Semi-automatic Context-aware Composition of Context-aware Semantic Web Services

Iman Keivanloo, Hassan Abolhassani
Web intelligence Laboratory, Sharif University of Technology
keivanloo@kish.sharif.edu, abolhassani@sharif.edu

Abstract—In many cases, atomic Web services are not able to provide requested functionality. Therefore, the composition task is required on available atomic services. In addition, Context awareness in Web services is gaining momentum as the opportunity to benefit the interactions between human, applications and the environment. So, it seems reasonable to use available context information in composition procedure to provide a composite Semantic Web service which is adapted to the context of participant Web services and user. In this paper, we introduce a novel approach for semi-automatic context-aware composition of stateless context-aware Semantic Web Services. One of the main characteristics of proposed approach is the ability of considering the degree of context-awareness of the service in the composition procedure. Moreover, we use a new schema for Semantic Web service description which is able to represent both functional and non-functional information about context-aware Semantic Web services.

Index Terms—Composition, Context, Semantic Web Service, Service Description.

I. INTRODUCTION

NOWADAYS Web service plays an important role in software development area [1]. In view of the fact that it can be used in small and large scale environments, many different entities participate in each interaction. A lot of information, which is related to the interaction, is available around these entities. This kind of information is called context [2]. In some cases, context information is used to provide outputs, and it is called context-aware system [3].

Context-awareness in Web services is not a new field of research and many researches have been done not only because of providing better outputs but also it seems that context-awareness is strongly required. For example, [4] enumerates many reasons for being context-aware in Web service related interactions. Being aware of available resources is a simple example of context-awareness like the one that proposed in [5]. Another example is using context information in Web service discovery [6]. These instances show the need for context-awareness in Web service environments.

Since Web Services are loosely coupled software components that are published, located and invoked across the web. Currently, many services are offered on the web and their number would increase exponentially in the future. But these atomic services may not be able to satisfy specific requests individually. So, there must be brokers between service providers and service requesters which integrate the atomic services of service providers and create new value added composite services.

As a result, many entities are participating in each interaction (e.g. users, agents, brokers and composers). Each of them can use context information from other sources in its procedures; for example a composer can use the user’s preferences available in his profile to provide an appropriate composite service like [7]. But creating service compositions is a time consuming, error-prone task. The aim of this work is to exploit available context-related information around the users and Web services to provide a method for semi-automatic composition of stateless information providing web services. The main issue is that there is a difference between context-aware composition of Web services and composition of context-aware Web services. Until now some context-aware composers for Web services has been proposed like [7] but there is no composer for context-aware Web services. Therefore in this paper, we propose a composer for context-aware Semantic Web services which itself is context-aware too.

In order to make an applicable context-aware composer, appropriate context representation is a necessary requirement. Therefore, we provide a new context model for representation of context-related issues about users, environment and Web services. Therefore, the context information is split into three dimensions which are user context, environment context and Web service context.

On the other hand, since we are going to make a composer for context-aware Web services, the context-aware Web services have to represent the degree of their context-awareness in a formal manner. Degree of context-awareness for Web services is defined as the kinds of required context information and the types of dependency between the service and context information. Therefore we proposed a new approach for context-aware Semantic Web service description. As a result, the services which are deployed using the proposed description language will be capable of showing the required context information in addition to functional inputs.
At last formal description of context-aware Semantic Web services provides us a basis for composition of context-aware Semantic Web services. This type of composition selects participants regarding the ability of the application to provide context information. Moreover, the provided context model helps the composer to be context-aware similar to previous researches (e.g. [7]).

Therefore, when the component receives the user’s request, it is responsible to make a composite service from available context information and available context-aware Semantic Web services. One of the main reasons to propose this solution is availability of different context-aware Semantic Web services providing similar output, but their required context information is different. Therefore the composer will be able to select the appropriate one based on available context information of the user. In addition, inasmuch as the context information provider and consumer are different, some processing on context information may be required like type conversions. So, available Semantic Web services in the registry can be used for this purpose too.

This paper is organized as follows. In section 2, we will have an overview on background information and related work for both Semantic Web services and context-awareness issues. Section 3 describes the context model, and section 4 presents the proposed service description approach. In section 5, the solution to make context-aware composition of Semantic Web services is introduced. In the two last sections, evaluation and conclusion are presented.

