Constructing composite web services from natural language requests

JongHyun Lim, Kyong-Ho Lee*

Department of Computer Science, Yonsei University, Seoul, Republic of Korea

A R T I C L E   I N F O

Article history:
Received 14 November 2008
Received in revised form 22 August 2009
Accepted 28 September 2009
Available online 15 October 2009

Keywords:
Natural language interface
Web service
Composite service
Ontology

A B S T R A C T

As a demand on composite Web services increases, there is a growing interest in a convenient access to them. This paper proposes a natural language interface to Web services, which can be used even by a novice user who does not know Web service technologies. Given a user’s natural language request to a composite service, the proposed method generates an abstract workflow, which describes the constituent tasks and their transitions in a composite service. Specifically, the proposed method constructs a sophisticated abstract workflow from complex sentences with phrases and control constructs. Experimental results with a variety of natural language requests show that the proposed method successfully extracts abstract workflows, resulting in an accuracy of 95.2%.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

As Web services [1] facilitate the interoperability of heterogeneous platforms, they are gaining momentum as a standard of distributed software systems in ubiquitous environments. While a single service may not satisfy a user’s complicated request like a sentence block, a composite service may satisfy the complex request. However, it is hard and time consuming to explicitly specify how to compose services. We need a convenient user interface, which can be used by a novice user who does not know Web service technologies. A natural language interface would be one of the most user-friendly and convenient interfaces to invoke services.

Several methods concerning natural language interfaces for Web services have been proposed. Bosca et al. [2–4] construct an abstract workflow from a natural language request. Xie et al. [5] transform a user’s natural language request to the SOBL (Semantic Object Behavior Language) language and find a service, which satisfies the request. Englmeier et al. [6] propose WS-Talk, which computes a mapping relation between a business process and a natural language storybook, which is described by a process designer. Jang et al. [7] find a Web service to answer a user’s query and provide the execution result of the service in a natural language sentence. The method of Al-Muhamed and Embley [8–10] is based on domain ontologies. It translates a natural language request into a predicate calculus form to represent the request in a formal way. Most of previous works can handle user requests with simple control constructs. While they find a mapping relation between a sentence block and a service, there might be a case where even a sentence block also corresponds to a composite service. Other than Web services, there have been researches regarding to natural language interfaces such as question and answering [11,12], conversational agents [13,14], databases [15,16], and program code generation [17,18].

In this paper, we propose a sophisticated method to extract an abstract workflow, which can be used to construct an actual composite service, from a user’s natural language request. Specifically, to deal with a request with a complicated control flow, the proposed method extracts candidate workflow templates by repeatedly applying basic workflow patterns. To find out the constituent services of a workflow, the proposed method determines the action and object of each sentence block based on its verb and nouns. The action and object of a sentence block are used to discover an appropriate service. Finally, the services discovered are mapped to the constituent tasks of an abstract workflow. Since an abstract workflow describes an overall flow of control among its constituent services, its concretization process, which is beyond the scope of this paper, is needed to execute the workflow.

To evaluate the performance of the proposed method, we experimented with a large number of user requests. The experimental results show that the proposed method extracts an abstract workflow from a complex request with a lot of control constructs, resulting in an accuracy of 95.2%. Particularly, if there is no service that matches a single sentence or a user’s simple request, the proposed method composes existing services into a composite one to satisfy the user request.

The organization of this paper is as follows. Section 2 describes how the proposed method publishes the descriptions of services based on proposed ontologies. Section 3 gives a detailed descrip-
tion of the proposed method to extract an abstract workflow from a user’s natural language request. In Section 4, the performance of the proposed method is evaluated and analyzed. Finally, the conclusions and future works are summarized in Section 5.

2. Description and publication of services based on ontologies

The proposed method finds mappings between a natural language sentence and existing services, which have been published at a service registry. From a given natural language request, the proposed method should find out what kind of services a user intends to use and what kind of parameters the services takes as input and returns as output. To do this, at the publication time of a service, its characteristics should be described semantically and the related natural language expressions should be registered. To identify input and output parameters from a natural language request, for each input and output parameter of a service, the relevant natural language expressions should be also registered. This section proposes Action and Object ontologies to describe the action and object of a service. An Input and Output (I/O) ontology for specifying the parameter types of a service is also proposed.

