calculates the similarities between the organized services via using our semantic edit-distance metric for comparing schema instances.

\(^1\)The non-functional service organization is not covered in this work.

\(^2\)Our work is not related to service-recommendation approaches that consider historical usage data for predicting consumers’ interests.

\(^3\)More aspects of pragmatics will be used in the next mechanism version.
conceptual model of service interface. Section IV defines our pragmatics-aware service-similarity metric. Section V describes the proposed life-cycle of self-adaptive service-organization. Section VI specified the modus operandi of the on-line service-organization algorithm. Section VII presents the experimental evaluation. Section VIII summarizes our approach and discusses its future research directions.

II. RELATED WORK

Our work belongs to the fields of the service discovery (Section II-A) and self-adaptive software (Section II-B).

A. Service Discovery

We categorize the approaches along two dimensions: service organization and pragmatics-aware service discovery. Due to the vast number of approaches in the first dimension, we describe the top approaches based on the ranking in [11].

Service organization. It is performed in a top-down (schemes are pre-defined and incrementally populated) or bottom-up fashion (groups/classes are reverse-engineered).

1) Top-down: The majority of the approaches uses centralized UDDI registries. [12] extends UDDI’s keywords-based searching-facilities with ontology-based matching. There are also distributed approaches, which use Peer-to-Peer (P2P) nodes for indexing the service corpus. The underlying registry of a peer can be a UDDI registry [13], [14] or a database system [15]. [2], [16], [17] do not use UDDI or P2P nodes, but they define service (XML-based [16], label-based [2], OWL-S-based [17]) abstractions as representatives of service groups/classes.


Comparing the approaches in terms of the used service descriptions, organization process and fashion (Table I(a)), we observe our approach is the only (i) self-adaptive and (ii) (bottom-up) on-line approach that (iii) considers pragmatics.


B. Self-adaptive Software

Self-adaptive software autonomically evolves in response to changes, caused by external sources [5]. Self-adaptive SoA approaches do not focus on the service-organization task. In particular, [25] focuses on integration problems that derive from changes in used services and [26] proposes a formal model for the actions taken by self-adaptive SoA software. To the best of our knowledge, our approach is the first self-adaptive service-organization approach.

III. PRAGMATICS-AWARE SERVICE-INTERFACE MODEL

Our conceptual model of service interface is derived by the WSDL-based specification of services and extended with data-structures suitable for storing the top-k schema instances of each service interface. In particular, a service interface SI (Table II (Eq. 1)) is characterized by its name and set of operations OPs (Table II (Eq. 2)). An operation OP (Table II (Eq. 3)) accepts an input message in and produces an output message out. A message M (Table II (Eq. 4)) is characterized by its id (it is unique across all of the available services), name, XML schema S, and top-k schema instances. In each message, only the instances d[k] of its schema are stored that are the most similar to

---

4uddi.xml.org
instances of other service messages \( \text{id}[k] \). Their similarity values \( \text{sim}_D[k] \) are also stored. The arrays \( d \), \( \text{sim}_D \), and \( \text{id} \) (hereafter called \textit{pragmatics arrays}) have position correspondence (Table II (Eq. 4)).

Message schema is represented as rooted tree of elements (Table II (Eq. 5)). Each element is characterized by its name, built-in type (subtype of XML \textit{anyType}\(^5\)), and children (Table II (Eq. 6)). We consider only the name and built-in type attributes of element, since they play the most important role in element similarity [27]. Similarly, an important XML document \( D \) of schema, it is also represented as rooted tree of labels (Table II (Eq. 7)). Each label \( L \) is characterized by its name, value, and children (Table II (Eq. 8)).

Illustrative example. We consider three services that manage information about pre-, post-graduate and PhD students. Fig. 1 represents the example service-interfaces by using our model. It also represents two example schema-instances\(^6\), along with the content of the pragmatics arrays.

IV. PRAGMATICS-AWARE SERVICE SIMILARITY

To calculate the service-interface similarity, we follow the hierarchical structure of the service-interface model and further consider the stored schema-instances, as follows.

