The Journal of Systems and Software 104 (2015) 1–16

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A process to identify relevant substitutes for healing failed WS-* orchestrations

Okba Tibermacine [a,*], Chouki Tibermacine [b], Foudil Cherif [a]

[a] Biskra University, LESIA, P.B. 145 R.P, Biskra 07000, Algeria
[b] LIRMM, CNRS and Montpellier University, France

ARTICLE INFO

Article history:
Received 21 March 2014
Revised 4 December 2014
Accepted 10 February 2015
Available online 16 February 2015

Keywords:
Web services orchestration
Web service selection
Formal concept analysis
Similarity measurement

ABSTRACT

Orchestrating web services aims to compose multiple services into workflows that answer complex user requirements. Web services are software components which are exposed to errors and failures that can occur during web service orchestration execution. Thus, many error-handling and healing approaches have been proposed to guarantee reliable orchestrations. Some of these approaches rely on the identification of relevant service substitutes to heal (by substitution) the defected services. In this paper, we propose an identification process of web service substitutes for healing failed web service orchestrations based on the measurement of similarity between service interfaces. The process reveals both simple and complex (compositions of) substitutes. We validated the approach via a set of experiments conducted on a collection of real web services.

1. Introduction

One of the most challenging topics in web service technology that was addressed by many researchers in industry and academia is web service composition design and maintenance (Fan and Kambhampati, 2005). Web service compositions are software components that satisfy client needs, and which no unique web service can satisfy. They enable developers to rapidly create new applications by reusing existing services.

Web services are software components that are exposed to errors and failures that may have different origins: a failure in the network, the unavailability of the application server or the database server, an error or an exception in the program implementing the service, etc. Some of these faults cannot be detected immediately, and could have many consequences on the compositions using them (Li et al., 2007). In the literature, many works tackle the recovery actions for healing web services using different techniques of diagnosis, monitoring, exception handling, substitution and adaptation (Alhosban et al., 2013; Ardagna et al., 2006; Ding and Xiang, 2013; Psaier and Dustdar, 2011; Simmonds et al., 2010a).

Identifying and selecting substitutes is a crucial and challenging task, especially with the absence of behavioral descriptions about web services. Thus, many approaches have been proposed for service selection. The study of the similarity between web services is one of the basic techniques used by these approaches. It has been extensively studied in many works (Ait-Bachir, 2008; Crasso et al., 2008; Kokash, 2006; Tibermacine et al., 2013). The goal behind studying similarity is to: (1) select and retrieve from directories services that match some criteria, (2) identify substitutes (replacements) between web services or between their operations, (3) determine service composability, i.e. whether in two operations, belonging to two services, the output of the first operation is similar to the input of the second operation. Hence, the first service can be composed with the second service. Thus, an efficient similarity assessment could be considered as a key solution for seeking relevant substitutes in order to heal (reconstruct) failed orchestrations. Since it is better to replace deficient services by similar ones (simple substitutes) or a composition of services (complex substitutes), instead of building a new web service orchestration from scratch.

In a previous work (Azmeh et al., 2011), we have proposed an approach that identifies simple substitutes; i.e. one operation that replaces one other operation (1-to-1 substitute). Nevertheless, some operations cannot be replaced by simple operations. This is frequently due to the complexity of their implemented functionality which sometimes makes them the only operations available in the Web. Their failure brings developers to look for other operations which can be composed together to obtain an equivalent functionality. These compositions are considered as *N*-to-1 substitutes; i.e. orchestrate *n* services/operations to replace one service/operation in case of failure. In this paper, we cover this shortcoming by proposing a process that reveals both simple (1-to-1) and complex (N-to-1) substitutes.

* Corresponding author. Tel.: +213 555850334.
E-mail addresses: o.tibermacine@univ-biskra.dz, otiber@hotmail.com (O. Tibermacine), tibermacin@lirmm.fr (C. Tibermacine), foud_cherif@yahoo.fr (F. Cherif).

http://dx.doi.org/10.1016/j.jss.2015.02.028
0164-1212/

The approach uses the WSDL (Web Service Description Language) interface description of the failed web service (partner link) to discover, from a service pool, a set of web service candidates that offer the same or related functionalities. The set is then refined using a filtering algorithm that exploits the similarity assessment technique proposed in Tibermacine et al. (2013), in order to keep only web services that have similarity relationships between them. Using the same similarity assessment technique, a similarity matrix between the input and the output of web service operations is built, and then it is exploited to create a concept lattice using Formal Concept Analysis (FCA, Ganter et al., 2005). FCA is a formal technique that enables to build classifications of objects. In our case, the objects are web services and their operations. A process of refinements is applied on the similarity matrix and its associated formal concepts to build the final concepts and the final lattices. A navigation and interpretation mechanism is used to identify, from the resulted lattices, all simple (1-to-1) and complex (*N*-to-1) substitutes for the failed operations in the defected partner link.

The main contributions of this paper are:

- Automatic browsing mechanism for the identification of simple and complex substitutes based on the similarity between web service interfaces.
- Selection, filtering and clustering algorithm based on service interface similarity for service candidate organization.
- A validation of the proposed approach through an illustrative case study and an experiment.

The paper is organized as follows: In Section 2, a background material is presented about the Formal Concept Analysis technique and its application on the web service context. In addition, an overview about the assessment of similarity between web services is also covered. In Section 3, we present the proposed approach illustrated by an example; we start by a general overview, then we present our illustrative example, and finally we explain in details the process of service substitutes identification and its components. The validation experiment conducted on real web services is described in Section 4. The paper concludes by a literature review section, a summary of the contributions and some perspectives.

2. Background

In this section, we present a background material about Formal Concept Analysis and its application in the context of web service classification. Then, we give an overview about the similarity assessment between web services, focusing more particularly on the assessment technique adopted for this work.

2.1. Formal concept analysis

Formal concept Analysis (FCA) (Ganter et al., 2005; Wille, 2009) is a technique of data mining and branch of mathematical lattice theory. FCA is used, among others, in data analysis, information retrieval, software engineering and data mining. FCA analyzes data which describes a relationship between groups of objects that share common attributes and provides an associated graphical representation. The main goal of FCA is to model concepts of thought as a unit of two parts:

- The concept Extension which comprises all objects that belong to the concept.
- The concept Intention which contains all attributes that these objects share.

The notion of formal context of objects and their attributes is essential for the application of FCA. The Formal context is an incidence table indicating which attributes belong to which object. Mathematically, a formal context (*G,M,I*) consists of:

Table 1
A formal context for a set of objects *G* and set of attributes *M*.

	Odd	Even	Prime	Composite	Square
1	x				x
2		x	x		
3	x		x		
4		x		x	x
5	x		x		
6		x		x	
7	x		x		
8		x		x	
9	x			x	x
10		x		x	

- *G*: a set of formal objects,
- *M*: a set of attributes, and
- *I*: an Incidence relation between the objects and the attributes. $I \subset G \times M$ is a binary relation where $(G,M) \in I$ is read: "object *g* has attribute *m*" such that $g \in G$ and $m \in M$.

A formal context is represented as a cross-table where the rows represent *G*, the columns represent *M* and the incidence *I* is represented by a series of crosses x as shown in the example presented in Table 1 (example taken from Ganter and Wille, 1999).

In this example (Table 1), we have a formal context *K* for a set of objects $G = \{1,2,3,4,5,6,7,8,9,10\}$ and a set of attributes $M = \{odd, even, prime, composite, square\}$. From the formal context *K*, FCA generates a set of all possible formal concepts where every formal concept is a maximal collection of objects (Extent) sharing a maximal set of attributes (Intent).

Mathematically, the set of common attributes is defined $\sigma(G) = \{m \in M |\ (g,m) \in I, \forall g \in G\}$. Analogously, the set of common objects is defined as $\rho(G) = \{g \in G | (g,m) \in I, \forall m \in M\}$. A formal concept is a pair of sets (G, M) such that $M = \rho(G)$ and $G = \sigma(G)$.

For example, in Table 1, $C1 = (\{3,5,7\}, \{odd, prime\})$ is a formal concept because only objects 3, 5 and 7 share the attributes odd and prime (and vice-versa). By contrast, $(\{4,9\}, \{square\})$ is not a formal concept because Extent $\{4,9\}$ is not maximal: object 1 shares the same attribute. The set of all concepts of a given formal context forms a partial order via super-concept and sub-concept ordering $\leq$. Either, $(G_1, M_1) \leq (G_2, M_2) \Leftrightarrow G_1 \subseteq G_2$. Or dually, $(G_1, M_1) \leq (G_2, M_2) \Leftrightarrow M_1 \subseteq M_2$. The set of all formal concepts of a given formal context and the partial order relation form a concept lattice (Galois lattice). Fig. 1 depicts the resulting concept lattice for the previous example, where (a) represents the resulting lattice, and (b) a focus on concept c1. The Concept lattice was built with Concept Explorer (ConExp) tool.[1]

The visualization of the lattice facilitates the detection of relationships between concepts. For example in Fig. 1(a), the concept $y = (\{4\}, \{even, composite, square\})$ is a sub-concept of *x*, where $x = (\{6, 8, 10\}, \{even, composite\})$. In this case, *y* inherits the attributes of *x*, and extends it by the attribute square.

In the context of web services, FCA has been successfully applied for web service selection, because it offers a formal classification and browsing mechanism thereby allowing the organization of web services in groups that share common characteristics (e.g. similarity values, keywords, QoS attributes, operation signatures, and/or functionalities). In addition, FCA allows to visually representing this classification by concept lattices that facilitate the navigation and the browsing for needed services and their potential substitutes.

[1] Available online: http://conexp.sourceforge.net/.

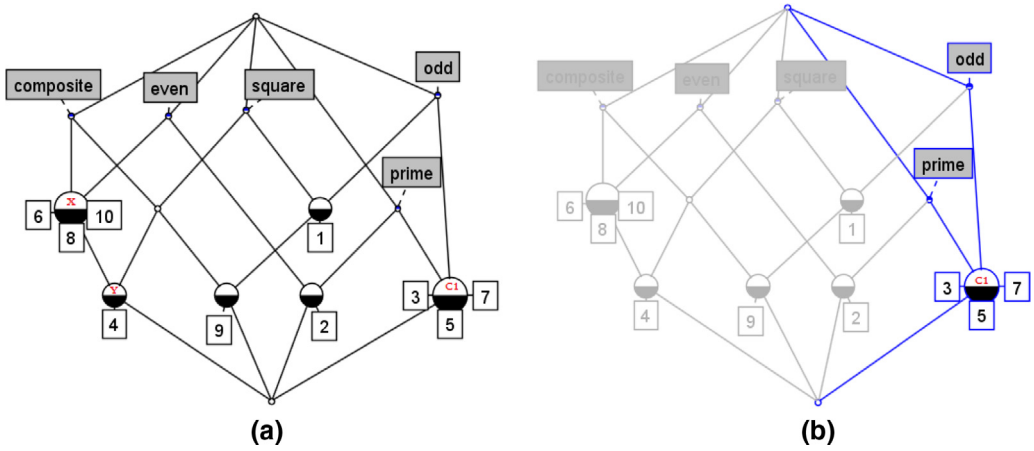

Fig. 1. Resulting formal concept lattice.

