A Platform for Discovery and Execution of Semantic Web Services Compositions

Guilherme C. Hobold  
Department of Informatics and Statistics  
Federal University of Santa Catarina  
Florianópolis, SC – Brazil  
gwic@inf.ufsc.br

Frank Siqueira  
Department of Informatics and Statistics  
Federal University of Santa Catarina  
Florianópolis, SC – Brazil  
frank@inf.ufsc.br

Abstract— Semantic descriptions provide more accurate information related to operations supported by Web services, enabling their dynamic discovery and execution without human intervention. Furthermore, semantic descriptions allow Web services to be automatically combined by using discovery mechanisms able to identify composed services. These compositions can also be described and published as if they were a single service, allowing service consumers to discover and invoke a composition transparently. This paper presents a platform for automatic discovery and execution of semantic Web services compositions. A composer mechanism identifies semantic Web services compositions based on the information annotated on service descriptions using SAWSDL (Semantic Annotations for WSDL). The identified service compositions are described and published as a single service. The interaction among services is described using WS-BPEL (Web Services Business Process Execution Language) and executed by a BPEL engine.

Keywords- Semantic Web Services, Web Service Composition, Web Service Discovery, SAWSDL.

I. INTRODUCTION

The current hype around SOA (Service-Oriented Architecture) and SaaS (Software as a Service), combined with the ubiquity of the World Wide Web, have contributed for the widespread availability of Web Services. This technology, despite enabling the integration of computing systems in heterogeneous environments, still requires human intervention during the integration process, given that only humans are able to infer the semantics behind data and operations provided by different services.

In this context, combining two or more services is often necessary in order to accomplish a required task. However, the assembly of service compositions is not favored by the syntactic nature of the description language employed for service description – i.e., WSDL (Web Services Description Language) [1] – which limits itself to associate keywords with operations and messages exchanged by services.

The Semantic Web Services (SWS) technology has been advocated as a solution for enabling system integration without human intervention. Semantic languages such as OWL-S (Web Ontology Language for Web Services) [2], WSML (Web Service Modeling Language) [3] and SAWSDL (Semantic Annotations for WSDL) [4] provide resources for describing the semantic meaning of messages and operations provided by Web Services. Furthermore, such information may be explored to automatically discover, compose and execute services.

This paper describes a platform for discovery and execution of Semantic Web Service compositions. This platform comprises a mechanism for dynamically identifying compositions based on semantic descriptions of services, and an engine for execution of compositions, which are invoked transparently as a single service.

The remainder of this paper is organized as follows. Section II discusses the use of semantic languages for Web Services description, the processes of web service discovery and composition, and the related work in this field of research. Section III presents the platform for automatic semantic web services discovery, composition and execution. Section IV presents the prototype of the platform and analyzes its performance. Section V concludes the paper, summarizing the contribution brought by this work, comparing the proposed platform with other research works and describing potential improvements to this work.

II. STATE-OF-THE-ART

This section presents the available technology to describe Semantic Web Services and to discover and execute compositions, as well as recent research efforts in this area.

A. Semantic Description

Semantic description languages, such as OWL-S, WSML and SAWSDL provide means for describing Web Services based on ontologies. Both OWL-S and WSML create semantic descriptions dissociated from the syntactic description given by WSDL. On the other hand, SAWSDL, which is the most recent W3C recommendation for SWS description, allows annotations to be added to WSDL elements in order to associate them with concepts defined in a domain ontology.

SAWSDL is language-neutral, since it allows any semantic-capable language to be employed for defining domain ontologies. Besides, it is more flexible and easier to use than the other languages, because it requires only the addition of semantic annotations to WSDL descriptions [5].

SAWSDL also lets behavioral constraints to be specified in ontologies and referenced by inputs and outputs through annotations, in order to associate pre- and postconditions with operations provided by a web service.

B. Discovery of Semantic Web Services

Semantic descriptions allow discovery mechanisms to interpret the meaning of functionalities exposed by web services. This process requires inferences to be performed based on ontologies that describe elements contained in
service descriptions. These mechanisms are able to compare the functionalities exposed by the existing semantic services with the requirements specified in a discovery procedure, selecting services that semantically match the specified requirements.

The matching process requires each operation under analysis, as well as its inputs and outputs, to be compared with the desired operation, including the available inputs and the required outputs, which were specified in the discovery procedure. This comparison is based on the concepts defined in ontologies associated with operations, input and output messages, in the aim of identifying semantic matches [6].