A. Suite of Similarity Metrics for Service Interfaces

Interface similarity. The similarity metric \( \text{sim}_{SI} \) (Table III (Eq. 1)) for two service interfaces is defined as the arithmetic mean of (i) the complement of the normalized edit distance \( \text{NED} \) (Table III (Eq. 2 and 3)) between the interface names\(^7\) and (ii) the arithmetic mean of the similarities of their matched operations. To find the best possible operation matching \( \text{M}_{OP} \) (Table III (Eq. 4)), we solve the optimization problem of the maximum weighted matching in a bipartite graph [29]. The nodes of the graph correspond to the operations of the interfaces, while the edges correspond to the similarities of the operation pairs.

![fig1.png](https://www.w3.org/TR/xmlschema-2)

Message similarity. The similarity metric \( \text{sim}_M \) (Table III (Eq. 6)) for two messages is defined as the arithmetic mean of (i) the normalized edit distance between the message names and (ii) the aggregation of their schema and instance similarities (bold font in Table III). We assume that instance similarity has higher priority than schema similarity.

To this end, we propose the priority-based aggregation function \( \text{F}_{PR} \) (Table III (Eq. 13)), in which the prioritized role of the \( x \) objective is reflected by the fact that \( x \) contributes with its entire value. On the contrary, the non-prioritized \( y \) objective contributes with its product with \( x \), which is of lower magnitude than the \( x \) value (\( x, y \in [0, 1] \)).

Regarding the similarity of schema instances, motivated by the fact that similar instances are usually specified by similar hierarchical structures [30], we propose in Section IV-B a semantic metric that compares tree structures.

Schema similarity. The similarity metric \( \text{sim}_S \) (Table III (Eq. 7)) for two schemas is defined as the arithmetic mean of the similarities of the matched schema elements. To find the best possible schema matching \( \text{M}_S \) (Table III (Eq. 9)), we solve again the maximum weighted matching problem.

Element similarity. The similarity metric \( \text{sim}_E \) (Table III (Eq. 8)) for two elements is defined as the arithmetic mean of (i) the normalized edit distance between the element names and (ii) the similarity between the element built-in types (\( \text{sim}_T \) metric), \( \text{sim}_T \) values are statically defined in [27].

B. Semantic Edit-distance Metric for Schema Instances

The recent robust tree edit-distance (RTED) technique [31] converts the compared trees into strings (their format is depicted at the bottom of Fig. 1). Following, it counts the min number of the operations (relabel, insert, delete) required to transform one tree into another, without though dealing with semantic similarities (i.e. exact label matching). The RTED result is the optimal edit script (i.e. the sequence of operations) for the compared trees. Without changing the algorithmic logic of RTED, our metric takes as input a RTED edit-script \( \text{ES} \) (Table III (Eq. 11)), which is defined in [31], and computes a new edit-distance value as follows. For each pair of matched labels, if they are internal nodes, the metric calculates the normalized edit distance (Levenshtein’s metric [28]) of their names (first branch in Table III (Eq. 12)). If the matched labels are tree leaves (a.k.a. real data), the metric considers only the case of numeric data, calculating their absolute difference divided by the absolute value of the min numeric datum (second branch in Table III (Eq. 12))\(^8\).

---

\(^5\)www.w3.org/TR/xmlschema-2

\(^6\)Due to lack of space, we present instances for the first two services.

\(^7\)We adopt the Levenshtein’s string-based metric [28].

\(^8\)We leave as future work the comparison of other kinds of data, since they depend on the service domain (domain-specific ontology is needed).
our metric is commutative, since RTED and Levenshtein’s metrics are commutative. Overall, the similarity between two schema instances $sim_D$ (Table III (Eq. 10)) equals to the complement of the normalized TED.

**Illustrative example.** Returning to the previous example (Fig. 1), the RTED similarity of the schema instances equals to 10 (normalized value, 0.63) and is identified by the different label pairs (bold font). On the contrary, their $sim_D$ similarity equals to 5.4 (normalized value, 0.8).