Technically, a concept in the web service context is modeled in a unit of two parts:

1. Concept Extension (G) which comprises all the considered web services. In specific cases, the concept extension comprises parts of the web services such as the operations, or even more the input and output messages.
2. Concept Intention (M) which holds the manipulated attributes that could be keywords, operations, messages, functionalities, QoS attribute, or only meaningful strings that represent any other web service (operation/message) characteristics.

In this approach, the role of FCA is the classification of service candidates in a set of concepts. Each concept holds services (operations) that are considered similar to each other. FCA reveals specialization relationships (composition and substitution) between the concepts, and organizes them into concept lattices for facilitating the retrieval of service substitutes.

During FCA application for the classification of web services, we build the so-called square concepts. They are defined as concepts with equal intention and extension sets (Azmeh, 2011); i.e. these concepts form square gatherings on the binary context matrix. They allow the identification of groups of mutually related objects (web services, operations, or messages). A better recognition of square concepts is achieved by performing mutual column/line interchange in the binary context matrix. Concrete examples about square concepts are provided during the presentation of the approach.

2.2. Web service similarity

Similarity measurement is essential for many pattern recognition problems such as clustering, classification and retrieval problems. The similarity is defined as the measure of how close to each other two instances are. The closer instances are to each other, the larger is the similarity value. The similarity is assessed in the form of real values that are comprised in the range [0, 1]. Thus, if the compared instances are sufficiently similar, the similarity value approaches 1, otherwise it approaches 0.

For the web service context, many similarity measures have been proposed to assess the similarity between web services or their parts, like operations and messages, according to their syntactic and semantic structure depicted in the web services WSDL files.

Mathematically, let O be a set of web services elements (Web Services, operations, messages). We define the similarity measure Sim: $O \times O \Longrightarrow$ [0,1] where:

- $\forall e_{i,j} \in O \rightarrow \mathrm{Sim}(e_{ij}, e_{ij}) = 1$ (service element with itself).
- $\forall e_{i,j}, e_{l,k} \in O, i \neq l \Rightarrow \mathrm{Sim}(e_{ij}, e_{ik}) \in [0, 1]$ (service elements in different services).

This generic function (Sim) has different concretizations. We use in the current work the similarity assessment function proposed in (Tibermacine et al., 2013). The function assesses the similarity between different parts after analyzing WSDL interface descriptions. These parts include services, operations, input and output messages, parameters, simple and complex types and documentation elements. The function incorporates, at the same time, different structural and semantic similarity metrics between identifiers. Stoilos (Stoilos et al., 2005), JaroWinkler (Winkler, 1990), Levenshtein (Levenshtein, 1966) are examples of the structural metrics, and Jiang, Lin, Pirro Seco and Resnik (Pirró, 2009) are WordNet based semantic metrics used in the similarity assessment process. Moreover, the similarity between message structures with complex type schema is evaluated using schema matching through a similarity-flooding algorithm, where complex types are modeled as labeled-oriented graphs.

Two operations are accepted as similar only if their similarity value (score) returned by the Sim function is greater or equal to a specific threshold; i.e. a similarity threshold ($\theta \in [0, 1]$) which is the lower limit of the similarity value of two objects (services, messages, operation, etc.) to be considered as similar. We consider five classes of similarities (very high, high, medium, low, and very low). A similarity value that ranges in [0.6, 0.8[is considered as a high similarity value, while a value that ranges in [0, 0.2[is a very low similarity value. Hence, if we fix the similarity thresholds to 0.85 for instance, we are accepting to be similar only compared objects with very high similarity values. A more relaxed threshold such as 0.55 allows to consider, as similar, all compared objects with medium to very high similarity scores.

We use in this work the similarity assessment technique that we described in Tibermacine et al. (2013) mainly because of its (i) completeness (deals with complex types of messages that we frequently find in real-world services), (ii) accuracy (Tibermacine et al., 2014) and (iii) the existence of a tool support (available online: https://code.google.com/p/wssim/). Nevertheless, the process is generic, and any other similarity assessment method, such as the method of Liu et al. (2010) or the approach of Garriga et al. (2013), can be adopted for similarity assessment in the proposed substitute identification process.

3. The proposed approach

This section covers the proposed process for substitute identification. First, we give an overview about the proposed process. Then, we present an example that we use for illustrating the application of the identification process. Finally, we describe in full details, within the remaining subsections, the different components of the process.

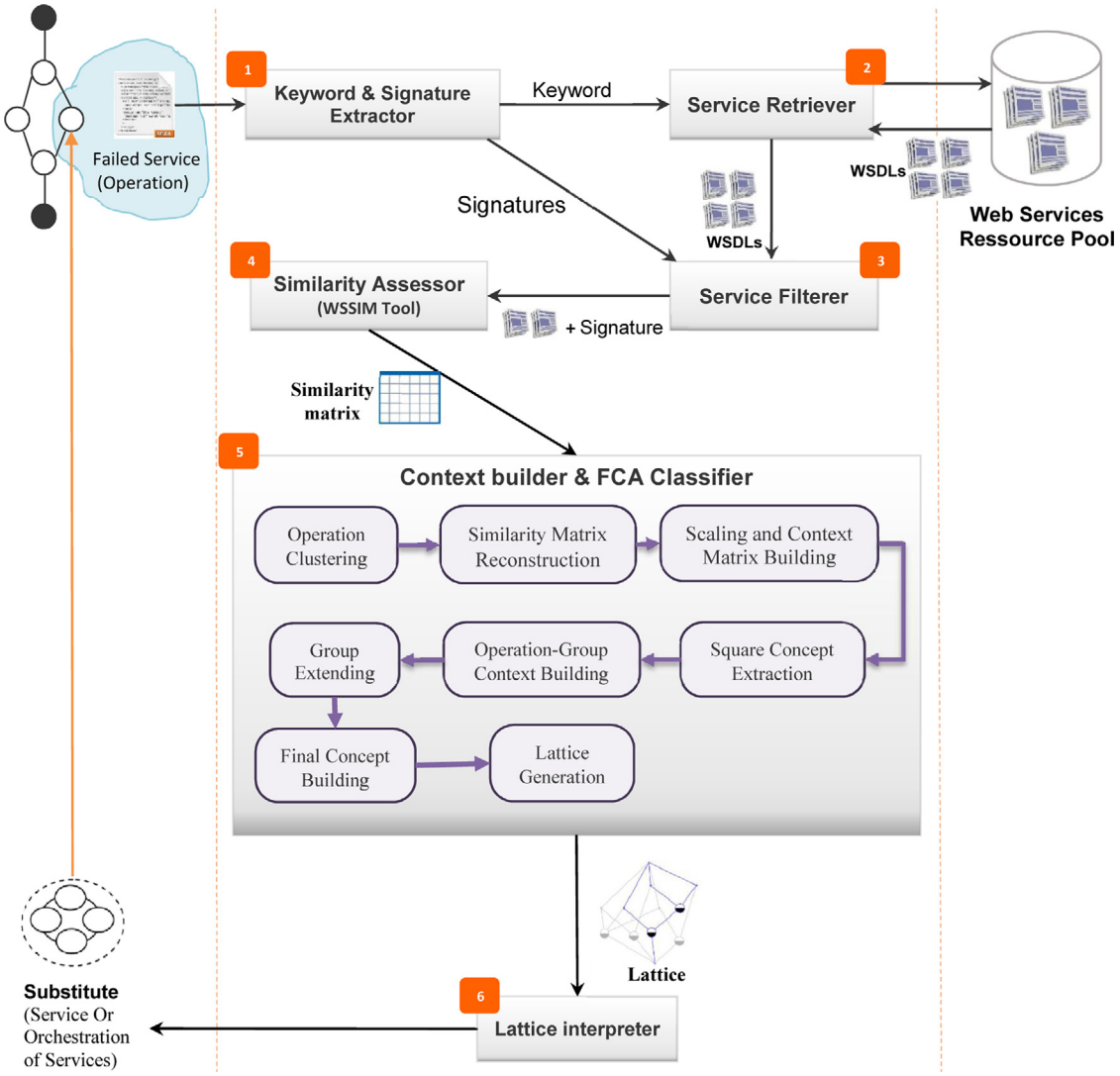

Fig. 2. Global schema of the substitute identification process.

3.1. Overview

As mentioned previously, our approach is built upon two techniques; the first is the similarity assessment between web services that enables to evaluate similarity and composability relationships between a set of web service candidates; and the second technique is FCA, which is used for classifying the compared services to retrieve substitutes. The identification process is depicted in Fig. 2. It starts by analyzing the WSDL document of the failed web service by the first component (Component 1 in Fig. 2). This component extracts, from the WSDL specification a set of representative keywords and the signature of the concerned operation. Then, the service retriever (Component 2 in Fig. 2) uses the set of keywords to select from a web service pool, all possible similar services, i.e. it selects web services that hold at least one of the keywords and may offer the same or related functionalities. Next, the service filterer (Component 3 in Fig. 2) analyzes the retrieved service candidates. It compares, through similarity assessment, the operation signatures of these services with the signature of the failed one. The component keeps in the filtered service set only services that have operations which are similar to the failed service, or services that have a composability relationship with the similar services.

The next step is the construction of the similarity matrix between the elements of the filtered set and the failed service. Hence, Component 4 in Fig. 2 assesses the similarity scores between all operation inputs and outputs (messages). The generated similarity matrix allows the identification of similarity relationships between all operations. Indeed, the similarity matrix can reveal simple substitutes. Nonetheless, the classification of operations in related groups can reveal also hidden similarity dependencies between these operations, which enables an identification of complex substitutes. For this end, the approach uses the FCA technique.