The similarity between two concepts may be evaluated using the criteria proposed in [7], which specifies an accuracy degree based on how similar concepts are. The similarity analysis takes into account any existing hierarchical relationship, as well as the existing properties and other characteristics defined in the domain ontology [6]. During this evaluation, the concepts under analysis are compared and classified into one of the similarity degrees defined in Table I.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>Concepts being compared are identical</td>
<td>0</td>
</tr>
<tr>
<td>Plugin</td>
<td>A supertype of the concept is available</td>
<td>1</td>
</tr>
<tr>
<td>Subsume</td>
<td>A subtype of the concept is available</td>
<td>2</td>
</tr>
<tr>
<td>Fail</td>
<td>Concepts are not hierarchically related</td>
<td>3</td>
</tr>
</tbody>
</table>

C. Semantic Web Services Composition

A composition can be defined as a group of services that, by working together, are able to cooperatively accomplish a task, and therefore can be seen as a new service. A composition is necessary when a single service that executes the whole task is not available [8].

The assembly of compositions requires understanding messages and operation semantics. Dealing with behavioral constraints, requirements and results provided by each service involved is also mandatory. [9]. The composition process may rely on user interaction, resulting in non-automated approaches, or on semantic technologies, which allow the process to be partially or fully automated. During this process, new services may be dynamically discovered and added to the composition. This process is repeated as many times as needed, until the result obtained by the composition matches the required outputs and behavior. It is also possible to give up searching based on a stop criterion (e.g., reaching the maximum number of services allowed in a composition) or when no more services able to take part in the composition are found.

D. Execution of Composed Services

The execution of service compositions, either semantically described or not, may be accomplished by adopting two different strategies: orchestration or choreography.

The first strategy relies on an orchestrator, which is responsible for invoking each service in a composition and for handling inputs and outputs. The WS-BPEL (Web Services Business Process Execution Language) standard allows compositions to be described and executed by orchestration engines [10].

The second strategy, on the other hand, relies only on the services themselves to collaboratively execute the composed service. The WS-CDL (Web Services Choreography Description Language) standard defines how choreographies can be described and executed [11].

E. Related Work

In the past few years, the research on semantic web technologies has grown constantly. Different approaches for semantic web service discovery and execution of compositions have been proposed. A representative share of these research works is described in this section.

Puntheeranurak and Tsuji [12] propose a discovery mechanism that executes a matching algorithm which relies on semantic information obtained through SAWSDL. The idea behind the proposed mechanism is to compare the functional requirements of the requested service with the available services and identify levels of similarity among them. The semantics associated with the description of the services through SAWSDL is employed to make inferences. Inputs and outputs are compared based on subsumption reasoning. The algorithm proposed in [12] does not discover compositions, but just individual services.

Prazeres et al [6] propose a solution based on OWL-S and on an algorithm that implements a minimum cost policy for discovery and composition of semantic web services. Services published in a UDDI registry are associated with a cost, which is calculated based on the amount of inputs and outputs of the corresponding service. A graph is created in order to allow compositions to be identified. The algorithm does not take into account the inputs available at request time. When a request is made, the graph is updated to take into account the available inputs, and the cost is adjusted. Discounts are given on the cost of a service if it has inputs that are currently available. Finally, compositions that have the minimum cost are selected. However, even with the discounts, the selected compositions may require inputs that the client does not have. Besides, the algorithm uses the available inputs to select only the first service of the composition, and does not combine them with outputs produced by other services to be used as inputs to new ones. This is done because the graph is not created at request time, aiming to reduce the search time needed to find a composition. Services with the same inputs and outputs but different semantics are not distinguished by the algorithm.

Mehandjiev et al. [13] present a semi-automatic approach based on templates for assisted web service composition. Through a tool that abstracts the technical details and the data flow between services, users select a template that conforms to their needs and start building the composition. The tool provides options for selecting services and highlights those that are compatible through the analysis of the semantics associated with the inputs, outputs, pre- and post-conditions of each service. The discovery process requires user assistance and occurs at design time.
Finally, the work of Belouadha et al. [14] proposes an approach for describing and composing semantic web services based on UML (Unified Modeling Language). The semantic description of services is given by a UML language-independent semantic metamodel. Later, this metamodel is transformed in a WSDL annotated with SAWSDL using transformation rules. At last, the authors use BPMN (Business Process Modeling Notation) for modeling web services flows and for defining the execution plan of the service composition. The authors create compositions statically, and the process is not automatic with respect to the creation of compositions.