V. LIFE-CYCLE OF SELF-ADAPTIVE ORGANIZATION

The proposed life-cycle is in accordance to the generic one for self-adaptive software [8], which includes the off-line and on-line phases. In our case, the off-line phase corresponds to the Initial Service Organization (Section V-A), while the on-line phase to the Self-adaptive Service Organization (Section V-B).

A. Initial Service Organization

This phase is depicted at the left-hand side of Fig. 2. Since we focus on the second phase, we assume that the first phase adapts an existing off-line hierarchical clustering mechanism [32]. The mechanism is adapted to use our service-interface model and similarity metrics.

**Illustrative example.** Giving as input the three service-interfaces of Fig. 1 to the off-line clustering mechanism, the dendrogram of clusters at the left-hand side of Fig. 4 is produced. The off-line clustering method at its first iteration merges ($si_1$, $si_2$), since their similarity is greater than the similarities of the remaining pairs, ($si_1$, $si_3$) and ($si_2$, $si_3$).

B. Self-adaptive Service Organization

The second phase is depicted at the right-hand side of Fig. 2. During this phase, the On-line Calculation of Instance Similarity and On-line Evolution of Organization Scheme mechanisms are executed, whose algorithmic details are provided in Section VI.

VI. SELF-ADAPTIVE SERVICE ORGANIZATION

The On-line Calculation of Instance Similarity is iteratively executed by the arrival of new schema instances. If the content of the pragmatics arrays is not updated for a fixed continuous number of newly arrived instances, then the On-line Evolution of Organization Scheme is executed for releasing a new version of the scheme. The underlying algorithms and their complexity, are detailed in Sections VI-A and VI-B.

A. On-line Calculation of Instance Similarity

Algorithm 1 accepts a schema instance $d_x$, the constant $k$, and the list of the service interfaces $sis$ that are contained in the organization scheme (input line in Alg. 1). The algorithm

---

**Figure 1.** The representation of the example service-interfaces by using the pragmatics-aware service-interface model.

**Figure 2.** The life-cycle of the self-adaptive service organization.
returns the same list of service-interfaces \( s_i \)s (output line in Alg. 1) with potentially updated their pragmatics arrays.

Technically, the algorithm compares \( d_s \) against the stored instances of the other service interfaces (Alg. 1 (1-6)) and by using our \( \text{sim}_M \) metric, it identifies the instance with the maximum similarity (Alg. 1 (7-11)). The algorithm compares instances only if their schemas are similar (Alg. 1 (6)). Following, the algorithm checks if the max similarity is greater than at least one of the top-k similarities that have been stored in the corresponding message of \( d_s \) (Alg. 1 (12-14)). If it holds, then the algorithm updates the pragmatics arrays of this message (Alg. 1 (15-19)).

Each message stores its own list of top-k instances, since this list is not necessarily the same with those of other messages. However, if the same pair of instances appears in two lists, then their similarities are very close to each other or identical, as shown in the evaluation results.

**Complexity.** For a new instance, the algorithm iterates over the existing interfaces, their operations, and the stored instances. Thus, the time and space complexity of the algorithm is captured by the linear expression, \( \mathcal{O}(k*|s_i|*|ops|) \). Note that the above product equals to the total number of the stored schema instances in all service interfaces.

**Illustrative example.** Returning to our running example, the mechanism is iteratively executed giving as input the three services, randomly generated schema instances\(^{10}\), and the value 2 for \( k \). The mechanism execution stops when no changes are performed in the pragmatics arrays, whose final content is depicted in Fig. 3(a). We observe that the instances of the pair \((s_i, s_j)\) have the highest similarity, in contrast to the pair \((s_i, s_2)\) identified by the off-line mechanism. We also observe that there is a convergence in the similarities of the bidirectional pairs.

![Figure 3. The results of our mechanisms for the running example.](image)

---

\(^{10}\)The generator of schema instances is described in Section VII.
A. Experiment Setup

We firstly executed the off-line phase, and then, the on-line phase. The mechanism, **On-line Calculation of Instance Similarity**, is iteratively executed for randomly generated schema instances. If none pragmatics-array is updated for continuous iterations, the mechanism, **On-line Evolution of Organization Scheme**, is executed. Both mechanisms are executed for the value 2 of k as a representative low k value, since high values negatively affect the mechanism efficiency. Thus, our evaluation includes two parts, one part for each mechanism (Sections VII-B and VII-C).

