The Use of Semantic-based Suggestions for Web Service Composition

Diogo Phelipe Busanello da Silva and Frank Siqueira
Department of Informatics and Statistics, Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil

Abstract - The creation of complex SOA applications requires multiple services to be combined. Building service compositions, though, requires analyzing if inputs and outputs of composed services are compatible, and this is a costly task to be done manually. The semantic description of Web Services using ontology concepts to describe the provided operations, inputs and outputs, is able to facilitate the service composition process. This paper describes a mechanism that helps the user to build a service composition, providing service recommendations based on the semantic description of Web services. The semantic service suggestion mechanism is integrated with a composition tool, called CVFlow, in order to provide a complete support for composition developers.

Keywords: Semantic Web Services; Service Composition; SAWSDL.

1 Introduction

Web Services is a well-known technology for building and reusing services in a SOA-based environment. Although this technology provides means to describe the abstract functionality of services and to specify how they can be invoked, this description has merely syntactic meaning for service consumers running on remote machines. Therefore, in order to consume the provided service, it is still necessary the intervention of a developer, who assimilates the meaning of service operations and the corresponding inputs and outputs and builds the code to invoke the service properly.

Semantic Web Services are a merge of the Web Services technology with the concept of Semantic Web defined by Tim Berners-Lee [1]. With the association of these technologies, Web Services become meaningful for other machines and the automation of Web Service compositions is made possible.

This paper proposes a semi-automated approach to compose Web Services based on a semantic service suggestion mechanism, called S'M, which is integrated with an existing workflow tool. SAWSDL (Semantic Annotation for WSDL and XML Schema) was chosen as underlying semantic technology due to its recent adoption as a W3C standard [2]. Recommendations are provided based on the semantic match between inputs required by services stored in a repository and a list of outputs produced by services in a partially built composition.

The remainder of this paper is organized as follows. Section 2 presents the existing technology to describe and compose Semantic Web Services. Section 3 introduces the semantic service suggestion mechanism proposed in this paper and presents its architecture and selection algorithms. Section 4 shows how the proposed service suggestion mechanism is integrated with a composition tool, called CVFlow. Section 5 discusses related proposals found in the literature and compares them with the service suggestion mechanism described in this paper. Finally, Section 6 presents the concluding remarks and proposes some future work in this research field.

2 Technological background

2.1 SAWSDL

The lack of semantic support on the Web Service Description Language (WSDL) is an obstacle for automating service discovery and composition. Ambiguities may arise while a composition is being built, given that type compatibility between input and response messages is neither sufficient nor necessary to guarantee that they are semantically equivalent. A response message produced by a service may have the same type of an input message required by another one, and despite this, data semantics may be completely different. On the other hand, response and input messages may have different identifiers and/or types but still be semantically compatible.

SAWSDL [2] is one of the existing solutions for the lack of semantic information on service descriptions. It is based on a previous proposal entitled WSDL-S [3], and was recently published as a W3C standard.

SAWSDL by itself does not provide any specific semantics, but allows the service developer to add annotations to elements described in purely syntactic WSDL descriptions that associate them with semantic concepts defined by ontologies [4]. This is the main difference between SAWSDL and other approaches, such as OWL-S (Web Ontology Language for Web Services) [5] and WSMO (Web Service
Modeling Ontology) [6], i.e., it does not create another language; instead, it is just an extension of the WSDL specification [7].

The attributes defined by SAWSDL to enable semantic annotation on WSDL elements are summarized by Table 1 and described in the following paragraphs [4].

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelReference</td>
<td>Reference to concepts in a semantic model</td>
</tr>
<tr>
<td>liftingSchemaMapping</td>
<td>List of pointers to data-lifting transformations</td>
</tr>
<tr>
<td>loweringSchemaMapping</td>
<td>List of pointers to data-lowering transformations</td>
</tr>
<tr>
<td>attrExtensions</td>
<td>Attaches attribute extensions where only element extension is allowed</td>
</tr>
</tbody>
</table>

The modelReference attribute allows multiple annotations to be associated with a given WSDL component through URIs (Uniform Resource Identifiers) that identify concepts in a semantic model [2].