The context builder and the FCA classifier (Component 5 in Fig. 2) transforms the similarity matrix into a formal concept after many adjustments on the matrix itself. First, it groups similar operations in clusters, which reduces the size of the matrix and enhances, at the end of the FCA classification, the visualization and the interpretation of the generated lattice. Second, it rebuilds the similarity matrix based on the identified groups. Next, the component builds the context matrix corresponding to the reconstructed similarity matrix. After that, the FCA classifier analyzes the context matrix to select square concepts. In fact, square concepts are concepts with equal intention (operations, input and output messages) and extension (operations, input and output messages) sets. These concepts allow the component to construct another context called the operation-group context. In this context, the extensions are the service operations (or cluster representatives), and the intentions are groups of operations identified as square concepts. Afterward, the component extends these intentions

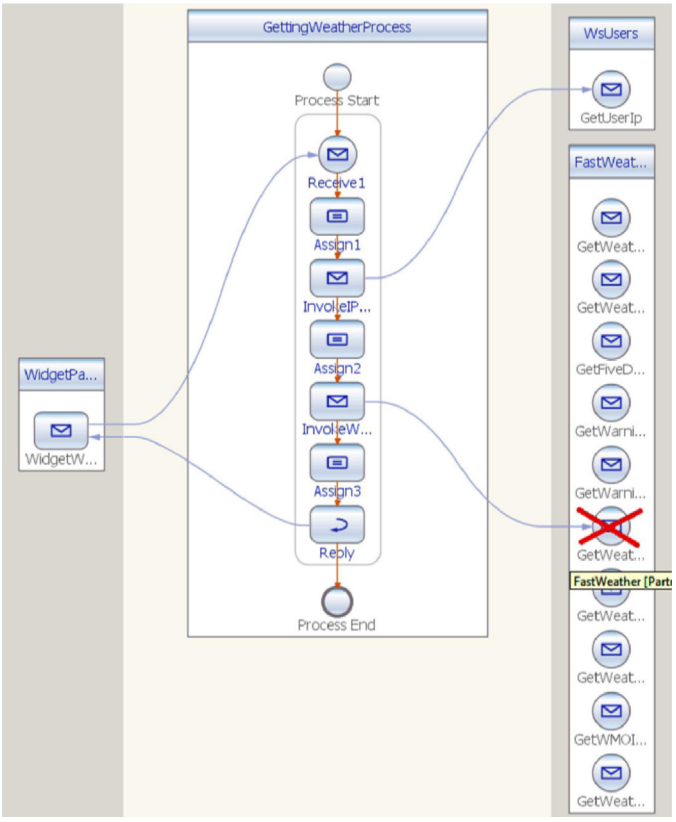

Fig. 3. Abstract BPEL description for the weather widget example.

by adding similarity relationships to the elements of each group. Finally, the component builds the final concept matrix and generates its corresponding operation lattice.

The lattice interpreter (Component 6 in Fig. 2) browses the lattice and interprets its content. It focuses on analyzing the extension of the failed operation to extract the list of simple and complex substitutes.

3.2. Illustrative example

We use for illustration a weather widget example. The widget uses an orchestration of web services to show weather information of the user's region based on its IP address. The widget interacts mainly with the web service WWS (*WidgetWeatherService*). This web service itself invokes an orchestration of two services GWP (*GettingWeatherProcess*); the WU (*WsUsers*) web service that affords IP information, and the FW (*FastWeather*) web service that returns weather information. Fig. 3 shows the BPEL (Business Process Execution Language) abstract description associated to this example. Now, let us assume that the invocation of the operation marked with a red cross in the figure (*getWeatherByIP*) initiates an error due to the unavailability of the service FW. This situation directly leads to the defection of the orchestration GWP. One possible solution, to heal this orchestration, is to find a substitute for the failed service (FW). Then, we reconstruct the orchestration using the identified substitute. In the next sections, we use the weather widget as a running example for illustrating our process' components.

3.3. Keyword and signature extractor

The identification process starts from Component 1 in Fig. 2. The component parses and analyses the WSDL interface description of the failed web service to extract a set of keywords. These keywords represent semantically the functionalities implemented by the web service. The set of keywords are used as criteria for the selection of web service candidates that may offer equivalent functionalities. Moreover, the component extracts the signature of the failed operation. Later, this signature is used for similarity evaluation and candidate filtering, where the system removes all candidates, which do not hold similar operations or do not have similarity relationships with similar operations.

3.3.1. Keyword extraction

Technically, the keyword and signature extractor component retrieves the representative keyword set from the parsed WSDL document through the following steps:

- The component retrieves all the identifiers from the WSDL file. Then, it adds them to an identifier set (*IdSet*).
- The component tokenizes the identifiers of the *IdSet*. It adds the extracted tokens to the Keyword Set (*KeywordSet*).
- The component treats the *KeywordSet* as follows :
  - It removes all redundant words.
  - It removes all stop words.
  - It stems words.
  - Finally, it enriches the set by adding words synonyms.

Thus, the component produces a set of keyword (*KeywordSet*) that represents the failed web service.

3.3.2. Signature extraction

Component 1 extracts also the signature of the failed operation after parsing the whole WSDL document. The signature is produced in textual format that includes operations identifier and the signature of its input and output messages. By its turn, the signature of messages holds their identifiers and a list of simple and complex parameters.

3.4. Service retriever

The service retriever (Component 2 in Fig. 2) uses the elements of the keyword set to search for web service candidates that may offer the same functionalities of the failed service. The component retrieves these candidate services by either seeking available resource pool using Algorithm 1, or directly by requesting web service engines such as Service Xplorer.[2]

Algorithm 1 takes as input the initial *keywordSet* (extracted from the failed service by Component 1), and a set of available services in the service pool (*ResPoolSet*). The algorithm analyzes each web service in the pool. If an analyzed service does not hold any keyword similar to those in the keywordSet, the service is shifted to the analyzed service set (*examinedServSet*). Otherwise, the service is shifted to the selected service set (*selectedServSet*), and its keywords are added to the *keywordSet* (lines 14–16). A re-analysis of the previously analyzed but not selected services is necessary, because the new added keywords could belong to one of the analyzed services. Consequently, the elements of the *examinedServSet* are shifted back to the *ResPoolSet* (lines 18–22). Finally, the algorithm returns a set of selected service candidates.

Note that within the returned service set, we have at least one service that shares some keywords with the failed web service. The other services share keywords between each other, which is interpreted by the existence of dependencies (similarity or composability relationships) between them. Components 3 and 4 check and evaluate respectively these dependencies.

[2] Service Xplorer: http://eil.cs.txstate.edu/ServiceXplorer/.

Algorithm 1 Candidate_Selection.

```
Input: KeywordSet, ResPoolSet
Output: SelectedServSet
Begin
 1: Boolean matches;
 2: Create wordSet';
 3: Create examinedServSet;
 4: while (ResPoolSet != φ) do
 5:     wsdl = getElement(ResPoolSet);
 6:     wordSet' = Keyword_Extraction(wsdl);
 7:     matches = false;
 8:     for all (String word in wordSet') do
 9:         if (Contains(KeywordSet, word)) then
10:             matches = true; break;
11:         end if
12:     end for
13:     if (matches) then
14:         for all (String w in wordSet') do
15:             Add w To KeywordSet;
16:         end for
17:         Add wsdl To SelectedServSet;
18:         while (examinedServSet != φ) do
19:             wsdl' = getElement(examinedServSet);
20:             Add wsdl' To resPoolSet;
21:             DeleteElement wsdl' from examinedServSet;
22:         end while
23:     else
24:         Add wsdl To examinedServSet;
25:     end if
26:     DeleteElement wsdl from ResPoolSet;
27: end while
End
```

3.5. Service filterer

The filterer component refines the set of web service candidates. It removes services that do not have a similarity relationship with the failed service according to the signature provided by Component 1. The similarity relation is interpreted by the fact that a given service holds an operation that is similar by its input or output message to the input or the output message of the failed operation/service. Or, it is similar by its input or output message to the input or the output message of an operation/service already evaluated as similar to the failed one.

The component uses the filtering algorithm (Algorithm 2). This algorithm takes as input the signature of the failed operation (extracted by Component 1), the selected service set (retrieved by Component 2) and a similarity threshold that represents the accepted lower limit to consider two operations as similar. The algorithm keeps in the filtered services set (*filteredServiceSet*) all services that hold an operation that is similar to the failed one, or services that hold operations that are similar to operations that are similar, by their turn, to the failed one.

To illustrate the remaining steps, we use the weather widget example. The service selector component retrieves a set of 142 services from a total of 3792 web services including 15,927 operations in the resource service pool. Then, the service filterer component reduces the number of selected services to 7. Fig. 4 summarizes the information about the concerned operations in the filtered services while the non-similar operations are ignored. The failed service in the example is "*FastWeather*" service (the file W15 is not included in Fig. 4), and the failed operation in the orchestration is "*GetWeatherByIp*".

3.6. Similarity assessor

The role of the similarity assessor component is to measure the similarity values between input and output messages that belong to the operations in the filtered service candidates. The component arranges these values in a similarity matrix. Fig. 5 depicts the generic form of this matrix. Practically, the component uses WsSim tool[3] for the similarity measurement. However, the component is generic, and any other similarity assessment approach can be integrated.

Algorithm 2 Operation_Filtering.

```
Input: Signature, selectedServSet, similarity Threshold θ
Output: FiltredServiceSet
Begin
 1: Boolean matches;
 2: Create examinedServSet;
 3: while (selectedServSet != φ) do
 4:     wsdl = getElement(selectedServSet);
 5:     parsedWsdl = wsdlParser(wsdl);
 6:     matches = false;
 7:     for (int i = 0; i <parsedWsdl.operationCount; i++) do
 8:         operationInfo = parsedWsdl.getOperation(i);
 9:         if (Sim(signature.InputMessage, operationInfo.InputMessage)≥ θ ||
            Sim(signature.InputMessage, operationInfo.OutputMessage)≥ θ ||
            Sim(signature.OutputMessage, operationInfo.InputMessage)≥ θ ||
            Sim(signature. OutputMessage, operationInfo.OutputMessage)≥ θ )
            then
10:             matches = true; break;
11:         end if
12:     end for
13:     if (matches) then
14:         Add wsdl To FiltredServiceSet;
15:         Boolean matches';
16:         Create examinedServSet';
17:         examinedServSet' = Duplicates(examinedServSet);
18:         while (examinedServSet' != φ) do
19:             wsdl' = getElement(examinedServSet');
20:             parsedWsdl' = wsdlParser(wsdl');
21:             matches' = false;
22:             for (int i =0; i <parsedWsdl.operationCount; i++) do
23:                 for (int j =0; j <parsedWsdl'.operationCount; j++) do
24:                     operationInfo = parsedWsdl.getOperation(i);
25:                     operationInfo' = parsedWsdl'.getOperation(j);
26:                     if (Sim(operationInfo.InMessage,
                        operationInfo'.InMessage)≥
                        θ ||Sim(operationInfo.InMessage,
                        operationInfo'.OutMessage)≥
                        θ ||Sim(operationInfo.OutMessage,
                        operationInfo'.InMessage)≥
                        θ ||Sim(signature.OutputMessage,
                        operationInfo.OutputMessage)≥ θ ) then
27:                         matches' = true; break;
28:                     end if
29:                 end for
30:             end for
31:             if (matches') then
32:                 Add wsdl' To FiltredServiceSet;
33:                 DeleteElement wsdl' from examinedServSet;
34:             end if
35:             DeleteElement wsdl' from examinedServSet';
36:         end while
37:     else
38:         Add wsdl To examinedServSet;
39:     end if
40:     DeleteElement wsdl from selectedServSet;
41: end while
End
```

Actually, the elements to consider for the interpretation of the similarity matrix (Fig. 5) are the following:

- $WS = \{Ws_y | 1 \leq y \leq l\}$. *WS* is the filtered Web service set (filteredServSet). Ws_i is the service number *i* in the filtered service set.
- $OP = \{Op_x | 1 \leq x \leq n\}$. *OP* is the operation set. $\forall Op_x, Op_x \in Ws_y | 1 \leq y \leq l$; i.e. all the operations in the operation set are belonging to a service in the Web Service Set.
- The function Sim(*A*, *B*) evaluates the similarity between messages *A* and *B* where :
  - A,B ∈ MS; MS = $\{Op_i.InputMessage \bigcup Op_i.OutputMessage | Op_i \in OP\}$. MS is the message set that groups the input and output messages of the operations.
  - ∀ A∈ MS,∀ B ∈ MS, A = B ⇒ Sim(A,B) = 1.
  - ∀ A∈ MS,∀ B ∈ MS, A ≠ B ⇒ Sim(A,B)∈ [0, 1].
- $Op_j \in$ OP and $Op_k \in$ OP are the compared operations and both belong to the operation Set.