In this paper, services are described using OWL-S [19], which is a de facto standard ontology for describing the semantics of Web services. Additionally, the action, object, and parameters of a service are associated with appropriate concepts of the proposed ontologies. Here, we assume that the I/O ontology contains all the parameter concepts used by services. Likewise, the Action and Object ontologies represent the functions and objects of services, respectively. Furthermore, the Action and Object ontologies contain natural language representations for each concept, which are used to search for a concept that matches verbs or nouns within a user’s natural language request. Fig. 2 shows an example of publishing an OWL-S description of a FlightTicketSearch service. The service is mapped to the Find concept of the Action ontology and the Flight concept of the Object ontology. The dotted line from the Flight concept to the Find concept indicates that they have been used together to describe a service. The input and output parameters are also associated with the corresponding concepts of the I/O Ontology.

3. Extraction of an abstract workflow from a natural language request

To extract an abstract workflow from a natural language query, the proposed method goes through four phases as shown in Fig. 3. First, the proposed method divides a request into sentence blocks based on control constructs and verbs. It also extracts workflow templates by applying basic workflow patterns to control constructs. Second, for each sentence block, it determines its service type and a list of candidate services. Third, it selects a service from the candidate service list by calculating a similarity with the sen-
tence block. Finally, it generates an abstract workflow by combining the workflow templates and the services selected.

3.1. Extracting workflow templates

A user's request may involve the invocation of a number of services. The proposed method divides a natural language request into sentence blocks based on verbs and extracts workflow templates based on the sentence blocks and control constructs such as conjunctions.

Fig. 4 illustrates extracting workflow templates. A natural language processing software parses and analyzes a user's natural language request. We use the RASP system [20] as a natural language processor. It finds control constructs to determine workflow templates. Verbs are also found to select an appropriate service. A sentence may contain more than a verb. A conventional natural language processor can identify the types of verbs in a sentence such as main verbs, auxiliary verbs, and to-infinitives. The proposed method divides a natural language request into sentence blocks based on main verbs. Auxiliary verbs, which modify other verbs, are excluded. The request of Fig. 4(a) contains ‘if’, ‘then’, and ‘and’ as control constructs and ‘find’, ‘reserve’, and ’rent’ as verbs. The proposed method divides the request into sentence blocks based on control constructs and verbs and extracts a candidate set of workflow templates by repeatedly applying basic workflow patterns. The occurrence pattern of control constructs is compared with a list of basic workflow patterns to find out a workflow template like Fig. 4(b).

In the case of a user's request with a complicated control flow, alternative templates may be available. For example, the request of Fig. 5(a) contains nested loops and multiple coordinating conjunctions such as ‘and’ and ‘or’. Different workflow templates may be selected since the precedence and associativity rules of control constructs are not explicit in natural language sentences. Workflows (1) and (2) in Fig. 5(b) show an ambiguity problem when ‘if-then’ and ‘if-then-else’ appear repeatedly.
Fig. 5. Various workflows for a request. (a) An example of extracting sentence blocks based on control constructs and verbs. (b) Different workflow templates extracted from (a).

Workflows (3) and (4) show two different workflows when binary control constructs such as ‘and’ or ‘or’ appear sequentially in a sentence. Therefore, if there are multiple workflow templates, we need a process to determine the best match that reflects the intention of a user. The proposed method solves this problem by considering the semantics and structure of sentence blocks. The workflow selection process is described in Section 3.4.

3.2. Analyzing sentence blocks

A natural language query is just a character stream in itself. To understand its meaning, the proposed method computes the mapping relationships between the given words and ontological concepts. As mentioned before, each concept in an ontology has a list of its natural language representations. The action, objects, and parameters can be found by mapping the given words in a natural language query to the natural language representations of ontologies. This section describes how to extract a list of services from each sentence block as shown in Fig. 6.