II. RELATED WORK

In this section, related literature to context-awareness, Web services and service composition is reviewed in different parts. For Web services composition, both basic issues and context-aware researches are presented separately.

A. Semantic Web Service Description

WSMO [8], OWL-S [9] and SWF [10] are some of famous languages for service description. One of their main characteristics is ability to represent functional parameters. For example, ServiceProfile of OWL-S can be used by the service provider to represent the input and output of the service using hasInput and hasOutput properties. In addition, it lets us specify the preconditions and effects of the service. Moreover, Non-Functional information about the service can be shown using serviceParameter section. However, there is no way to represent the required context information for context-aware Web services explicitly.

B. Context-aware Web Service Composition

AI planning-based and workflow-based techniques are two major categories of Web service composition. In addition to the two previous categories, some other composition techniques are proposed. One of them is used in this research owing to its properties [11]. It proposed a graph-based composition algorithm for stateless Web services. Additional details about this algorithm will be given in next sections.

As it is described before, by participating context information in composition procedure, the output can be adapted regarding the context of user, Web services and environment. In this area, policy and QoS are two addition topics which are sometime manipulated as context or beside it, and some researches focused on one of them as the context-awareness issue.

[12] integrated services’ workflows to realize a target user task enabling context-awareness for pervasive environments. The composition is done by mapping the problem to finite state. In [13], they also concentrated completely on QoS aware composer. [14] proposed a matching framework for Web service composition using context information. Authors proposed a context categorization based on ontology concept and a two-level context modeling. The first level is related to Web service specific context, and the second level used for context policy modeling. The only research the aim of which was the implementation of a real context-aware Web service composer is [7]. It proposed a context-aware planning method which comprises global planning and local optimization based on context information.

Although their architecture is context-aware, their context modeling part is poor and very simple. Another issue is that their architecture is developed based on predefined composition plan which are available on library; therefore it is not fully automatic and also suffers from AI planning limitation which are discussed in literature.

III. CONTEXT MODEL

Many types of context information are available around each interaction between a user and related service provider. They are often categorized into three sections which are user context, service context and environment context. Although, most of the context-aware Web services related entities (e.g. composer) use two first types of context, in this research we model all kinds of context using the extended version of CO4WS [15] ontology (Context Ontology for Web Service).

There are two reasons to use it in this research. The first one is the evaluation result of CO4WS which is available in [15]. CO4WS also has all the requirements listed in [3] for a context ontology. Therefore, it can be a good candidate for context representation in a context-aware system. The extended model is able to represent the general and domain specific context of the user, environment and the Web services.

IV. SEMANTIC WEB SERVICE DESCRIPTION

Context-aware Semantic Web services need to access some part of the context information, in order to adapt themselves. Many approaches were proposed to make context information available for them. Two common approaches are context API and passing context information via functional input
parameters to the services. The second approach is not an appropriate solution because of reasons which are listed in [16]. On the other hand, the first one is very flexible but has some drawbacks. One of them is the arbitrary context usage. It means the context-aware Semantic Web service can use context information via context APIs and nobody knows it will use which type of context information. Any information about the type of context-awareness of the service (degree of context-awareness) is an important issue which can be helpful in service discovery, selection and composition. Therefore we propose a solution in which the context-aware Semantic Web service will be able to use context information via context APIs (we call them Context Provider Web service). It also prevents arbitrary context usage by specifying the required context information in service description. We exploit this approach to make the composer aware about services’ required context information and the ability of providing required information by users’ application.

Using OWL-S [9], a single type of input can be used to specify the parameters of services. But as described before, we need to specify input context information beside the functional inputs. So we extend OWL-S in a way to be able to represent these types of input:
- **Functional input**
- **Context input**

Using the first type in Web service description, a common type (Functional) of inputs will be defined. Required context information will be specified using the other type of input which is Context input.

It is clear that sometimes the context-aware Web service will be able to continue its job without requested context information specified in its service description using proposed approach. Thus, it will cause some side effects for service selection progress if it does not consider this issue. So, we add another property to the Context input. The new property shows whether it is mandatory to provide this context input for context-aware Semantic Web service. Therefore this property will be used for service selection procedure. At last, we will have three types of input parameters which help us describe context-aware Semantic Web services in a precise manner.