[3] Available online: https://code.google.com/p/wssim/.

File	Service Name	Input		Output	
		Name	Parameter	Name	Parameter
IP02.xml (IP2Geo)	IP2Geo	ResolveIp	IpAdress :String Licence : String	ResolveIp	City : String
IP07.xml (GeometryInfo)	Geometry	IPQuery	Ip : String key : String	IPQuery	Code : String
IP15.wsdl (p2LocationWebService)	IP2Loc	IP2Location	IpAdress :String Licence : String	IP2Location	City : String
W02.wsdl (WeatherByZip)	Weather	GetWeatherByZip	Zip : String	GetWeatherByZip	CityTo1ZipResult : String
w03.wsdl (Weather)	Weather	GetWeather	Zip : String	getWeatherforZipCode	getWeatherfor-ZipCodeResponse : String Array
w06.WSDL (WeatherService)	Weather	GetWeather	Zip : String	getWeatherReturn	getWeatherReturn : String Array
Z23.asmx (ZIP)	ZIPCode	CityToZipCode	City : String	CityToZipCode	CityToZipCode : String

Fig. 4. Filtered web services information.

		Op_1		Op_2		...	Op_n	
		Input	Output	Input	Output	⋮	Input	Output
Op_1	Input	1		S_{I1_I2}	S_{I1_O2}	...	S_{I1_In}	S_{I1_On}
	output		1	S_{O1_I2}	S_{O1_O2}	...	S_{O1_In}	S_{O1_On}
Op_2	Input	S_{I2_I1}	S_{I2_O1}	1		...	S_{I2_In}	S_{I2_On}
	Output	S_{O2_I1}	S_{O2_O1}		1	...	S_{O2_In}	S_{O2_On}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Op_n	Input	S_{In_I1}	S_{In_O1}	S_{In_I2}	S_{In_O2}	...	1	
	output	S_{On_I1}	S_{On_O1}	S_{On_I2}	S_{On_O2}	...		1

Fig. 5. A generic form for the similarity matrix (SimMatrix).

- $S_{Ij\_Ik} = \text{Sim}(Op_j.InputMessage, Op_k.InputMessage)$, S_{Ij_Ik} is the similarity score between the input message of operation Op_j and the input message of operation Op_k.
- $S_{Oj\_Ik} = \text{Sim}(Op_j.OutputMessage, Op_k.InputMessage)$, S_{Oj_Ik} is the similarity score between the output message of operation Op_j and the input message of operation Op_k.
- $S_{Ij\_Ok} = \text{Sim}(Op_j.InputMessage, Op_k.OutputMessage)$, S_{Ij_Ok} is the similarity score between the input message of operation Op_j and the output message of operation Op_k.
- $S_{Oj\_Ok} = \text{Sim}(Op_j.OutputMessage, Op_k.OutputMessage)$, S_{Oj_Ok} is the similarity score between the output message of operation Op_j and the output message of operation Op_k.

Moreover, we give the following definitions to figure out similarity and substitutability relations between operations from the similarity matrix.

Definition 1. (Similar Operations): we consider two operations as similar if and only if the similarity value between operation inputs and the similarity value between operation outputs; both are greater than or equal to a threshold θ'. θ' is fixed experimentally. Mathematically, $\forall Op_x \in OP, \forall Op_y \in OP$, and $x \neq y$, $\text{Sim}(Op_x.\text{inputMessage}, Op_y.\text{inputMessage}) \geq \theta'$, and $\text{Sim}(Op_x.\text{inputMessage}, Op_y.\text{inputMessage}) \geq \theta' \Leftrightarrow Op_x \equiv Op_y$. Op_x is similar (equivalent) to Op_y.

Definition 2. (1-to-1 substitute): from Definition 1, similar operations to a failed operation are 1-to-1 substitute to this operation; If Op_f is the failed operation, $\forall\, Op_x \in OP, f \neq x$ and $Op_f \equiv Op_x$ then Op_x is 1-to-1 substitute to Op_f. These two definitions allow operation clustering and similarity matrix reconstruction which are explained in the next subsections.

For illustration, Fig. 6 shows the similarity matrix (*SimMat*) between the input and the output messages of the operations presented in Fig. 4, and the input and output messages of the failed operation (*getWeatherByIp*) in the service (W15). The matrix *SimMat* is an instance of the generic matrix depicted in Fig. 5. Lines and columns in *SimMat* represent inputs and outputs of the concerned operations (each service is represented by one operation in this example). The cells of *SimMat* hold the similarity values between lines and columns (operation's input/output, input/input, output/input or output/output).

3.7. Context builder and FCA classifier

The context builder and FCA classifier is Component 5 in Fig. 2. This component classifies and visualizes the operations that belong to service candidates in a lattice. This organization exploits the similarity matrix to reveal substitution relationships between operations, in a navigable way within a generated lattice. The context builder and the FCA classifier conduct many transformations depicted inside the component (Component 5, Fig. 2). It starts by grouping similar operations in clusters, which directly identifies simple (1-to-1) substitutes because operations in the same cluster represent substitutes for each other.

Next, the component reduces the lines and columns number in the similarity matrix based on the constructed clusters, which reduces consequently the lattice size, and hence the complexity of its interpretation. Then, the component uses the reduced similarity matrix for building its associated context matrix that includes the input and output messages in addition to their operations.

Afterward, the classifier analyzes the context matrix to recognize the square concepts, i.e. the maximal collections of messages

	W15.I	W15.O	IP02.I	IP02.O	IP07.I	IP07.O	IP15.I	IP15.O	W02.I	W02.O	W03.I	W03.O	W06.I	W06.O	Z23.I	Z23.O
W15.I	1		0,935	0,579	0,702	0,192	0,65	0,103	0,381	0,577	0,577	0,172	0,577	0,172	0,491	0,362
W15.O		1	0,108	0,423	0,551	0,184	0,511	0,519	0,611	0,684	0,612	0,966	0,612	0,739	0,254	0,119
IP02.I	0,935	0,108	1		0,65	0,116	0,65	0,113	0,362	0,095	0,376	0,571	0,376	0,571	0,119	0,502
IP02.O	0,579	0,423		1	0,544	0,758	0,507	0,687	0,095	0,121	0,093	0,395	0,093	0,571	0,901	0,124
IP07.I	0,702	0,551	0,65	0,544	1		0,68	0,304	0,435	0,263	0,447	0,27	0,447	0,286	0,565	0,565
IP07.O	0,192	0,184	0,116	0,758		1	0,582	0,681	0,252	0,604	0,277	0,248	0,277	0,212	0,895	0,121
IP15.I	0,65	0,511	0,65	0,507	0,68	0,582	1		0,472	0,378	0,472	0,262	0,472	0,279	0,128	0,312
IP15.O	0,103	0,519	0,113	0,687	0,304	0,681		1	0,227	0,373	0,277	0,492	0,277	0,549	0,912	0,135
W02.I	0,381	0,611	0,362	0,095	0,435	0,252	0,472	0,227	1		0,983	0,417	0,983	0,463	0,547	0,789
W02.O	0,577	0,684	0,095	0,121	0,263	0,604	0,378	0,373		1	0,64	0,683	0,64	0,683	0,535	0,483
W03.I	0,577	0,612	0,376	0,093	0,447	0,277	0,472	0,277	0,983	0,64	1		1	0,465	0,382	0,756
W03.O	0,172	0,966	0,571	0,395	0,27	0,248	0,262	0,492	0,417	0,683		1	0,51	0,769	0,246	0,176
W06.I	0,577	0,612	0,376	0,093	0,447	0,277	0,472	0,277	0,983	0,64	1	0,51	1		0,382	0,756
W06.O	0,172	0,739	0,571	0,571	0,286	0,212	0,279	0,549	0,463	0,683	0,465	0,769		1	0,246	0,321
Z23.I	0,491	0,254	0,119	0,901	0,565	0,895	0,128	0,912	0,547	0,535	0,382	0,246	0,382	0,246	1	
Z23.O	0,362	0,119	0,502	0,124	0,565	0,121	0,312	0,135	0,789	0,483	0,756	0,176	0,756	0,321		1

Fig. 6. Case study similarity matrix (SimMat)).

(intentions) that share the same similarity relationships with other messages (extensions). Then, the component forms groups of similar messages. Each group holds the objects (input and output messages) of the corresponding square concept. These groups maintain the similarity relationships between messages of the same group, which represent possible compositions between operation output (output message) and another operation input (input message).

In order to show these composition relationships, the component builds what we call Operation-Group context matrix. The objects in this context matrix are the operations and the attributes are the groups of messages extracted from the square concepts. In the next step, the component extends the groups of messages (the attributes of the last context matrix) by adding for each input message its output message, and for each output message its similar messages. This step is crucial before building the final context because it identifies all composition sequences (which operation could be composable with another). These composition sequences present, during the visualization of the lattice, the complex (*N*-to-1) substitutes.

After that, the component builds the final concepts based on the extended groups. These concepts hold the relationships between operations and their possible substitutes. Finally, the classifier generates the lattice that corresponds to the final context. This lattice classifies operations in groups that share common attributes. In this case, the attributes hold the possible substitutes for each operation. Component 6 interprets the lattice and extracts complex (*N*-to-1) substitutes.

In the following sections, we detail each transformation step using our illustrative example.

3.7.1. Operation clustering

In this step, the component groups operations in a set of clusters using Algorithm 3. This algorithm analyzes the similarity matrix for constructing clusters of similar operations. The operations are grouped in the same cluster if their similarity values are greater than or equal to a given similarity threshold (lines 15 and 16 of Algorithm 3). The threshold is fixed experimentally. It represents the lower accepted limit for the similarity value to consider two compared objects (messages), in the matrix, as similar.