First of all, the proposed method maps nouns and a verb in a sentence block to concepts of the Action and Object ontologies. If there is more than one object concept, the object concept that best represents a user’s intention is chosen. If a single service does not satisfy the function of the sentence block, the action concept mapped is extended to construct a composite service. As the result of analyzing the sentence block based on the Object and Action ontologies, the proposed method generates a list of services, which may consist of composite and/or unit services.

For example, in Fig. 7, verb ‘reserve’ is mapped to the Reserve concept of the Action ontology. Noun ‘ticket’ matches two concepts, i.e., MassTransit and Entertainment, in the Object ontology. If too many object concepts match a sentence block, the object of the user request becomes ambiguous and also the number of services to be investigated increases enormously. To select the concept that well
represents the object intended by a user, the proposed method tries to select the concept that covers as many object concepts as possible. For this purpose, the merge counter of a concept is defined to represent how many concepts the concept can cover. The object selection process is as follows. Specifically, the concepts related one another in the Object ontology are also considered (Step2). By considering the ancestor-descendant relation between the concepts mapped, a more specific one is selected (Step3). Finally, a more expansive and broad one is selected (Step4).

- **Step1:** Set the merge counter of the concepts mapped as 1 and the merge counter of the others as 0.
- **Step2:** Select the concepts referenced by the concept mapped, i.e., the value of attribute rdf:seeAlso, and increase the values of their merge counters. But do not duplicate this step on the same concept referenced.
- **Step3:** If two concepts have an ancestor-descendant relation, deselect the ancestor concept and increase the merge counter of the descendant concept as much as the value of the merge counter of the ancestor.
- **Step4:** If two different concepts have a common ancestor, select the ancestor concept and increase its merge counter as much as the sum of the two merge counters and deselect the descendants. However, if the common ancestor is the root concept, do not merge the two concepts.

Fig. 8 shows an example of selecting the object of a sentence block by applying the above object selection steps. Fig. 8(a) shows the concepts mapped from the sentence block of Fig. 7 and the initial values of their merge counters (Step1). The merge counters of the Airport and Flight concepts have the value of 2 since they are in reference relations with the concepts mapped (Step2). The Flight and MassTransit concepts have an ancestor-descendant relation and so the value of the merge counter of the MassTransit concept is added to the Flight counter and the MassTransit concept is deselected (Step3). Fig. 8(b) shows the final result of calculating the values of merge counters. The larger the value of the merge counter is, the more concepts the corresponding concept covers. The Flight concept comes to have the largest counter value and is finally selected as the representative object of the given sentence block. If more than one concept has the same counter value, they are selected as the representative objects.

A verb in a sentence block is associated with an action concept, which represents the function of a service requested by the sentence block. A list of services can be found by searching services, which are related with the action concept in a service registry. However, there might be a case where a single service that satisfies the user request does not exist.

Fig. 9 shows a sentence block, which requests a flight reservation service by using verb 'reserve'. A user provides the number of seats, a departure location, and an arrival airport. The proposed method should find a service, which takes this information as parameters. In the case of Fig. 9-(1), however, the single service does not fully use the information provided. The service of Fig. 9-(2) demands much more information than a user provided. To solve this problem,
the proposed method composes reservation and search services as shown in Fig. 10. In the case of Fig. 10-(1), while the number of the input parameters required increases, the information provided is fully used. In the case of Fig. 10-(2), a user should provide more information but the amount of the information decreases.

To extend an action concept, the mapping information between the Action and Object ontologies are used. An object concept maintains a list of the action concepts of services, which are associated with the object concept. In Fig. 11, the action list of the Flight concept contains the Reserve, Find, and Pay concepts of the Action ontology. A service list for the Reserve action concept may contain single services like FlightReservationService1 as shown in Fig. 9. By extending the action concept, two more action concepts, i.e., Find and Pay, are found. As a result, the service list is extended to contain composite services. For example, the proposed method composes FlightReservationService1 and FlightSearchService1 as shown in Fig. 10-(1).

3.3. Selecting services

From a list of services for each sentence block, the proposed method determines the best service that reflects the intention of a user. The proposed method analyzes parameters in a sentence block to calculate a similarity between a service and a sentence block. Specifically, the natural language representations of the I/O parameter concepts of a candidate service are compared with a sentence block.