The liftingSchemaMapping and loweringSchemaMapping annotations address post-service discovery issues. The former converts XML data obtained from a service into a semantic model, while the latter maps a semantic model into XML data [4].

The former attributes are valid for all versions of WSDL. The attrExtensions attribute is used in WSDL 1.1 files, in which the extensibility attribute is not allowed, but element extensibility is.

2.2 Web service composition

Web Service composition is a process that combines and binds services, either atomic or composed ones, in order to create a new service. Different techniques to compose Web Services can be found in the literature. This process is usually classified into two categories: dynamicity and automation [8].

Compositions may be classified as either static or dynamic. Static composition takes place during the design phase, while the components are chosen, linked together, compiled and deployed. This technique works fine as long the components do not change. On the other hand, a dynamic composition occurs at request time and is more adaptable to changes in the employed web services [9].

The second classification category is based on the degree of automation with which compositions are built. A manual composition relies on the developer to combine services. An automated composition is the process that selects, combines, integrates, and executes services automatically [10]. Even an automatic approach relies on the user to specify parameters such as the existing inputs, desired operations and required outputs. There is also a semi-automatic approach that provides an intermediate degree of automation, with which the composition is built using a GUI and, as the user creates the composition, the tool suggests services that can be composed based on their characteristics.

The use of semantic annotations in Web Service descriptions allows the automation of the composition process. There are two aspects that directly affect the automatic service composition: the service description technology and the matching between services.

The most adopted standard for Web Service description (i.e., WSDL) is only capable of describing service interfaces syntactically. In order to avoid this limitation, the research community created ways to add support for semantic technology to describe services (e.g., OWL-S, WSMO, SAWSDL). This is done by associating vocabulary terms defined in a domain ontology with operations provided by a service and the corresponding input and output messages. The more detailed the domain ontology is, the more accurate the inference process of operations, inputs, and outputs becomes.

Semantically described services may be subject to semantic matching, in order to verify the compatibility of two operations by comparing the outputs produced by a given operation with inputs required by the other operation.

Fig. 1 shows an example of match combining three operations. In the portrayed example, output $m_{\text{out}}$ matches with input $o_{\text{in1}}$, and output $n_{\text{out}}$ matches with input $o_{\text{in2}}$, allowing operations $M$ and $N$ to be composed with operation $O$.

Paolucci et al. [13] defined a set of criteria to measure the similarity between the output of one operation ($Op1$) and the input of another one ($Op2$). A match is considered exact when the input required by $Op2$ and the output produced by $Op1$ are associated with the same ontological concept. When the output of $Op1$ is an ontological subclass of the input of $Op2$ it is considered a plugin match. The subsumption match in the opposite case, when the output of $Op1$ is an ontological superclass of the input of $Op2$. When there is no correlation between the (input, output) pair, the similarity is considered a fail. Table 2 associates these similarity levels with values that are employed in this work to calculate the semantic mismatch between operations that are candidates to be composed.

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>0</td>
</tr>
<tr>
<td>Plugin</td>
<td>1</td>
</tr>
<tr>
<td>Subsume</td>
<td>2</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
</tr>
</tbody>
</table>
The semantic service suggestion mechanism

The service composition approach proposed in this paper is based on the use of semantic technology aiming to create a service suggestion mechanism, which selects services from a repository with inputs that semantically match outputs produced by services that are part of a composition that is being designed.

3.1 Architecture

The architecture of the Semantic Service Suggestion Mechanism, from now on called S3M, is illustrated by Fig. 2. The figure shows two user profiles: the provider, which is responsible for registering services in the repository; and the composer, which receives suggestions from the S3M mechanism and creates the service composition. Besides, eight software components are shown in the figure. These components will be described in the following paragraphs.