As an example, in Fig. 1, a service description based on proposed approach is shown. The original service description is available in SWS-TC 1.1 [17]. In the service description, it is obvious that the service requires two context inputs. One of them is used optionally to sort hospitals based on distance, and another one which is mandatory is used to provide outputs according to user’s device.

We believe context-aware Semantic Web service developer should use this technique for service description. Due to the fact that he knows how to categorize service inputs in these proposed classes and which of context inputs are mandatory. However, it is possible to convert conventional OWL-S service descriptions to the proposed service description model by automatic software for previously published services.

The only other issue that is needed to be described is the way of using CO4WS with Web service description languages. We suggest the following solution (although other approaches are applicable like [18]), because the result is backward compatible which is a very important measure. In order to implement this solution, ServiceParameter which belongs to Profile section of OWL-S can be used. ServiceParameter is prepared for non-functional requirement representation, so it is best candidate for this purpose. An abstract view is shown in Fig. 2.

V. CONTEXT-AWARE COMPOSITION

In this section, we are going to introduce the proposed context-aware composer for context-aware Semantic Web services. It is necessary to mention that although this composer is a context-aware one; in addition, it is designed for context-aware Semantic web services. As in the previous section was described, by using proposed approach for service description, some new type of information (about the way that service is context-aware) will be available for other entities like the composer. Therefore using this kind of information will be helpful for the composer as described later.

A. Prerequisite

This composer is established based on late binding property [19]. It means that the available services in the repository are

```xml
<profile:Profile rdf:ID="GetCityHospitals-Profile">
  <service:isPresentedBy rdf:resource="#GetCityHospitals-Service"/>
  <profile:serviceName xml:lang="en">Get City Hospitals</profile:serviceName>
  <profile:textDescription xml:lang="en">Returnds the hospitals in a city</profile:textDescription>
  <profile:hasMandatoryContextInput rdf:resource="#CityInfo"/>
  <profile:hasOptionalContextInput rdf:resource="#DeviceInfo"/>
  <profile:hasOutput rdf:resource="#Hospitals"/>
</profile:Profile>
```

Fig. 1. Sample of Service description using proposed approach

```xml
Web service
  └── ServiceProfile
      └── ServiceModel
          └── ServiceGrounding
              └── CO4WS Concept
```

Fig. 2. Using CO4WS concept beside OWL-S.
categorized in some abstract services based on their functionalities and inputs. The way that the services are categorized is not related to the composer therefore we do not describe the available solutions here. As a result, we have a several abstract services which each of them contains some real semantic Web services. It is obvious that the instances of each abstract service should have similar functional and context inputs.

B. Proposed algorithm

In our proposed algorithm for context-aware composition of context-aware stateless Semantic Web services, we have adopted and extended the composition algorithm for stateless Web services which is introduced in [11]. Therefore that algorithm is the basis for the proposed one here and several changes and extensions are described in the following in order to make it context-aware and also adapt to context-aware stateless Semantic Web services.

First, a directed graph, called dependency graph, is used to store information about input/output (both Functional and Context) parameters of services. Three different kinds of nodes exist in this graph structure: Service nodes, Functional Datatype nodes and Context Datatype nodes. Service nodes are representatives for abstract services which are stored in the service registry, while Functional Datatype nodes are representatives for datatypes that are generated by these services as functional output or consumed by these services as functional input. The last one which is Context Datatype nodes are representatives for required context information as Context Input.

The edges in the dependency graph are used to show dependencies between services and datatypes. If service $S$ has an input of type $I$ then there would be an edge from node $I$ to node $S$ in the dependency graph. On the other hand, if service $S$ has an output of type $O$ then there would be an edge from node $S$ to node $O$ in the dependency graph. It is obvious that each type of input (Context or Functional) has its specific type of edge. In addition to the aforementioned edges, another kind of edges also exists in this graph which is represented using dashed arrows and shows the dependency between datatypes. An edge from datatype $D1$ to datatype $D2$ states that $D1$ is a sub class of $D2$.