In Fig. 12(a), three I/O concepts (Seat, DepartureLocation, and ArrivalAirport) are found in a sentence block by comparing the sentence block with the I/O concepts of its candidate services. When a word in a sentence exactly matches an I/O concept in a candidate service, this is denoted as ExactMatch. In the case that a word corresponds to a parent concept, it is denoted as PartialMatch. The Seat concept in Fig. 12(b) is an example of ExactMatch and the DepartureAirport concept in Fig. 12(c) is an example of PartialMatch. The similarity between a sentence block and a candidate service is computed by Eq. (1), where TotalInput means the number of the input concepts used by a service. OfferedConcept indicates the number of the concepts provided by a sentence block. EM and PM indicate the numbers of ExactMatch and PartialMatch, respectively. Here,
Fig. 10. An example of composing services.

The value of ServiceSimilarity defines how well the information offered by a sentence block is used by a candidate service. Without considering the information offered, a service with one parameter would have the highest similarity if the parameter corresponds to ExactMatch no matter how much information is contained in a sentence block. For example in Fig. 12, while the similarity value of the single service is 0.11 (EM: 1, PM: 0, TotalInput: 3, OfferedConcept: 3), the similarity value of the composite service is 0.54 (EM: 2, PM: 1, TotalInput: 5, OfferedConcept: 3).

On the other hand, the proposed method has limitations in handling natural language expressions. Given a negative sentence like “I reserve a flight ticket from LA to NY not on September 11,” the proposed method can find a flight reservation service, which takes a parameter of a conventional date datatype as input. Meanwhile, in order to fulfill the user request, the part of “not on September 11” should be precisely interpreted and delivered to the corresponding service as an input parameter. Furthermore, a flight reservation service, which takes an input of a negative date datatype rather than a conventional date datatype, should be selected. Particularly, this kind of precise information will be helpful for the concretization process that is needed to execute the workflow. At this time, our method does not handle a negative sentence properly. While Fig. 9 is to illustrate the proposed method, strictly speaking, Block4 remains ambiguous in terms of transport mode. The user intention might be a train reservation service rather than a flight reservation service. The ambiguity of natural language expressions may cause different interpretations from a user intention. Therefore, we need a method to clarify what a user really wants through an additional user interaction.

3.4. Generating an abstract workflow

This phase combines the services selected and the workflow template extracted to generate an abstract workflow. If more than one template were extracted, the most suitable one that reflects a user intention should be determined. Specifically, by considering the semantic or structural balance of the constituent sentence blocks, the proposed method selects the most balanced workflow template.

Fig. 13 shows examples of workflow templates, which may be generated from an input request with ‘and’ and ‘or’. The request can be interpreted into two different workflow templates depending on alternative grouping of conjunctions. However, it is more reasonable to execute the choice group of Block4 and Block5 in Fig. 13(b)-(2) after executing Block3 due to the fact that Block4 stands on equality with Block5 conceptually.

Fig. 14 shows another example, for which multiple interpretations are possible. In this example, the request can be interpreted into more different ways because it includes more conjunctions. If we consider structural balance, a possible interpretation would be Fig. 14-(1), in which the group of Block1 and Block2 is executed or the group of Block3 and Block4 is executed. If control constructs are used to connect similar sub-workflows, it can be considered that a user wants for sub-workflows to be executed in balance.

Based on the above observation, sub-workflows, which are connected through ‘and’ or ‘or’, are compared, and a proper interpretation is made so that the connected sub-workflows have similar meaning or structure. Likewise, ‘then’ and ‘else’ clauses (or ‘if’ and ‘then’ clauses) of the if-then-else structure often have similar meaning or structure. Therefore, in the case where ‘if-then’ and ‘if-then-else’ appear in a nested form, sub-workflows of ‘if’ and ‘else’ (or ‘then’ and ‘else’) are compared by considering their semantic and structural balances.