III. COMPOSITION AND EXECUTION PLATFORM

This work targets the design and development of a platform that allows automatic composition and execution of semantic web services. In this platform, SAWSDL annotations are extracted from WSDL files and employed to build composition graphs, in which compositions are discovered based on service semantics. The composition can be invoked by clients through a single request, ignoring the fact that the request will be fulfilled by multiple services.

A. Architecture

The platform was designed to be both programming language and platform-independent. Its architecture has 9 components, which are shown in Fig. 1.

The Web Service component exposes 3 operations that are invoked by clients to request a specific service, to publish a new web service or to execute a service composition. As a result, the first operation returns the discovered composition, the second stores the service into the repository and the third returns the response produced by executing the composition.

The Semantic Annotations Extractor Module is responsible for extracting semantic annotations from WSDL descriptions during the publishing process and for storing them in a relational database model, represented by the SAWSDL Repository. This strategy is adopted in order to optimize the discovery process, since it is faster than parsing WSDL descriptions every time a request is made. This module is also responsible for retrieving annotations from the SAWSDL Repository during the discovery process.

The Discovery and Composition Module is responsible for actually discovering the web services compatible with the request requirements. Based on the annotations extracted previously and on the parameters specified in the request, the semantic matching is performed to verify its similarity with the requested service. This module builds a composition graph based on the relationships among concepts associated with inputs, outputs and operations. At the end of this process, the discovered compositions are stored into the Compositions Repository and returned to the client through the Web Service.

The result of a request may contain none, one or multiple paths leading to the desired outputs. Multiple paths will be returned when each one meets part of the request. If two paths that lead to the same output are found, just one will be selected and returned based on a proposed criterion, which will be described in section III.E.

As soon as they are discovered, compositions are processed by a WS-BPEL Generator. This component retrieves compositions stored in the Composition Repository, generates WS-BPEL files that describe these compositions and stores them in a WS-BPEL Repository.

Finally, the client can invoke a composition through a single request directed to the Web Service. The request will be dispatched by a Request Handler Module, which manages the ongoing requests. This module activates the Orchestrator Module, which executes the corresponding WS-BPEL file. The outputs produced by the composition are then returned to the client.

B. Discovery Request

To locate a service composition, the client must invoke an operation exposed by the Web Service component. This operation requires six parameters: availableInputs, desiredOutputs, desiredOperations, maximumDepth, timeout and allowRebuild. The availableInputs, desiredOutputs and desiredOperations are lists of URIs (Uniform Resource Identifiers) that refer to concepts defined in domain ontologies. The maximumDepth determines how many services that, sequentially, may be present in each path of the compositions graph. A composition with a sequential execution flow of three services has three depth levels. The timeout parameter corresponds to the time interval the client is willing to wait for the result of the request. Finally, the parameter allowRebuild specifies whether previously found compositions can be rebuilt or not (this will be further explained in section III.F).

C. Discovery Algorithm

Fig. 4 shows the pseudo-code of the discovery algorithm, which takes as input three lists of URIs: availableInputs, desiredOutputs and desiredOperations. The services stored into the repository are recovered and used with the available inputs to match the startup services (lines 4-5). Next, the target services are selected based on the desired outputs and operations (lines 6-7).
The semantic matching of startup services selects services from the SAWSDL repository that require inputs which semantically match the available inputs specified in the discovery request. For being selected, all inputs of the web service must be classified as exact or plugin. The condition for service selection is given by:

$$\forall x \in \text{In} : M_x \leq 1$$

where:
- In: set of service inputs;
- $M_x$: value resulting from the match of input $x$ of the service, according to Table 1.

The semantic matching of target services, on the other hand, takes into account not only if the service produces desired outputs, but also if it has at least one operation that semantically matches one of the desired operations. First, the outputs of the selected operations are compared with the desired outputs. If in the list of requested outputs there is a match with value lower or equal to 1, the web service is then selected. Then, during the semantic matching of operations, the algorithm selects services that have at least one operation similar to one of the desired operations. Thus, the matching is performed for each operation provided by services with the desired operations. If the list of desired operations has one with a similarity degree lower than or equal to 1, the web service is then selected.

At the end of this step, only services that have operations classified as exact or plugin, which also have outputs classified as exact or plugin, will be selected as target services. Therefore, the output is strictly related to the operation semantics, since the web service must be selected in the operation matching and in the output matching.