Formally speaking, the dependency graph can be represented using the seven-tuple which are shown in (1).

$$ DG = (SN, DF, DC, IF, IC, OE, SE) $$

$$ IF \subset DF \times SN $$

$$ IC \subset DC \times SN $$

$$ OE \subset SN \times DF $$

$$ SE \subset DF \times DF $$

Where, $SN$ is the set of service nodes, $DF$ is the set of functional datatype nodes, $DC$ is the set of context datatype nodes, $IF$ is the set of functional input edges, $IC$ is the set of context input edges, $OE$ is the set of output edges and $SE$ is the set of edges that represent subclass of relationships between datatypes. It is necessary to mention that output edges and subclass edges are only defined and used for functional inputs.

For each composition request the composer will make a dependency graph based on available abstract services and data types. After this step, for each node and edge a weight will be calculated and after that based on their weights participants will be selected. The entire scenario is split into three levels which are described in the following.

In first level, dependency graph will be produced and weighted similar to original algorithm with the exception of context nodes. In a context-aware environment always context information is not available in the needed format and datatype. For example suppose that the location information (which is a context datatype) of the user is available as a postal address field but some context-aware Semantic Web services needs location information as a postal code. In order to solve this kind of situations, in the proposed algorithm the link between location context node and those context-aware Semantic Web services will be established and in next levels if it is needed, those conversions and value-added tasks will be applied. Moreover, as described before, the proposed description method for context-aware Semantic Web services has the ability to represent optional context inputs besides mandatory ones. Therefore, this kind of information will be useful for next steps. So those context input edges which are optional for each service are labeled as optional edges.

Next the algorithm should select the participants. In this section, the main difference with the original algorithm is the way that the new one behaves with context nodes. Original algorithm has a list which contains provided (from user) or available datatype nodes. Hence, Based on them possible service nodes will be examined and the one that has the highest weight will be added to the participants list and so on (more details about participant selection are available in [11]).

Since here we have two types of context input edges which are produced in previous step, the original algorithm is changed in a way to exploit this information too. In a situation that the composer should select the next participants among candidates, the main parameter, which is used in decision making, is the weight of services but another issue is the optional and mandatory context inputs of them. So the proposed composer first eliminates those services which their required mandatory context inputs are not available. Then it gives a rank to each candidate based on the number of required available optional context inputs and its weight. Ranking for candidates is calculated by using (2) where $m$ is the number of mandatory context inputs for the candidate $c$, $a$ is the number of provided optional context inputs for $c$ and $t$ is the total number of required optional context inputs for $c$ and Weight is the original algorithm’s rank. This formula is achieved from experimental results and it is not mandatory and can be replaced by any other ones.

Until now the composer selects the participants in the composite service and this is the end of first level. In this step, an expert or who has enough knowledge about the functionality of requested composite service is needed.
Sometimes he can be the user and it depends on the type of application and the way that it is developed. However, he should check the participants and their relations in order to validate the functionality of composite service against requested one. If it is acceptable the next step will be applied. We call this algorithm a semi-automatic one because a graph-based algorithm is not always able to provide a valid composite service and therefore a supervisor is required.

$$\text{ContextRank}_i = \begin{cases} \frac{m + a}{t} & \text{if } (t \neq 0) \\ 0 & \text{if } (t = 0) \end{cases}$$  \tag{2}

$$\text{TotalRank}_i = (\alpha \times \text{Weight}_i) + ((1-\alpha) \times \text{ContextRank}_i)$$

In the second step, for each context datatype node which participates in the composite service, a sub composition task will be started. The reason which we need this step is described before in this section. Since it is a simple composition task the original algorithm suffices. Therefore after this step all required context information will be available in appropriate type and format for composite service’s participants.

After this step a composite service which satisfies the application’s request is available and can be used in the application development progress. But it is obvious that these steps most of the time are done in an offline manner and in the time of application development and there can be a long gap between these steps and online usage of it. So the context of users and services will be varied from the composition time and usage time and this is the main reason which we emphasize that the participants in the composite service should be abstract services. In other words, the services and the composer are context-aware against context of user, service (real participants not abstract ones) and environment therefore it is not rational that using real services as participants because who knows the context of the interaction in advanced (in development time). So the composer has to wait until the usage time to retrieve context of the interaction (user, real context-aware Semantic Web services and environment) and then select the real services instead of each abstract participant based on the current context. And this is the last level which we call it context-aware service selection.