SubWorkflowSimilarity = StructureSimilarity + ActionConceptSimilarity + ObjectConceptSimilarity
\[ \text{ConceptSimilarity} = \frac{EM \times 1 + PM \times 0.7}{\text{NumberOfConcepts}} \]  

The similarity between sub-workflows is calculated by Eqs. (2) and (3), where NumberOfConcepts indicates the number of the object or action concepts mapped with sentence blocks in two sub-workflows. The number of exactly matching concepts between two sub-workflows is denoted as EM. PM indicates the number of concepts, which are in parent–children or sibling relations. If two sub-workflows have the same structure, the structure similarity is set to 1. Otherwise it is set to 0.

Fig. 15 shows two workflow templates, which may be extracted from the request of Fig. 5. It also shows the similarities between sub-workflows calculated by Eqs. (2) and (3). In Fig. 15-(1), the ‘then’ and ‘else’ clauses within the nested ‘if-then-else’ have the same structure and action concepts. Among the three object concepts, two of them were matched. The similarity between the two sub-workflows comes to 2.67 (StructureSimilarity: 1, ActionConceptSimilarity: 1, ObjectConceptSimilarity: 0.67). In Fig. 15-(2), the ‘then’ and ‘else’ clauses at the first level have different structures. Their similarity comes to 1.25 (StructureSimilarity: 0, ActionConceptSimilarity: 0.75, ObjectConceptSimilarity: 0.5). The total sum of the similarity values is used to evaluate the balance of a workflow template and the one with the highest value is selected. The workflow template selected is combined with the services selected from its constituent sentence blocks to generate an abstract workflow as the final result of the proposed method. For example, the proposed method extracts the abstract workflow of Fig. 1(b) from the request of Fig. 1(a).

4. Experimental results

To evaluate the performance of the proposed method, the proposed method was experimented with 127 natural language queries and 312 services. Generally, the performance of a method in semantic Web research is highly dependent on how well ontologies are organized. So in this paper, we experimented under the assumption that the proposed ontologies (Action, Object, and IO) were well organized. Experimental data was prepared as follows. First of all, we constructed a total of 127 natural language queries, which consist of 50 simple queries collected from the Web and 77 complex queries created manually based on the simple queries. While we played a major role in preparing the test query set, a few more people were also involved.

<table>
<thead>
<tr>
<th>Complexity</th>
<th># of queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>50</td>
</tr>
<tr>
<td>Medium</td>
<td>46</td>
</tr>
<tr>
<td>High</td>
<td>31</td>
</tr>
<tr>
<td>Total</td>
<td>127</td>
</tr>
</tbody>
</table>

Tables 1 and 2 show the classification of queries by their complexity and domain, respectively. Queries were designed and classified into three categories by complexity: low, medium, and high. The low category contains queries with 1 or 2 sentence blocks and the medium category consists of queries with a control construct and 2 or more sentence blocks. The high category includes queries with 2 or more control constructs and 3 or more sentence.
I want to book a train ticket to Amsterdam from the Cologne station and reserve a hotel from the 18th August to the 22nd August or reserve a flight to Amsterdam from the Cologne and reserve a guesthouse from 18th August to the 22nd August.

(a) An example of extracting sentence blocks based on control constructs

(b) An example of multiple interpretations.

Fig. 14. Selecting a workflow template by considering the structural balance of sentence blocks. (a) An example of extracting sentence blocks based on control constructs. (b) An example of multiple interpretations.

Table 2
The classification of sentence blocks by domain.

<table>
<thead>
<tr>
<th>Domain</th>
<th># of sentence blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accommodation</td>
<td>61</td>
</tr>
<tr>
<td>Book</td>
<td>17</td>
</tr>
<tr>
<td>Camera</td>
<td>7</td>
</tr>
<tr>
<td>Car</td>
<td>28</td>
</tr>
<tr>
<td>Computer</td>
<td>4</td>
</tr>
<tr>
<td>DVD</td>
<td>21</td>
</tr>
<tr>
<td>Mass Transit</td>
<td>56</td>
</tr>
<tr>
<td>Movie</td>
<td>8</td>
</tr>
<tr>
<td>MP3Player</td>
<td>13</td>
</tr>
<tr>
<td>Music</td>
<td>25</td>
</tr>
<tr>
<td>Photo</td>
<td>2</td>
</tr>
<tr>
<td>Route</td>
<td>14</td>
</tr>
<tr>
<td>Shipping</td>
<td>3</td>
</tr>
<tr>
<td>Traffic Information</td>
<td>2</td>
</tr>
<tr>
<td>Translation</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>262</td>
</tr>
</tbody>
</table>

blocks. Table 3 shows the total number of control constructs in the queries.