On the one hand, the identification of clusters reduces the complexity of the computation in the next steps. On the other hand, it reveals simple (1-to-1) substitutes; i.e. operations in the same cluster which are simple substitutes for each other.

For instance, if we consider the operation clustering for our example using this algorithm, the similarity matrix (depicted in Fig. 6), with a similarity threshold θ' that is equal to 0.65 ($\theta' = 0.65$), produces the three clusters (IP, W, and Z23) shown in Fig. 7.

Afterward, the similarity matrix has to be reconstructed based on the identified clusters as we will explain in the following section.

Algorithm 3 Operation_Clustering.

Input: *OP, SimMatrix, similarity Threshold θ'*
Output: *Clusters*
Begin

```
1:  Boolean Added;
2:  int nbc = 0;
3:  while (OP != φ) do
4:      op = getElement(OP);
5:      Added = false;
6:      DeleteElement op from OP;
7:      if (nbc == 0) then
8:          nbc ++;
9:          Create cluster_nbc;
10:         Add op To cluster_nbc;
11:     else
12:         int i; Added = false;
13:         while (i ≤ nbc) do
14:             op' = getElement(cluster_i);
15:             if (GetSimScore(SimMatrix, op.InputMessage, op'.InputMessage)≥
                θ' && GetSimScore(SimMatrix, op.OutputMessage,
                op'.OutputMessage)≥ θ') then
16:                 Add op To cluster_i;
17:                 Added = true;
18:                 break;
19:             else
20:                 i++;
21:             end if
22:             if (!Added) then
23:                 nbc++;
24:                 Create Cluster_nbc;
25:                 Add op To Cluster_nbc;
26:             end if
27:         end while
28:     end if
29: end while
End
```

$Cluster_1(\boldsymbol{IP}) = \{IP_{02}, IP_{07}, IP_{15}\}$
$Cluster_2(\boldsymbol{W}) = \{W_2, W_4, W_6\}$
$Cluster_3(\boldsymbol{Z23}) = \{Z_{23}\}$

Fig. 7. Clusters identified from SimMat.

	W15.I	W15.O	IP.I	IP.O	W.I	W.O	Z23.I	Z23.O
W15.I	1		0,935	0,579	0,381	0,577	0,491	0,362
W15.O		1	0,108	0,423	0,611	0,684	0,254	0,119
IP.I	0,935	0,108	1		0,362	0,095	0,119	0,502
IP.O	0,579	0,423		1	0,095	0,121	0,901	0,124
W.I	0,381	0,611	0,362	0,095	1		0,547	0,789
W.O	0,577	0,684	0,095	0,121		1	0,535	0,483
Z23.I	0,491	0,254	0,119	0,901	0,547	0,535	1	
Z23.O	0,362	0,119	0,502	0,124	0,789	0,483		1

Fig. 8. Clusters similarity matrix (*CLSimMat*).

3.7.2. Similarity matrix reconstruction

The component reconstructs the original similarity matrix (*SimMat*) based on the identification clusters. The new generated matrix is the Cluster Similarity Matrix (*CLSimMat*). The component reduces the number of lines and rows in the original matrix (*SimMat*). Thus, it keeps one operation from each cluster, and it removes the remaining operations. Note that each operation is represented by two lines (two columns) in the matrix. The first line (column) represents its input message and the second its output message.

For instance, Fig. 8 shows the cluster similarity matrix (*CLSimMat*) which resulted from the reconstruction of the similarity matrix *SimMat* depicted in Fig. 6, based on the identified clusters presented in Fig. 7.

3.7.3. Scaling and context matrix building

In this step, the component scales the matrix according to a given similarity threshold θ'. The component removes, from the cluster similarity matrix (*CLSimMatrix*), all the values that are less or equal to the threshold θ'. The removed values are presented by blank cells in the similarity matrix (e.g., matrix (a) in Fig. 9). The deletion of these values represents the elimination of all relationships between the messages that are not considered as similar. Thus, they will not appear in the context matrix nor in the generated lattice later.

Then, the component replaces all the remaining values in the matrix which are replaced by 'x' to obtain the context matrix (*ContextMat*). For illustration, Fig. 9 shows in (a) the scaled version of the *CLSimMatrix* presented in Fig. 8, and in (b) its context version based on the similarity threshold θ' that is equal to 0.65.

We recall that both objects and attributes of the built context (presented by *ContextMat*) are input and output messages, and the relationship between objects and attributes is the similarity relationship.

3.7.4. Square concepts extraction

In this step, the component analyzes the context matrix (*ContextMat*) to extract groups of messages that have mutual similarity relationships. These groups of messages are used by the component for detecting composability relationships between operations. These groups are presented in the context matrix by square concepts. By definition, a square concept is a collection of objects with equal extension and intention sets. They are better viewed in the context matrix by interchanging lines and columns. For instance, Fig. 10 shows the square concepts marked by the crosses in the interchanged context matrix (a). In addition, the component forms four corresponding groups as it is depicted in part (b) of the figure.

	W15.I	W15.O	IP.I	IP.O	W.I	W.O	Z23.I	Z23.O
W15.I	1		0,935					
W15.O		1				0,684		
IP.I	0,935		1					
IP.O				1			0,901	
W.I					1			0,789
W.O		0,684				1		
Z23.I				0,901			1	
Z23.O					0,789			1

(a) The scaled Matrix

	W15.I	W15.O	IP.I	IP.O	W.I	W.O	Z23.I	Z23.O
W15.I	x		x					
W15.O		x				x		
IP.I	x		x					
IP.O				x			x	
W.I					x			x
W.O		x				x		
Z23.I				x			x	
Z23.O					x			x

(b) The Context matrix (*ContextMat*)

Fig. 9. Scaled and context matrix.

	W15.I	IP.I	W15.O	W.O	IP.O	Z23.I	W.I	Z23.O
W15.I	x	x						
IP.I	x	x						
W15.O			x	x				
W.O			x	x				
IP.O					x	x		
Z23.I					x	x		
W.I							x	x
Z23.O							x	x

(a) The Interchanged Context Matrix (*IContextMat*)

G1 = $\{W_{15}.I, IP.I\}$
G2 = $\{W_{15}.O, W.O\}$
G3 = $\{IP.O, Z_{23}.I\}$
G4 = $\{W.I, Z_{23}.I\}$

(b) The Identified groups

Fig. 10. Square concepts, Interchanged Context Matrix and identified groups. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

	{W15.I , IP.I}	{W15.O , W.O}	{IP.O , Z23.I}	{W.I , Z23.O}
W15	x	x		
IP	x		x	
W		x		x
Z23			x	x

Fig. 11. Operation-group context matrix (OPGContextMat).

Iteration 0 : (Initial groups)

G1={W_{15}.I , IP.I}
G2={ W_{15}.O, W.O}
G3={IP.O, Z_{23}.I}
G4={W.I, Z_{23}.O}

Iteration 1 :

G1'={W_{15}.I-***W_{15}***.O , *IP*.I- ***IP***.O }
G2'={ W_{15}.O, *W*.O}
G3'={*IP*.O, Z_{23}.I- **Z_{23}.O** }
G4'={W.I - **W.O** , Z_{23}.O}

Iteration 2 :

G1'={W_{15}.I-***W_{15}***.O , *IP*.I- ***IP***.O=>**Z_{23}.I** }
G2'={ W_{15}.O, *W*.O}
G3'={*IP*.O, Z_{23}.I- **Z_{23}.O => W.I** }
G4'={W.I - **W.O** , Z_{23}.O}

.....

Iteration 4 :

G1'={W15.I - **W15.O** , IP.I - **IP.O => Z23.I - Z23.O =>W.I - W.O**}
G2'{W15.O , W.O}
G3' {IP.O , Z23.I - **Z23.O =>W.I - W.O**}
G4'= {W.I - **W.O** , Z23.O}

Fig. 12. Extended groups.

3.7.5. Operation-group context building

According to the groups identified in the previous step, the component builds a new context matrix called the Operation-Group Context Matrix (*OPGContextMat*). The objects of the new context are the operations. The attributes are the groups of mutually similar messages (objects of the square concepts) identified previously. The relationships between the objects and the attributes are the membership of at least one message in the attribute to one operation. Fig. 11 shows the *OPGContextMat* constructed for our example based on the groups identified in part (b) of Fig. 10.

3.7.6. Group extending

In this step, the component extends the attributes of the *OPGContextMat* to hold composition sequences. These consequences show the composition relationships between the operations (e.g., operation 1 is composed with operation 2 and the latter is composed with operation 4 and so on). These consequences are built based on the similarity between messages identified from the square concepts identified in Section 3.7.4.

The component extends the attributes (extension of the *OPGContextMat*) by applying the following steps:

- **Step 1**: for every operation input, add its output (e.g. $Op_1.In$ becomes $Op_1.In - Op_1.Out$).
- **Step 2**: for every operation output, add its similar inputs by looking at the initial groups (e.g. $Op_5.Out$ becomes $Op_5.Out \Rightarrow Op_1.Out$).
- **Step 3**: repeat step 1 and step 2 until no change occurs.

The application of these steps on the groups, depicted in part (b) of Fig. 10, is illustrated in Fig. 12. The component produces 4 extended groups (shown in Fig. 12). These groups contain different composition sequences such as "Z23.I- Z23.O ⇒ W.I-W.O" in group G3'.

3.7.7. Final Context Building

In this step, the Context Builder and the FCA Classifier component uses the extended groups as attributes for building the final context matrix (*FinContMat*). The objects of *FinContMat* are the same operation candidates. Crosses are added in this context if one operation has at least one message in the corresponding attribute.

For instance, Fig. 13 shows the final context of the illustrative example.

3.7.8. FCA classification and lattice generation

In the final transformation step, Component 5 uses Concept Explorer (Yevtushenko, 2000) to generate the operation lattice corresponding to the final formal context elaborated previously. The lattice classifies operations in related groups. Each group has common attributes (intention). In this lattice, the attributes contain the possible substitutes for each operation. Component 6 interprets the lattice and extracts complex (*N*-to-1) substitutes for the failed operation in the defected web service. Fig. 14 depicts the operation lattice associated to the final context shown in Fig. 13.

3.8. Lattice interpreter

The Lattice Interpreter (Component 6 in Fig. 3) queries the generated lattice for determining the substitutes of the failed operation. More precisely, the component parses the intent label of the failed operation to extract composition sequences that could be appropriate substitutes. The component uses the following steps for the interpretation of the intent label:

- **Step 1**: operation input and its output separated by the minus sign (−) are replaced by the name of the operation itself (e.g. $Op_1.In - Op_1.Out$ becomes Op_1).
- **Step 2**: the sign (⇒) between operations means composability between them (e.g. $Op_1 \Rightarrow Op_2$ means Op_1 can be composed with Op_2).
- **Step 3**: if the intent holds an orchestration (⇒ sign) and all the elements in that orchestration are operations (but not only the input or the output of operation as *Op.In* or *Op.Out*), then the sequence of operations (orchestration) is a potential substitute (*N*-to-1 substitute) for the selected object (failed operation).