After performing these steps, the algorithm checks if any service was selected in both inputs and outputs matching. If this occurs, a service that provides at least one of the desired outputs based on the available inputs was found. In the sequence, the algorithm tries to find paths linking startup services to target services, building a composition graph.

### E. Composition Graph

A composition is characterized by paths in the graph linking a startup service to a target service, producing at least one desired output. Thus, we know from where to start and where to stop. Each node of the graph is represented by a web service and the edges are the semantic links between them.

Before starting to build the composition graph, the outputs of the startup services, now used as inputs, are added to the list of available inputs, initially composed by the inputs sent in the request.

The algorithm uses layers of services, as shown in Fig. 2, to represent the composition graph. The first layer is composed by the startup services. The other layers emerge as the algorithm iterates over the other services. Each iteration compares semantically the inputs required by the available services with the available inputs. The selected services form a new layer and their outputs are added to the list of available inputs for use in subsequent iterations. Thus, new paths may be created in the graph at each iteration.
Services of a layer are linked to services of the top layer through the origin of their inputs. As illustrated in Fig. 3, the inputs of service Z may come from service X or from inputs specified in the request. Eventually, the inputs of a service may come from one or more services, such as occurs with service Z, depicted in Fig. 3.

As the algorithm progresses, there are several paths in the graph that can result in different desired outputs. Each path may or may not find an operation that meets the request. Once services are selected to compose a layer, the algorithm checks if any of them offer a desired operation and output. If so, it means that a path to an output was found. Nevertheless, the path of the graph continues being built looking for new outputs. A path is interrupted only when it is not possible to establish a semantic link with other services or when a service appears repeatedly in the same path, featuring a loop, or when the path reaches the depth specified in the request.

For the algorithm, each path in the graph leads to a single output, even if multiple requested outputs are in the same path in the graph. Fig. 3 shows such a situation, where services Z and T provide requested outputs and, despite being on the same path in the graph, the algorithm treats them as two different paths – one composed by services Y → Z and the other by services Y → Z → T.

At the end of the graph construction, either by a timeout or because all the requested outputs were found, the algorithm examines whether two or more paths lead to the same output. In this case, the following function is applied to each path in order to select the one with the lowest semantic mismatch degree:

\[
SMD(P) = \sum_{i=1}^{N_s} \text{Min}_s * \alpha + \text{Mout}_s * \beta + \sum_{i=1}^{N_s} \text{Mop}_s * \omega
\]

where:
- \(SMD(P)\): Semantic Mismatch Degree of Path C;
- \(N_s\): Number of services in the path.
- \(\text{Min}_s\): Value resulting from the semantic matching of inputs of each operation in the path, according to Table I;
- \(\text{Mout}_s\): Value resulting from the semantic matching of the output of the path, according to Table I;
- \(\text{Mop}_s\): Value resulting from the semantic matching of each operation in the path, according to Table I.

Once calculated the cost of each path, the algorithm selects the path with the lowest SMD leading to each output. This means that the path that provides the best similarity degree of their inputs, output and operations, along with the lower depth, will be selected. This calculation is performed only when there are two or more paths that provide similar outputs. Otherwise, calculating these metrics is unnecessary and the single path obtained is selected directly.

At the end, the SMD(C) function is applied to the final composition, resulting from the union of all paths, in order to calculate the global SMD according to the following function:

\[
SMD(C) = \sum_{i=1}^{N} SMD(P_i)
\]

where:
- \(SMD(C)\): Semantic Mismatch Degree of composition C;
- \(SMD(P_i)\): Semantic Mismatch Degree of path \(P_i\);
- \(N\): Number of paths in composition C.
F. Composition Repository

All compositions discovered as result of a request are stored in the Compositions Repository for future requests. Before making the whole discovery process when a request for a new service is received, the platform verifies the existence of any composition that meets the request needs. If it exists, this composition is returned to the requester just after a validation.

Validation of compositions consists in checking the availability of services involved. This process is necessary because web services can be removed from the repository. If all services remain available, the composition is returned; otherwise, a new discovery process begins.

There is also the case in which new web services become available and, in some way, improve the existing compositions. In such cases, when requesting a service that already has a composition available, the algorithm checks if any new service was published after the creation of the composition and reexecutes the discovery process, in order to identify any eventual improvement. However, this reconstruction can be prevented through the allowRebuild parameter sent in the request. If the value specified in this parameter is true, the reconstruction of the composition is performed; otherwise, the existing composition is returned.