For context-aware service selection many approaches are available like [20]. And any of them can be used for this level which the third one. Therefore when the user requests the execution of the composite service (which is produced and validated before), the composer use the context-aware selection algorithm to find out real participants in the composite algorithm based on current context and pass them as a composite service to the application for execution. The pseudo code of the level 1 and 2 of the proposed algorithm is shown in Fig. 3.

At last, a simple example is given in the following. It is about a Movie suggestion scenario. In this scenario, an application wants to suggest a cinema to its user which it is showing the most popular movie of the week. In the Fig. 4 the first level of the composition is shown. Datatypes are shown by oval (context datatypes’ captions are bold) and context-aware services are shown by rectangular.

$$\text{GenerateComposition}:$$
- CreateDependencyGraph
- SetValues(requestedOutputs)
- addTokens(availableFunctionalInputs, availableContextInputs)
- while (existsNotGeneratedNode(requestedOutputs))
  - Find fireable services
    - if there is no fireable service
      - return “Failure”
    - for each of the services in fireable services
      - if required “mandatory context inputs” are not provided
        - Remove service
      - Rank fireable services
        - sn = fireable service with the greatest rank
        - increment number of fires of sn
        - produce output tokens of sn
        - Add sn to composition
      - if sn generates one of the requested outputs
        - remove the outputs of sn from requested outputs
        - SetValues(requestedOutputs)
- Validate(composition)
- for each of the context input nodes in composition
  - Compose(ProvidedContextNode, RequiredContextNode)

Fig. 3. pseudo code of the two first levels of the proposed algorithm

Since in this scenario the application is only able to provide location context information therefore the FindNearestCinema2 abstract service can not be used and as a result, the composer chooses BestMovie and FindNearestCinema1 as the participants. After validation in the next level a composition on location datatype is needed. Since FindNearestCinema1 needs the location information is postal code and the application provides it as longitude and latitude the composition in Fig. 5 will be applied.

It is obvious that in the real composition the dependency graphs are more complicated because of the number of abstract services and datatypes is high but in this sample we have just indented to show a simplified one.
VI. EVALUATION

For evaluation phase first, we have applied the algorithm on SWS-TC 1.1 [17], which is a test collection of Semantic Web Services (non-context-aware). As it is expected the result of the proposed algorithm and the original one has not any difference because in the case of having no context information both of them must work similarly.

In addition, in order to evaluate the proposed algorithm on context-aware service repository, since there is no such a repository we have converted SWS-TC using the proposed service description approach and after that apply the proposed algorithm on it. All the cases it was able to provide an applicable output based on required functionality and available context information. However since it is the first composer for context-aware Semantic Web services, there is no any previous result for comparison and evaluation. Therefore we have just evaluated its result by the applicability parameter. Applicability shows that whether the output of the algorithm is applicable in the application. Since it is obvious, the result shows that the composer will return an applicable output always if there is any.

The overall evaluation contains about 150 context-aware Semantic Web services which are described with the proposed language. These services can be categorized in 4 domains which are medical care, education, travel and economy. For all of the composition tasks, it returns at least one answer (if it is possible) which matches to the current context. We have tested about 10 query for each domain and composer (on a regular desktop computer) responses in less than a second for all of them. Finally, it is needed to mention again that the result of the proposed composer in this paper can not be compared with any other one, because it is the first one of this kind of composer. Therefore we have just tried to evaluate the soundness of the proposed approach in this phase.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented a context-aware composer for context-aware stateless Semantic Web service which is the first composer for context-aware services. Therefore, one of the unique characteristic of it is the ability of considering the degree of context-awareness of the services. It means that by using proposed description method for context-aware Semantic Web services, it is able to choose an appropriate participant based on available context information. In addition, it is a context-aware composer too. So, it is based on late-binding property, in the runtime according to the current context of the service, user and environment (which are represented using the described context model) the real participants will be selected. We believe that for graph-based context-aware composition, late-binding property should be adopted. Because this kind of composition is not able to guarantee the validity of its output, therefore always there is a gap between composition and usage (since it needs a human to validate the output).

In our future research, we will work on the previously proposed improvement on the original composition algorithm in order to apply them on our composition algorithm.

REFERENCES