For our example, if we select the object labeled *W*15 in the lattice (the failed operation), and we look at its intent (see Fig. 15), we obtain the following:

	{W15.I-W15.O , IP.I-IP.O => Z23.I-Z23.O =>W.I-W.O}	{W15.O , W.O}	{IP.O , Z23.I-Z23.O => W.I-W.O}	{W.I - W.O , Z23.O}
W15	x	x		
IP	x		x	
W	x	x	x	x
Z23	x		x	x

Fig. 13. Final concept matrix.

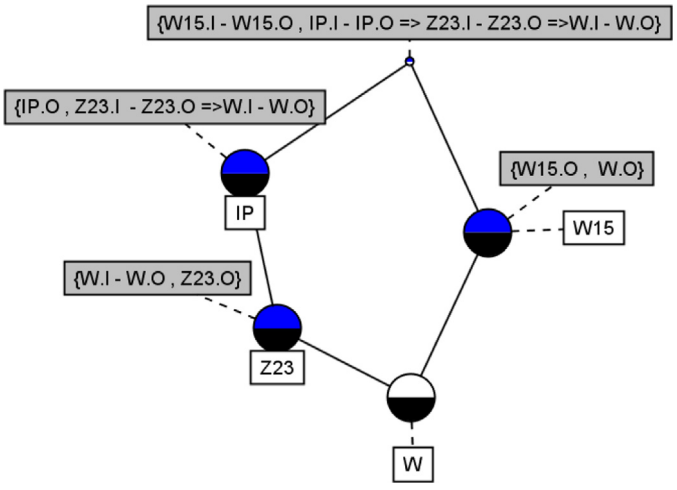

Fig. 14. Final lattice.

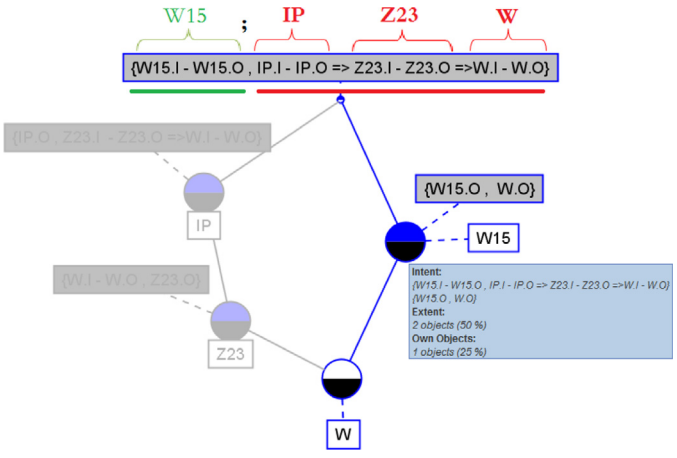

Fig. 15. Final lattice interpretation.

- **W15.O, W.O**: means that the output message of operation $W15$ is similar (equivalent) to the output message of operations in $cluster_2$(W). This does not offer any valuable information.
- **W15.I-W15.O, IP.I-IP.O ⇒ Z23.I-Z23.O ⇒ W.I-W.O**: if we follow step 1 we obtain **W15, IP ⇒ Z23 ⇒ W**. The first part (parts are separated by semicolons) does not offer any valuable information because operation **W15** is similar to itself. For the second part, if we follow steps 2 and 3 we get an orchestration of three operations **IP ⇒ Z23 ⇒ W**. This orchestration is a substitute for the failed operation **W15**.

Additionally, IP and W represent $Cluster_1$ and $Cluster_2$. So, elements in the same cluster can be used interchangeably. Consequently, we obtain a set of substitutes (6 orchestrations), as illustrated in Fig. 16.

The lattice interpreter component returns a set of substitutes for the failed operation. These substitutes are simple operations (1-to-1 substitutes), and sets of operations that form composite sequences which are considered as complex (N-to-1) substitutes.

4. Experiment and validation

According to the previous descriptions, the key concept of the approach is the similarity between a web service orchestration and single web service candidates (similarity between the input/output of the orchestration with the input/output of the substitute). So, starting from a given similarity degree (threshold), we consider the service as equivalent/substitute to the orchestration. Thus, if any of these

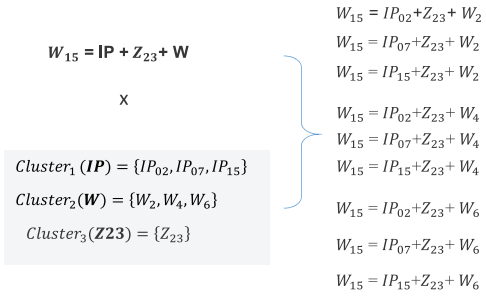

Fig. 16. List of (N-to-1) substitutes.

substitutes fails, it effortlessly could be replaced by its equivalent orchestration (1-to-N substitution) and vice-versa (N-to-1 substitution). We performed some experiments on a set of real web services to evaluate the approach efficiency and effectiveness. The experiments are conducted following four steps: data set selection, orchestration extraction, substitute extraction, and finally experiment evaluation.

We focus on the validation of the idea of complex substitute identification, because the selection of simple substitutes is a particular case in our approach that has been already addressed in the literature (details in Section 5). Before starting this experiment, we have made an investigation on the number of operations that have no simple substitutes in the WS-Dream dataset (Zhang et al., 2011). We have found, using WsSim tool, that among 15,952 operations in 3794 web services, there are 3809 operations with no simple substitute (23.88%). This rate motivates our interest to the validation of complex substitute identification.

4.1. Methodology

We have conducted the experimental process on following these steps:

- Select a set of web services (dataset),
- Extract possible combinations that could be considered as service orchestrations,
- Manually check the obtained combinations to define the set of orchestrations,
- Extract possible substitutes for each orchestration,
- Manually check the obtained substitutes,
- Measure results in terms of well know metrics (recall, precision, accuracy and F1-score).

4.2. Data selection

In this experiment, we have selected 64 web service WSDL documents from the WS-Dream dataset (Zhang et al., 2011). In fact, the data set contains 3792 web services, but we limited the number of used services to 64 to be able to check manually the obtained results in reasonable time. The first service is a weather service, and the remaining services are selected among the others using the selection and filtering algorithms presented previously. The list of the used services and experiment results are available for download in the following website: https://sites.google.com/site/wservicesubstitues/.

4.3. Orchestration extraction

From the dataset, we extracted all possible orchestrations from the web services. We assessed the similarity values between these web services, more precisely between their operations. We fixed the composability threshold (θ) at 0.95; which means that two operations are considered composable if the similarity between the output of the first operation and the input of the second operation is greater or equal to 0.95. We selected this score after several experiments to obtain a

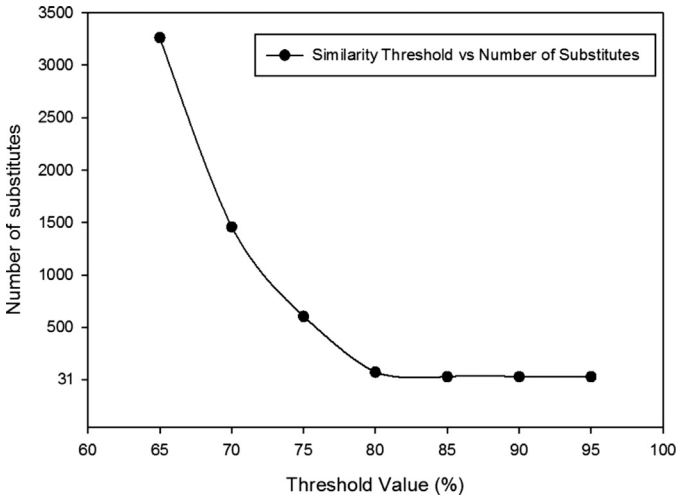

Fig. 17. Number of obtained substitutes.

reduced number of orchestrations. Using this composability threshold ($\theta = 0.95$) we obtained 113 orchestrations. The manual verification of these extracted orchestrations validated the composability between these combinations. Lower thresholds allowed the selection of a larger number of combinations. But a manual adaptation between inputs and outputs is needed to consider these combinations as acceptable.

4.4. Substitute extraction

In the next set of experiments, we used different similarity thresholds to automatically extract possible substitutes for each element in the orchestration set. Fig. 17 shows the values of the extracted substitutes according to different similarity threshold values (θ). Obviously, we obtain a larger selection once we use relaxed similarity threshold values ($\theta \leq 0.70$ and $\theta \geq 0.60$). Nevertheless, the manual examination shows that relaxed threshold values leads to obtain a lot of substitutes that need a manual adaptation (parameters of complex type to adapt). Consequently, we fixed the threshold value (θ) to 0.75.

4.5. System performances

The machine used for this experiment is running with an Intel processor (I3-2100 CPU 3.10 GHZ) and RAM of 4 GB under Windows 7 Operating System (64 bits). Fig. 18 summarizes the execution times for each run depending on the chosen similarity threshold value. Execution times in part (a) of Fig. 18 include execution times for WSDL parsing, similarity measurement between services, similarity matrix reconstruction, combination of operation extraction, and finally matrix analysis and substitute identification. Part (b) of the same figure depicts the obtained values for the WSDL parsing and substitute extraction times.

The WSDL document parsing time ranges from 0.3 to 0.4 s for the same number of WSDL files. The analysis of the similarity matrix to identify substitutes ranges from 0.07 to 1.4 s. These values depend on the matrix size which is correlated with the number of operations selected based on the used similarity threshold. The values depicted in part (a) of Fig. 18 are the execution times of the experiment runs. In each run, the WSDL documents are parsed, the possible combinations (orchestration) are extracted, the similarity matrix is constructed and then it is analyzed and the substitutes are identified based on the fixed similarity threshold. These execution times range from 360 to 372 s. They are inversely proportional with the similarity threshold, thus, with the number of selected operations.

4.6. Result measurement

Finally, we measure the effectiveness of the approach by calculating some information retrieval metrics such as recall, accuracy and precision. These metrics have been broadly used in the context of web service discovery and selection (Garriga et al., 2013; Rodriguez et al., 2010).

In Fig. 19, table (a) shows the number of extracted substitutes according to the fixed thresholds. Column “By tool” depicts the number of the substitutes that are extracted automatically; that are considered by the tool as correct substitutes for the orchestrations. Column “Manually checked” contains the number of substitutes that are manually verified. These values are used to identify:

- The number of true positives (number of substitutes that are identified by the tool and that are correct),
- The number of the false positives (number of substitutes that are identified by the tool and that are incorrect),
- The number of false negatives (number of substitutes that are correct but that are not identified by the tool),
- The number of true negatives (number of substitutes that are incorrect and that are not identified by the tool).