IV. IMPLEMENTATION AND EVALUATION

The discovery algorithm was implemented in Java, using JDK 1.6.0. Semantic annotations are extracted using the EasySAWSDL API [15] and stored using the MySQL 5.5 database management system The Jena [16] and Pellet [17] APIs are used to load and infer meanings of concepts defined in the ontologies that are associated with WSDL elements. The JUNG API [18] is employed to build the composition graph based on the relationships among concepts.

To evaluate the performance, measure the response time and validate the compositions selected by the discovery algorithm, some experiments were executed and the obtained results are presented in this section. The environment in which the experiments were performed was composed of an Intel® Core 2 Duo 2.10 GHz CPU, with 3 GB of RAM, running the Windows 7 operating system.

For the experiment, 1,000 semantic web services were obtained from the www.semwebcentral.org repository, and these were replicated until reaching 5,000 web services. On average, services had one operation with two inputs and one output. Each input, output and operation had a single semantic annotation. Services were published and their annotations were extracted and stored in the SAWSDL Repository.

Different numbers of services were used during the tests in order to compare the time needed to find the composition as the amount of services increased. Several factors have direct influence on the performance of the composition algorithm, such as: the number of services involved; the number of operations of a service; the number of inputs and outputs of an operation; the number of semantic annotations in operations, inputs and outputs; the number of ontologies involved; the number of concepts and relations in an ontology; and the length of the shortest path from a startup service to a target service.

The SMD function was evaluated with different weights in order to select the best combination of values. After executing the composition process numerous times, the best compositions in terms of semantic quality were obtained according Table II.

| TABLE II. WEIGHTS ASSOCIATED WITH EVALUATED ANNOTATIONS. |
|----------------|----------------|----------------|
|                | Input          | Output         | Operation      |
| Weight         | 0.3            | 0.35           | 0.35           |

Defined the weights, the web services were grouped into sets of 1000, 2000, 3000, 4000 and 5000 services. The request that had its response time evaluated was intended at finding a composition that returned 3 given outputs based on 2 available inputs.

The graph obtained with the experiment is shown in Fig. 5 and the time resulting of each evaluation is shown in Fig. 6. As the amount of services grows, the response time grows linearly, i.e., the response time is directly proportional to the amount of available services. Furthermore, the results presented confirm that the proposed approach is effective to discovery and composition of web services, since the time associated with the execution of the algorithm is polynomial.
The discovery process comprises two different stages, and the response time taken by each stage is shown in Fig. 6. The first stage, labeled Service Discovery, identifies startup and target services. Startup service selection requires the analysis of each input of each operation of a service in order to select it as a startup service. On the other hand, selecting the target services requires just one compatible output for service selection, resulting in a faster procedure. The second stage, named Composition in Fig. 6, is responsible for consuming the largest share of the time required to execute the algorithm. This result was largely anticipated, due to the successive matching operations to identify relationships between services needed for building the composition graph. In addition, this stage requires the calculation of the semantic mismatch degree of each path identified in the graph for selecting the composition that best fulfills the request.

The algorithm proves to be able to identify compositions and to obtain all the desired outputs through services available in the repository based on a list of inputs. Since the quality of the obtained results depends solely and exclusively on the matching between the parameters sent in the request and the annotations associated with services, the more detailed were the parameters and the services annotations, the better is the quality of the obtained compositions.

V. CONCLUSIONS AND FUTURE WORK

Assembling web service compositions is not a trivial task, specially with the large amount of services available nowadays. Composition is needed when a single service is not able to execute the required task. Semantic technologies can play an important role in this process, allowing the automation of the composition process.

In this context, this paper presented a platform to automatically build compositions of Web Services at request time based on the SAWSDL annotations. This platform comprises also an execution engine, which allows clients to invoke a composed service through a single web service request. This is achieved by executing a WS-BPEL description of the composed service.

Compared with related projects found in the literature that have similar goals, which were described in section II.E, compositions are obtained by the proposed algorithm at request time without user assistance, and will never require inputs that the client does not have. The results of the experiments demonstrated that the algorithm is able to combine single services to build compositions with a response time directly proportional to the amount of services available in the repository.

As a future improvement, the discovery mechanism will be capable of identifying the same concepts represented in different ontologies via ontology mediators. This is desired because people and organizations often develop different ontologies to represent the same domain, due to their different points of view of the same scenario.

Improvements are also being made to the developed implementation in order to allow the use of control structures and parallelism during the execution of the discovered compositions.

REFERENCES