B. Evaluation of On-line Calculation of Instance Similarity

For each OWLS-TC4 query, we compare the instance similarities against the similarities of the corresponding schemas in order to examine if their values are close (their present the evaluation results for $Q_1$-$Q_{20}$, but we reached to analogous conclusions for the remaining queries. Finally, OWLS-TC4 provides ideal matchings between the schema elements of the services, based on the ontology classes to which the elements belong.

VII. EXPERIMENTAL EVALUATION

We implemented the mechanisms and metrics, used by the on-line and off-line phases. The executable file of our research prototype is available online at this location\(^{11}\). We evaluate the effectiveness and efficiency of the mechanisms on the Web services of the benchmark, OWLS-TC4\(^{12}\). We firstly set up our experiments.

A. Experiment Setup

**Benchmark.** OWLS-TC4 has been used by many state-of-the-art approaches that calculate service similarity (e.g. [19], [27]). The majority of the OWLS-TC4 services was retrieved from public IBM UDDI registries. Each service document contains only one interface, which provides a single operation. OWLS-TC4 also provides the result sets of 42 queries, $Q_1$-$Q_{42}$, which correspond to groups of semantically similar services. Due to lack of space, we do not specify it in an algorithmic format.

Complexity. Since the mechanism iterates over the existing interfaces, their operations, and the stored instances, its time and space complexity scales with the same linear expression with the previous mechanism. This expression does not scale with the quadratic number of all possible interface pairs as happens in the classical clustering.

Illustrative example. Returning to our running example, the second mechanism is executed for the results of the first mechanism. From the new results (Fig. 3(b)), we observe that the similarities of bidirectional pairs converge to close values and that the pair, $(si_2, si_3)$, has the highest similarity. Consequently, the produced dendrogram (Fig. 4) differs from that of the off-line mechanism.

**B. On-line Evolution of Organization Scheme**

The mechanism calculates the similarities of the service-interface pairs, whose schema instances have been stored in the pragmatics arrays, and stores the pairs sorted with respect to their similarities. Then, it starts from the pair that has the highest similarity and clusters service interfaces contained in the same pair, until no more clusters can be formed (i.e. their similarities equal to zero) or only one cluster remains. Since the modus operandi of the mechanism is straightforward, we do not specify it in an algorithmic format.

**Algorithm 1**

\[
\text{On-line Calculation of Instance Similarity}
\]

\[
\text{Input: } d_x : D, \ k : \text{inc.}, \ sis = \{si_1 : SI, si_2 : SI, \ldots\} \\
\text{Output: } sis = \{si_1 : SI, si_2 : SI, \ldots\}
\]

1: $si_x \leftarrow \text{IDENTIFY}(d_x) \mid d_x \text{ is instance of } si_x.\text{ops.}\text{op.in.s}$
2: for all $si_i \in sis$ do
3: \hspace{1em} if $si_i == si_x$ then continue end if
4: \hspace{1em} for all $op_j \in si_i.\text{ops.\text{do}}$ do
5: \hspace{2em} $s_j \leftarrow op_j.\text{in.s}$
6: \hspace{1em} if $\text{sim}(s_x, s_j) > 0$ then
7: \hspace{2em} Double $\text{max} \leftarrow -1$
8: \hspace{2em} for all $d[z_1] \in op_j.\text{in.d}$ do
9: \hspace{3em} Double $\text{sim} \leftarrow \text{sim}_D(d_x, d[z_1])$
10: \hspace{3em} if $\text{sim} > \text{max}$ then $\text{max} \leftarrow \text{sim}$ end if
11: \hspace{2em} end for
12: \hspace{1em} for all $d[z_2] \in op_x.\text{in.d}$ do
13: \hspace{2em} if $\text{max} < d[z_2]$ then break end if
14: \hspace{1em} end for
15: \hspace{1em} if $z_2 <= k$ then
16: \hspace{2em} $\text{in}_x.\text{d}[z_2] \leftarrow d_x$
17: \hspace{2em} $\text{in}_x.\text{sim}[z_2] \leftarrow \text{max}$
18: \hspace{2em} $\text{in}_x.\text{id}[z_2] \leftarrow \text{in}_j.\text{id}$
19: \hspace{1em} end if
20: \hspace{1em} end if
21: end for
22: end for