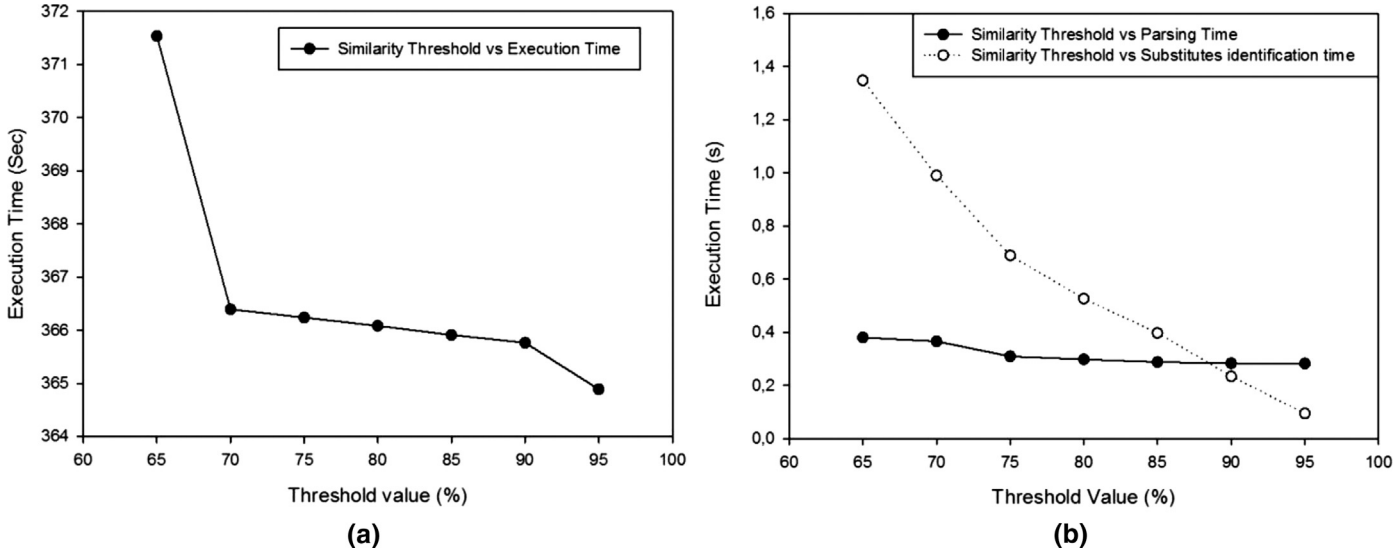

Fig. 18. Experiments execution time.

	Number of orchestrations	Number of substitutes	
		By Tool	Manually checked
Θ=0.95 Θ'=0.75	113	605	442
Θ=0.95 Θ'=0.73		645	452

(a)

TP (True Positive)	FP (False Positive)
442	163
FN (False negative)	TN (True Negative)
10	30

(b)

Fig. 19. Result of manual verification of extracted substitutes.

Values summarized in table (b) of Fig. 19 are used to calculate the following IR metrics:

- **Precision**: is the number of true results divided by the number of all returned results.
  Precision $(P) = \frac{TP}{TP+FP} = \mathbf{0.731}$
- **Accuracy**: is the number of true results (both true positives and true negatives) in the obtained results.
  Accuracy $(A) = \frac{TP+TN}{TP+FP+FN+TN} = \mathbf{0.732}$
- **Recall**: is the number of correct results divided by the number of results that should be returned.
  Recall $(R) = \frac{TP}{TP+FN} = \mathbf{0.98}$.
- **F1-Score**: is a measure of test accuracy. It can be interpreted as a weighted average of both precision and recall.
  F1-Score $= 2 \times \frac{P \times R}{P+R} = \mathbf{0.836}$.

The experiment shows that the precision rate is relatively low because of the number of the false positives. These false positives are the operations that are identified by the tool and considered as substitutes for some orchestrations. The manual verification shows that these operations are semantically different than the orchestration, but the tool selected them as substitutes because their input or output messages are syntactically similar to the input of the first operation in the orchestration or, respectively, the output of the last operation in the orchestration. However, the experiment shows that the accuracy rate is very high, and this is very important in the case of identification of substitutes.

In summary, the metric values obtained from this experiment on the one hand, and the case example that illustrated the section presenting the approach on the other hand, both show the practicability of the approach. Therefore, we can effectively retrieve (1-to-1) and (N-to-1) substitutes for failed services. Nevertheless, this experiment showed some drawbacks that we could avoid by taking into account the following:

- The better we measure the similarity and we fix its threshold the better results we obtain.
- Relaxed thresholds have to be avoided for substitutes identification.
- In case we choose medium thresholds, some substitutes need an adaptation (casting, complex type conversion, etc.) to replace failed operations. These service adaptations can be addressed at protocol level (Motahari Nezhad et al., 2007).

4.7. Threats to validity

Conclusion, construct, internal and external threats are the four categories of threats to validity proposed in Wohlin et al. (2000). An extension of this framework to cover Search-Based Software Engineering (SBSE) experiments is proposed by (Barros and Dias Neto, 2011). In this section, we discuss the threats to validity of the conducted experiment according to the former extension.

Conclusion validity threats have a concern with the relationship between the treatment and the outcome. The empirical design must make sure that there is a statistical relationship between the involved parts. The main conclusion threats include the non-consideration of random variation, the lack of good descriptive statistics and the luck of the use of a meaningful baseline (Barros and Dias Neto, 2011). In this experiment, we addressed the first threat by having many runs for each fixed threshold to measure the execution time. We note here that the number of experiment runs did not affect the number of obtained substitutes. Moreover, we cope with the remaining conclusion threats by comparing the obtained results with the manually identified ones, which is considered as a solid comparison baseline.

Construct validity threats are concerned with the relation between theory and observation, ensuring that the treatment reflects the construct of the cause and that the outcome reflects the construct of the effect. In SBSE experiments, construct threats involve using invalid efficiency and effectiveness measures and not discussing the underlying model subject to optimization (Barros and Dias Neto, 2011). We coped with these threats by discussing the cost measures of the experiments (the execution time). In addition, we addressed the validity of the effectiveness measures using the recall, precision and accuracy metrics, which are widely used in such experiments.

Internal validity threats are concerned with the evaluation of the causality of the relationship between the treatment and the outcome in an experimental study, or the result of a factor upon which the researcher has no control. Internal threats may include: (1) poor parameter settings, (2) luck of discussion on code instrumentation, (3) luck of clear data collection procedures, and finally (4) the luck of a real problem instance (Barros and Dias Neto, 2011). In this experiment, we cope with these threats by presenting the most important parameter (similarity and composability thresholds) used in the experiment, by providing the source code of the similarity assessor used in the approach, by conducting the experiment using a set of real web services, and finally by describing the data collection procedure.

External validity threats are concerned with the generalization of the observed results to a larger population, outside the sample instances used in the experiment. Specifically, these threats include the lack of a clear definition of target instances, the lack of clear instance selection strategy, and the fact of not having enough diversity in instance size and complexity (Barros and Dias Neto, 2011). In this experiment, the external threats come from the limited number of instances of selected web services used in the experiment. Even if we have selected these services from a large collection of (almost 3800) real web services, it is still difficult to generalize these results because they depend on the studied set of web services. Nevertheless, the goal of this experiment is to show the practicability and efficiency of the proposed substitute identification process on real-world data.

5. Related work

Many works have been proposed in the literature, focusing on different aspects of web service discovery, selection, classification, composition, composition healing and adaptation. In this section, we present a literature review for some of these research topics related to the work presented in this paper.

5.1. Similarity evaluation for service discovery and selection

The similarity evaluation between web services has been studied by many researchers for service discovery and selection. A survey with a comparative analysis between the proposed approaches has been presented in Kokash (2006) and Crasso et al. (2011). Most of

these works use Information Retrieval (IR) techniques to increase web service discovery precision without involving any additional level of semantic mark-up (Garriga et al., 2013).

In De Antonellis et al. (2006), the authors present their tool (ARTEMIS) which calculates a set of similarity coefficients to evaluate web service compatibility. The tool clusters similar services based on the obtained similarity coefficients.

Dong et al. present a search engine called "Woogle" (Dong et al., 2004). Based on similarity search, Woogle returns similar Web services for a given query. The search engine combines multiple techniques to evaluate similarity between the services and their operations. These techniques focus on operation parameters as well as operations and service descriptions. The authors introduced a clustering algorithm for grouping description terms in a set of concepts. After that, similarity between concepts is measured using a simple information retrieval metric; the TF/IDF metric.

The similarity evaluation in Kokash (2006) is implemented through combining lexical and structural matching. Likewise, in Plebani and Pernici (2009), the paper proposes a method for Web service retrieval called URBE (Uddi Registry By Example). The retrieval is based on the evaluation of similarity between Web service interfaces. The algorithm used in URBE combines the analysis of Web services structure and the terms used inside it.

Ait-Bachir (2008) studied the similarity measurement between behavioral interfaces of web services by simulation. Both structural and behavioral aspects of services are considered in this study. Structural aspects represent service operations, messages and their XML schema within the interface description document. Moreover, the behavioral aspect, which is presented by finite state machines, is defined by control flow and inter-dependencies between operations.

The approach presented in Crasso et al. (2008) proposes to discover the most relevant web services to a given query. The approach is based on the representation of a web service description and queries within classic space vectors. Then, it matches between the vectors that represent services and the vector which represents the query using the Cosine metric. It returns the nearest service to the given query.

Additionally, an approach for measuring the compatibility degree of services' protocols is proposed in Ouederni et al. (2011). The approach relies on a formal comparison that is based on generic-flooding techniques. The authors provide a formal model for describing web service interfaces with interaction protocols.

Another similarity measurement approach is proposed in Garriga et al. (2013) for service selection. The approach comprises an assessment process for service interface compatibility. The assessment process is based on a structural scheme for service matching. The scheme is divided into two main parts: automatic strong matching and semi-automatic potential matching. The former involves similarity cases directly recognized from Java interfaces of candidate services. The latter involves cases that could be solved through a semi-automatic assistance. The whole information package gathered from this process provides an important insight about candidate services and their required adaptations for integration.

The approach described in Tibermacine et al. (2013) uses different structure and semantic similarity metrics to conduct a similarity assessment between different WSDL parts of the compared web services (operation, messages, parameters, type, etc.). It uses also a schema matching technique to evaluate the similarity between input and output messages. The similarity function is parameterized by a set of weights to allow users determine which parts of a WSDL document have more impact on the similarity score. We adopt this approach in the current work for the similarity assessment between web services. Nevertheless, the identification process of service substitutes is a generic, and it is not limited by the use of a specific similarity measurement approach. So, any other measurement approach, such as Garriga et al. (2013), Kokash (2006), Plebani and Pernici (2009), could be incorporated during the similarity matrix construction between service candidates.