The service registration interface is provided by the Semantic Web Service Registry, which takes semantically annotated WSDL files and extracts their operations, inputs, outputs and the corresponding semantic annotations, and stores them in the SAWSDL Repository. This is done to speed up the service discovery and suggestion procedure, since querying the database is much faster than parsing WSDL files every time a new suggestion is requested. This approach is also adopted in [12]. Fig. 3 shows the entity-relationship model of the database built within the SAWSDL Repository. This component also executes a procedure that checks if registered services are running and updates the availability metric stored in the Service table.

The Service Composer provides the user interface for the composition designer. This component is responsible for arranging the composition in a linked graph of connected operations. In the beginning of the design process, the composition has only a set of input parameters provided by the user. These inputs are consumed by operations that are added to the composition, which provide outputs that can be consumed by other operations, and so on.

The Output Analyzer checks the list of outputs available in the composition that is being built and passes them to the Service Matcher module, which identifies operations with inputs that are semantically compatible with them.

The Service Suggester takes all operations with inputs that are semantically compatible with the outputs produced so far by the composition and passes them to the Service Matcher module, which orders operations executing the algorithm that will be described in the remainder of this chapter. The ordered list is provided to the Service Composer and then presented to the user, which can select a new service from the list and add it to the composition.
Finally, the Composition Registry is responsible for persisting designed compositions in the Composition Repository. This last component provides the information needed for executing the composition. Execution is done by a third-party tool that is out of the scope of this paper.

3.2 Availability function

A process executed in the background by the SAWSDL Repository updates the metric that is stored in the availability column of the Service table. This metric takes into account the availability history of each service stored in the repository. In the first two attempts to monitor a service, available services receive value 1, while services that are offline receive value 0. A weighted mean is calculated in subsequent monitoring intervals, giving increasing relevance to recent attempts than to past ones. The following description shows how the weighted mean is calculated by the availability function.

\[
m_n = \begin{cases} 
1 & \text{if available,} \\
0 & \text{otherwise}
\end{cases} \\
\frac{m_1}{1} = 1, \frac{m_2}{2} = 1, \ldots, \frac{m_n}{n} = 1 \\
\frac{m_n}{n} = \omega \cdot m_n + (1 - \omega) \cdot \frac{m_{n-1}}{n-1}
\]

where:
- \( m_n \): result of the monitoring procedure on time \( n \);
- \( \frac{m_n}{n} \): availability metric on time \( n \);
- \( \omega \): weight given to the last monitoring operation.

Equation (1) is periodically calculated to obtain the availability of each service registered in the SAWSDL Repository. The user is free to adjust the monitoring period and the weight factor.

3.3 Semantic match algorithm

The algorithm adopts the classification proposed in [13] to verify the semantic compatibility between an input, output pair. This is represented by Equation 2:

\[
SM(A,B) = \{ x \mid x \in (\text{exact}, \text{plugin}, \text{subsume}, \text{fail}) \}
\]

where \( SM(A,B) \) assumes the mismatch value taken from Table 2 for the pair \( A \) and \( B \).

Fig. 4 shows the pseudo code that identifies matches between a list of inputs and a list of outputs, associating a match value with each acceptable pair. The match value assumes the minimum value between the match of the input and each output, representing the value of the closest match.

The command in line 07 verifies if the comparison between semantic concepts associated with a given input, output pair results in an exact or plugin match. The algorithm keeps looking for better matches for inputs that do not have an exact match. After all input, output pairs are semantically analyzed, a list with all matching inputs found is returned by the algorithm (line 13).

3.4 Suggestion algorithm

The semantic service suggestion algorithm is the core function executed by the Service Matcher. Its pseudo code is presented in Fig. 5.

The algorithm receives as parameters an operation that is the focus of the matching procedure, a list of outputs available in the composition and a list of candidate operations provided by the Service Suggester to be evaluated as possible matches. It then begins by stepping through each operation (line 03), and then verifies the availability of the service that provides
the operation (line 04). If the availability is lower than 0.5, it is discarded for being considered unreliable.

If the service meets the availability requirement, the algorithm verifies the match between inputs required by the candidate operation and the outputs produced by the focus operation (line 05). The candidate operation will continue being considered only if at least one acceptable match is identified. In this case, it identifies the outputs available in the composition that match the inputs required by the candidate operation (line 07).