---

\(^{11}\)ecs.victoria.ac.nz/foswiki/pub/Main/DionysisAthanasopoulos/pragmatics.zip
\(^{12}\)projects.semwebcentral.org/projects/owls-tc
difference $\leq 0.2$) and if the similarities of the bidirectional pairs converge to close values (their difference $\leq 0.1$). Moreover, we counted the number of the stored schema instances, since the mechanism efficiency depends on it.

The results are depicted in Table IV (from the second to fifth column). At the bottom of Table IV, we also calculated the average results for large- (annotated by L) and small-sized queries (S, $|\text{pairs}| \leq 20$). As expected, the percentage of the stored pairs is low, 19%, for large-sized queries and medium, 65%, for small-sized queries. The percentage of the bidirectional pairs is medium, 42%, for large-sized queries and medium-high, 71%, for small-sized queries. This result verifies our previously mentioned intuition that the lists of the top-k instances are not the same across the services. The percentage of the bidirectional pairs that converges to close values is very high, 99% and 97%, for large- and small-sized queries, respectively. Finally, the medium-high percentage of the 69% and 71% for large- and small-sized queries, respectively, has instance similarity close to schema similarity. We examine below the impact of the complements of these percentages on the service organization.

### C. Evaluation of On-line Evolution of Organization Scheme

For each OWLS-TC4 query, we compare the ranked sets of service-interface pairs of the two phases. In particular, we compare the percentages of their common pairs and repositions. From the results (6th and 7th columns of Table IV), the percentage of the common pairs (resp. repositions) is high, 87% (resp. medium, 61%), for small-sized queries, and medium, 58% (resp. high, 92%), for large-sized queries. To explain the results, we calculated the effectiveness of the off-line service-similarity and we inspected the ranked sets.

We calculated the effectiveness of service similarity against the ideal matchings via using the F-measure metric [33].

We present in Fig. 5 a part of our inspection results for two representative queries ($Q_9$ and $Q_4$, small- and large-sized, respectively). Since the off-line similarity and F-measure values equal to 1.0 in $Q_9$, very small differences in the online similarities led to repositions. In $Q_4$, the differences between the instance and schema similarities (see previous subsection), along with the usage of the constraint $k$, led to discard the pairs, ($s_4$, $s_3$) and ($s_3$, $s_2$). However, these repositions improved the organization results, since the pair ($s_4$, $s_2$) correctly is of much higher similarity than those of the other pairs. Concluding, instances helped in distinguishing similar pairs in a fine-grained way.

### VIII. Conclusions & Future Work

We proposed a self-adaptive service-organization mechanism that follows an iterative life-cycle for autonomically evolving organization schemes by the arrival of pragmatics. Since pragmatics are not available beforehand, we proposed an on-line algorithm. We evaluated the efficiency and effectiveness of our mechanism on the services of a publicly available benchmark and the results show the effectiveness of the traditional service-organization is improved, while a low number of pragmatics is greedily stored.

Our future research purpose is to extend our service-interface model and on-line algorithm to use other aspects of service pragmatics (e.g. service-usage scenarios). An interesting direction is to extend our approach to cope with big real-time monitoring data. A final direction is to propose a user-collaborative platform that gathers service pragmatics
<table>
<thead>
<tr>
<th>(dl1, sl1)</th>
<th>Off-line</th>
<th>F-measure</th>
<th>On-line</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1.00, 1.00)</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>(1.00, 1.00)</td>
<td>1.00</td>
<td>1.00</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>(1.00, 1.00)</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Figure 5. Detailed evaluation results for two OWLS-TC4 queries.

REFERENCES