Additionally, we created a test set of services, which contain a total of 312 Web services. Specifically, in addition to the services that satisfy the test query set, the service set contains additional services, which were created varying the combinations of the actions and objects used by test queries. By varying the number of I/O

Table 3
The number of control constructs in queries.

<table>
<thead>
<tr>
<th>Constructs</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>If-then-else</td>
<td>0</td>
<td>7</td>
<td>17</td>
<td>24</td>
</tr>
<tr>
<td>Sequence (‘and’)</td>
<td>0</td>
<td>9</td>
<td>34</td>
<td>43</td>
</tr>
<tr>
<td>Choice (‘or’)</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>Split-join</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>Repeat</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>Any order</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td>0</td>
<td>46</td>
<td>83</td>
<td>129</td>
</tr>
</tbody>
</table>
parameters, we also created services, which take more (or less) parameters than those of test queries. Finally, we constructed the proposed ontologies so that they could support the semantic description of the service set. Specifically, the Action, Object, and I/O ontologies consist of 15, 46, and 92 concepts, respectively. The natural language representations attached to the ontologies were manually selected. To identify I/O parameters from a natural language request, the I/O ontology contains regular expressions as well as words. The numbers of the words or regular expressions in the Action, Object, and I/O ontologies come to 40, 101, and 192, respectively.

4.1. Performance analysis

The abstract workflows manually constructed by human experts were compared with those generated by the proposed method. Table 4 shows the experimental results.

The erroneous cases in the experiments can be classified two categories. First is the case where an incorrect service was discovered for a sentence block. To match a sentence block with a relevant service, the proposed method should analyze a service type requested by the sentence block. However, if the method cannot analyze the service type, it is impossible to compute the matching relationship between the sentence block and its corresponding service.

In our experiments, the natural language words attached to the Object ontology were designed to contain thematic keywords. For example, the Car concept includes thematic keywords such as ‘car’, ‘vehicle,’ and ‘sedan.’ However, given a sentence like “rent an Impala,” the proposed method could not recognize that the object requested was a car. This was due to the fact that the natural language representations attached to the ontologies were limited. To resolve this problem, the ontologies should be connected with conventional lexical databases such as WordNet [21].

Second, the proposed method extracted a workflow template, which was different from the intention of a test query. Let us illustrate the second erroneous case using the following requests.

Fig. 16 shows the workflows generated from the above user requests. Request1 is the result workflow for the first request. This workflow reflects the intention of a user. However, in the case of the second request, even though Request2-1 is more suitable, the proposed method generated the workflow of Request2-2. This is because the proposed method is based on the balance of a workflow, which is computed by the similarities between sentence blocks. The proposed method evaluated the similarity between Rent_Car and Reserve_Flight low, but rated the similarity between Reserve_Flight and Reserve_Hotel high because they used the same verb ‘reserve’.

The proposed method extracts all possible workflows from a user request and then selects the most suitable workflow by calculating similarities between sub-workflows. The more control constructs a request has, the more workflow templates the proposed method extracts. This causes a performance problem when the proposed method extracts a workflow from a request, which has many control constructs. As shown in Fig. 17, the number of workflow templates rapidly increases as the number of control constructs in queries grows.

4.2. Comparison with previous works

Table 5 shows the qualitative comparison result of the proposed method with previous works. The conventional methods extract workflows from a simple sentence block and cannot handle a complex logic with nested control constructs.

While most of the previous works take a natural language request as input, Englmeier et al. [6] restrict an input query to a controlled vocabulary. Bosca et al. [2–4] extract a workflow from a request with a single control construct such as ‘if-then’ or ‘sequence’. While the method of Xie et al. [5] does not extract a workflow from a sentence, it makes a workflow by connecting sen-
Table 5
Qualitative comparison with the previous works.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Al-Muhammed and Embley</td>
</tr>
<tr>
<td>Natural language</td>
<td>O</td>
</tr>
<tr>
<td>Workflow extraction</td>
<td>X</td>
</tr>
<tr>
<td>Nested/repeated control constructs</td>
<td>X</td>
</tr>
<tr>
<td>Multiple sentence</td>
<td>O</td>
</tr>
<tr>
<td>Composite service for a sentence block</td>
<td>X</td>
</tr>
<tr>
<td>Domain independence</td>
<td></td>
</tr>
</tbody>
</table>

* O: supported, Δ: partially supported, X: not supported.