Finally, the algorithm calculates the composition suitability of this candidate operation (line 08). The lower the value, the more compatible the operation is. Equation 3 shows how the composition suitability of a candidate operation \( O \) is calculated:

\[
CS_O = SM_O \times \alpha + SR_O \times \beta + (1 - av_O) \times \gamma \quad (3)
\]

where:
- \( CS_O \): Composition Suitability of operation \( O \);
- \( SM_O \): Semantic Mismatch between the available inputs and operation \( O \);
- \( SR_O \): Semantic Mismatch between the non-matching inputs of operation \( O \) and the available inputs;
- \( av_O \): availability of service that provides operation \( O \);
- \( \alpha \): weight associated with the matching inputs;
- \( \beta \): weight associated with non-matching inputs;
- \( \gamma \): weight associated with service availability.

After all candidate operations on the list are verified, the algorithm returns a list with suggested operations, ordered by their composition suitability (line 13).

4 The CVFlow tool

A composition tool has been integrated with the semantic service suggestion mechanism aiming to evaluate its appropriateness. This tool, called CVFlow, provides resources for defining workflows through the construction of a graph, in which a service corresponds to a node and the edges represent outputs produced by a service being used as inputs by another one. The creation of service compositions with this tool can be classified as dynamic and semi-automatic composition.

CVFlow is a workflow tool that was originally developed by the Lapix research group at the Federal University of Santa Catarina (UFSC). Although its first version focuses on image processing, it was designed to allow multiple uses. The Lapix group is currently developing extensions for hardware and logical control, for building 3D images and for video processing [11].

The nodes of a flow modeled with CVFlow consist of two types: Algorithms and Resources. Algorithms are programming functions defined by the developer that can be either local or remote. Resources are data items that serve as input parameter for Algorithms, e.g., images, numbers, strings, data structures and scripts.

Fig. 6 shows the graphical user interface of CVFlow. The central area of the interface is the composition area. The list of available Algorithms and Resources is displayed on the left side. The user can drag and drop any node to build the composition. The composition illustrated by Fig. 6 has three Resources and four Algorithms.

```
01 function semanticSuggestion(focusOperation, outputList, operationList)
02     def suggestList = Ø
03     for each operation in operationList do
04         if getAvailability(operation.service) >= 0.5 then
05             smFocus = semanticMatch(focusOperation.outputs, operation.inputs)
06             if smFocus.matchList ≠ Ø then
07                 smOutputs = semanticMatch(outputList, operation.inputs - smFocus.matchList)
08                 suitability = calcCSo(operation, smFocus.matchList ∪ smOutputs.matchList)
09                 suggestList.add(operation, suitability)
10             end if
11         end if
12     end for
13     return suggestList.sortBy(suitability)
14 end
```

Fig. 5. Pseudo code of the Semantic Service Suggestion Algorithm

Fig. 6. User Interface of CVFlow
CVFlow has an internal architecture composed by three layers: the visualization, the workflow, and the core layer. The visualization layer is the GUI, shown by Fig. 6. The workflow layer represents every aspect of the composition, such as the inputs, outputs, nodes (i.e., Resources and Algorithms) and the connections between nodes. The Core layer is responsible for executing the Algorithms that are part of the composition.

In the original version of CVFlow, it is up to the user to verify if outputs from one service match with inputs from other ones. Execution errors occur when incompatible nodes are connected by the user.

The S'M mechanism has been integrated with CVFlow in the aim of helping the user to easily select semantically compatible services (i.e., Algorithms in CVFlow terminology) and build a composed service. When the user selects a service in the composition, the mechanism orders the list of available services based on their suitability, and shows the ordered list on the right side of the CVFlow interface, as shown by Fig. 7. The user can then identify a suitable service and drag it to the composition area, connecting it to the selected service.

Based on the services suggested by the S'M mechanism, users can more easily compose services. Besides, they avoid connecting incompatible services, which result in execution errors, and avoid using services with low availability.

5 Related work

The literature has several examples of software tools aimed at helping designers to build service compositions, including some that feature service suggestion mechanisms. These suggestions, though, are mainly based on syntactic information contained in service descriptions, which is often misleading. Just a few proposals found in the literature adopt semantic technology to identify compatible services.

Hobold and Siqueira [12] approach the composition of service using SAWSDL to discover and compose services automatically. To optimize the discovery process, during the publication of a service they extract from the WSDL the semantic annotations associated with operations, inputs and outputs, and store them in a relational database. This approach is also adopted in this paper. The main difference between these approaches is the way service compositions are created.

Another related research work was developed by Prazeres et al. [15]. These authors proposed an approach for automatic composition of semantic web services exploiting a graph-based model that takes into account the semantic similarity of services computed using subsumption reasoning on their inputs and outputs. This is an automatic approach for service composition that uses OWL-S for describing semantic web services. The way similarity between inputs and outputs is computed is similar to the strategy adopted by the S'M mechanism.

Automatic approaches such as the ones proposed in [12] and [15] lead to combinatorial explosion, resulting in poor performance. Besides, the designer is excluded from the composition process, becoming unable to contribute with the knowledge he often has on the problem domain.

Pi et al. [14] propose a tool for semi-automatic composition of semantic web services based on a GUI called Flow Editor. This tool was developed using Flex and is capable of converting WSDL files into OWL-S files. The tool is capable of exporting and executing the compositions. The main difference between the Flow Editor and our work is that the first uses OWL-S for expressing Semantic Web Services, while S'M is based on SAWSDL annotations. Besides being a W3C standard, SAWSDL leverages from WSDL instead of requiring a parallel semantic description such as OWL-S does. Another important difference between the S'M mechanism and the Flow Editor is that the first takes into account the availability history of services in its suggestion algorithm.

6 Conclusions and future work

This paper described a mechanism that provides suggestions for designers during the service composition process. The S'M mechanism takes into account the semantic annotations contained in the WSDL files and verifies the semantic match between operations. The mechanism also evaluates service availability, in order to avoid the use of low quality services that may compromise the execution of a composition.

The proposed mechanism has been integrated with an existing workflow tool, called CVFlow, which allows the composition to be graphically built using the provided suggestions. Preliminary productivity tests have shown very promising results.

As future work, we plan to use the composition repository to include behavioral patterns of users in the suggestion mechanism. This can be done by modifying the suggestion algorithm to take into account the frequency in
which users select a given operation. Integration with a
service orchestration mechanism described in [12] is also
envisioned as future work. Furthermore, usability tests are
being planned, aimed at evaluating how helpful service
suggestion is for composition designers and how much time
of the composition process is saved using the suggestion
mechanism.

7 References


and XML Schema”, 2007. Available at http://www.w3.org/TR/sawxsd

2006. Available at http://www.w3.org/Submission/WSDL-S/


Services”, 2003. Available at http://www.daml.org/services/owl-s/1.0/


Prezerakos, G.N.; Kaklamani, D.I.; Venieris, I.S.,
"Service Composition: State of the art and future challenges", Mobile and Wireless Communications

Compositions”. International Journal on Web and Grid

Paolucci, K. Sycara, D. L. McGuinness, E. Sirin and N.
Srinivasan. “Bringing Semantics to Web Services with

http://www.lapix.ufsc.br/cvflow

and Execution of Semantic Web Services
Compositions", Proceedings of the International
Conference on Semantic Web and Web Services
(SWWS 2012), July 2012.

“Semantic matching of web services capabilities” in:
Proceedings of 1st International Semantic Web
Conference (ISWC), 2002.

[14] Bingfeng Pi; Gang Zou; Chaoliang Zhong; Jun Zhang;
Hao Yu; Matsuo, A., "Flow Editor: Semantic Web
Service Composition Tool," 2012 IEEE Ninth
International Conference on Services Computing (SCC),
24-29 June 2012.

Semantic Web Services discovery and composition:
paths along workflows. In ECOWS'09: Proceedings of the
7th IEEE European Conference on Web Services, pp.