Table 6
Performance comparison with the method of Bosca et al.

<table>
<thead>
<tr>
<th>Complexity</th>
<th># of wrong services</th>
<th># of wrong workflows</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The proposed</td>
<td>Bosca et al.</td>
<td>The proposed</td>
</tr>
<tr>
<td>Low</td>
<td>2</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>Medium</td>
<td>1</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>High</td>
<td>2</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>5</td>
<td>–</td>
<td>1</td>
</tr>
</tbody>
</table>
tences. While Englmeier et al. do not extract a workflow from a sentence, they explicitly represent a control structure by using a controlled vocabulary. On the other hand, the proposed method extracts a workflow from a complex natural language request, which contains nested control constructs and multiple sentences.

Unlike most of the previous works, when a user's request cannot be satisfied by a single service, the proposed method composes services to satisfy the user's intention. Xie et al. also provide composite services by extending objects. For example, if a user requests a travel schedule service, their method extends the request to services such as 'flight reservation', 'accommodation reservation', and 'car rental'. However, the user request cannot be satisfied if there is no single service, which matches the extended individual services.

Table 6 shows the result of performance comparison of the proposed method with the method of Bosca et al., which is considered as the most comparable work with ours since they generate an abstract workflow from a user's natural language request. The experimental result on the same test data shows that the proposed method is more accurate than the method of Bosca et al. This is due to the fact that Bosca et al. only consider unit services and cannot handle complex requests with nested control constructs.

Fig. 18 shows the difference between the proposed method and the method of Bosca et al. The Buy service, which is selected by the method of Bosca et al., does not take a publisher as input, which is provided by the user request. The proposed method, however, connects the Buy service to the Find service, which has a relation with the Buy service, to take as much information as possible from the request.

5. Conclusions and future works

With the wide spread of Web services in various fields, there is a growing interest in utilizing a composite service. However, it is difficult to explicitly specify how to compose services. We need a convenient user interface, which can be used by a novice user who does not know Web service technologies. A natural language interface is a user-friendly and convenient interface to invoke services.

In this paper, we proposed an efficient method to extract an abstract workflow, which can be used to construct an actual composite service, from a user's natural language request. Specifically, to deal with a request with a complicated control flow, the proposed method extracts candidate workflow templates by repeatedly applying basic workflow patterns. To find out the constituent services of a workflow, the proposed method determines the action and object of each sentence block based on its verb and nouns. The action and object of a sentence block are used to discover an appropriate service. Finally, the services discovered are mapped to the constituent tasks of an abstract workflow. Experimental results with a variety of natural language requests show that the proposed method successfully extracts abstract workflows, resulting in an accuracy of 95.2%. Particularly, if there is no service that matches a single sentence or a user's simple request, the proposed method composes existing services into a composite one to satisfy the user request.

As discussed before, the proposed method has limitations in handling natural language requests including negative sentences. The ambiguity of natural language expressions may also cause different interpretations from a user intention. Therefore, we will develop a more sophisticated method to process natural language expressions to identify what a user really wants. Our future works also include exhaustive evaluations of the future version of the proposed method on a large volume of natural language expressions and a real world Web services database.

The proposed method considers every possible combination of workflow templates. Therefore, we have a plan to enhance the proposed method to deal with a user's request with a number of control constructs more efficiently. As mentioned before, an abstract workflow describes an overall flow of control among its constituent services. To invoke services, an abstract workflow should be concretized to an executable workflow. The concretization process requires binding among the different types of parameters of constituent services. Our future work includes the development of a service mediation framework, which plays a role of bridging the information mismatch among services.

Acknowledgements

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (R01-2006-000-10864-0(2007)).

References