5.2. Fault recovery in web service compositions

Different kinds of faults may occur during service composition execution, and many strategies were proposed to repair the failed services. In Fugini and Mussi (2006), the authors present an approach for fault management in Web applications. The contribution is a self-healing system that holds all possible faults and their repair actions in a special registry. Authors present a reference architecture for faults treatments and a set of strategies for recovery. Moreover, a classification of faults has been studies and schematized. The core of the approach is based on searching substituting services for repairing compositions.

Dobson (2006) proposes to transform a BPEL process into a fault tolerance process using a fault tolerance pattern. The transformation is achieved by adding redundant behavior to the process. Baresi (Baresi and Guinea, 2005; Baresi et al., 2007) presents a supervision framework and a solution for self-healing BPEL processes based on Dynamo. The framework lies on the use of Aspect Oriented Programming techniques, the separation of concerns principle and a rule engine (JBoss rules) to allow recovery of faults in service composition.

WS-Diamond is a project for self-healing web services (Console and Team, 2007). It is based on a platform for observing symptoms in complex composed applications. It aims to diagnosis occurring faults, and for selection and execution of repair plans.

In Simmonds et al. (2010a, 2010b), the authors propose a framework for performing runtime monitoring of web service-based applications against behavioral correctness properties described as a finite-state automaton. The set of verified properties specify forbidden and desired interactions between services. The execution traces of web service-based applications described in BPEL are checked for conformance at runtime. The framework proposes different adaptation strategies in case of violation of properties.

A QoS-driven self-healing method for reliable web service composition was introduced in Dai et al. (2009). The method predicts QoS and performance during composition. It backups alternate web services during the selection step. Then, in case of failure, it re-selects from these backups based on QoS and performance predictions. Moreover, authors use a Semi-Markov process to predict the data transmission speed over the network where services are executing.

Behl et al. (2012) propose a replication architecture for executing workflows in fault-tolerant and configurable manner. In the proposed architecture, BPEL processes and Web services are actively replicated using output and input proxies. An automatic transformation of process definitions allows proxies to intercept the communication between services transparently. The proxies make use of a ZooKeeper service for coordination and dynamic configuration.

In this work, we focused on the selection of substitutes based on similarity measurement between a web service and a set of potential candidates. These substitutes that could be simple or complex serve as backups for the different partner links in the orchestration. Users can statically construct different execution scenarios based on the identified substitutes. These scenarios are the recovery plans in case of orchestration failure. Hence, our approach can complete the previously mentioned works.

5.3. Web services classification using concept lattices

Many works have addressed the classification of web services using concept lattices. In their paper (Peng and Chen, 2008), the authors present a formal definition of web service classification and retrieval using formal concept analysis. The essential of the approach is to build lattices using formal contexts where objects are web services and attributes represent the operations of these services. Then, they

retrieve similar services using algorithms that navigate the lattices. In order to understand relationships between web services, and among operations of complex services, Aversano et al. (2006) propose an approach based on Formal Concept Analysis. The approach analyzes service interfaces and documentations to build lattices. The generated lattices allow analysts to cluster similar services, highlight hierarchical relationships, and visualize, in general, similarities and differences between the analyzed services.

Restructuring of service registry at runtime using Formal Concept Analysis is studied in Chollet et al. (2010). The purpose behind this approach, as the authors claim, is to speed up the selection process of web services, and to improve decision making through the building of concept lattices. The service registry is viewed as a formal context where services are objects and service types are attributes. Non-functional characteristics (security) are considered as additional attributes.

The work proposed in Fenza and Senatore (2010) describes an approach for retrieving semantic web services, taking into account user's requirements and preferences. The approach exploits the fuzzy formal concept analysis for modeling concepts and relationships extracted from web service resources. The user formulates its query as conceptual terms, and through a conceptual based mechanism it returns the list of semantic web services that match the introduced query.

The authors in Driss et al. (2010) introduce a requirement-centric approach that allows modeling user's requirements, discovering and selecting web services. The authors use formal concept analysis only for selecting automatically relevant high QoS services.

In addition, the authors in Azmeh et al. (2008) elaborate a tool named WSPAB. The tool uses formal concept analysis to allow automatic discovery, classification, and selection of web services. The tool builds formal concepts where objects are web services and attributes are operation signatures. Then, it generates corresponding lattices that classify the studied web services. The same authors propose another approach to classify web services by keywords elicited from their WSDL documents (Azmeh et al., 2010). The approach clusters similar services using FCA, so it is possible later to identify relevant services and their substitutes from concept lattices.

An extension to the previous approach based on Rational Concept Analysis (RCA) is proposed to select composable web services driven by users requirements at the design phase (Azmeh et al., 2011). The approach is based on four main steps including: service collection, validation and compatibility filtering, QoS level calculation, and RCA classification. The resulting lattices group services that have common QoS and composition levels. User requirements are expressed as new services and they are classified in the corresponding lattices.

In fact, in Azmeh et al. (2011) the user has to introduce a requirement document that specifies an abstract process with the needed functionality and the expected QoS in each service, as well as the composability between each pair of services. Then, the approach selects concrete web services for each element in the abstract process. The selected services are simple and composable with each other according to the composability modes described initially in the requirement document. Certainly, this approach selects elements for building compositions, but the selected services are all simple. Moreover, the solution is static (number of services in the process is fixed) and totally guided by the user. In the opposite, the solution that we propose is used at the maintenance phase. It is dynamic, i.e. the approach retrieves service substitutes with a variable number of services in the composition for the same specification. In addition, the approach does not need any abstract description to guide the identification process; the approach browses automatically all composition possibilities and finds appropriate substitutes for a failed service described by its WSDL document.

Technically, we have updated the selection and filtering phases by proposing new similarity-based algorithms. Then, we used a more complex similarity method to measure the similarity between web service candidates. We focus on the similarity between input and output messages rather than the similarity between operations like Azmeh et al. did in their paper. The study of similarity between service inputs and outputs can reveal both composition and similarity relationships, but the similarity between operations reveals only similarity relationships which leads to the discovery of simple substitutes only. We also performed additional tasks for the classification of services (e.g. Group extending and the steps applied in this task and those applied during the interpretation task). This refined process enabled us to construct lattices that cluster operations and show their potential simple and complex substitutes.

6. Conclusion

In this paper, we proposed an approach for identifying relevant web service substitutes for healing failed service orchestrations. The approach relies on measuring the similarity between web service interfaces. We presented the necessary algorithms and techniques for selecting, filtering, and clustering web service candidates. We used a similarity matrix to determine the relation between services. We described the steps that aim to find simple substitutes (1-to-1) and complex substitutes (*N*-to-1). We incorporated FCA to classify and visualize the relevant results. We have shown the practicability of the approach using a case example. In addition, we validated the approach via a set of experiments conducted on a collection of real web services.

One of the original contributions of this work is the identification of complex substitutes; i.e. a set of services that can be orchestrated to replace a single service based on the similarity measurement between these services. The presented process is also generic, and other similarity assessment approaches can be incorporated in the substitute identification process.

As future works, we intend to study the similarity, composability and substitution between stateful web services. In fact, the current work focuses on conventional Web services, which are stateless in nature as they use request and response messages for communication, without keeping any state between requests. However, some web services applications require services to record their communication. These stateful services require a precise interaction protocol for session management, which makes the investigation for substitution between such services a hard task.

Moreover, we intend to integrate this approach in a fully automatic framework that ensures reliable web service compositions. The framework will hold among others; a web service monitor, an SLA (Service Level Agreement) controller, an exception handler, a substitute selector (based on the current work), a Quality of Service (QoS) evaluator, and a composition adapter.

References

Ait-Bachir, A., 2008. Measuring similarity of service interfaces. In: ICSOC PhD Symposium 2008, p. 59.

Alhosban, A., Hashmi, K., Malik, Z., Medjahed, B., 2013. Self-healing framework for cloud-based services. In: ACS International Conference on Computer Systems and Applications (AICCSA). IEEE, pp. 1–7.

Ardagna, D., Cappiello, C., Fugini, M., Mussi, E., Pernici, B., Plebani, P., 2006. Faults and recovery actions for self-healing web services. In: World Wide Web Conference.

Aversano, L., Bruno, M., Canfora, G., Di Penta, M., Distante, D., 2006. Using concept lattices to support service selection. Int. J. Web Serv. Res. (IJWSR) 3 (4), 32–51.

Azmeh, Z., 2011. A Web Service Selection Framework for an Assisted SOA (Ph.D. thesis). Montpellier 2.

Azmeh, Z., Driss, M., Hamoui, F., Huchard, M., Moha, N., Tibermacine, C., 2011. Selection of composable web services driven by user requirements. In: IEEE International Conference on Web Services (ICWS). IEEE, pp. 395–402.

Azmeh, Z., Huchard, M., Tibermacine, C., Urtado, C., Vauttier, S., 2008. WSPAB: a tool for automatic classification and selection of web services using formal concept analysis. In: IEEE Sixth European Conference on Web Services (ECOWS'08). IEEE, pp. 31–40.

Azmeh, Z., Huchard, M., Tibermacine, C., Urtado, C., Vauttier, S., 2010. Using concept lattices to support web service compositions with backup services. In: Fifth International Conference on Internet and Web Applications and Services (ICIW). IEEE, pp. 363–368.
Baresi, L., Guinea, S., 2005. Dynamo: dynamic monitoring of WS-BPEL processes. In: Proceedings of the International Conference on Service-Oriented Computing (ICSOC05). Springer, pp. 478–483.
Baresi, L., Guinea, S., Pasquale, L., 2007. Self-healing BPEL processes with dynamo and the JBoss rule engine. In: International Workshop on Engineering of Software Services for Pervasive Environments: In Conjunction with the 6th ESEC/FSE Joint Meeting. ACM, pp. 11–20.
Barros, M.d.O., Dias Neto, A.C., 2011. Threats to Validity in Search-Based Software Engineering Empirical Studies. Tech report 0006/2011. UNIRIO - Universidade Federal do Estado do Rio de Janeiro.
Behl, J., Distler, T., Heisig, F., Kapitza, R., Schunter, M., 2012. Providing fault-tolerant execution of web-service-based workflows within clouds. In: Proceedings of the 2nd International Workshop on Cloud Computing Platforms. ACM, p. 7.
Chollet, S., Lestideau, V., Lalanda, P., Moreno-Garcia, D., Colomb, P., 2010. Heterogeneous service selection based on formal concept analysis. In: 6th World Congress on Services (SERVICES-1). IEEE, pp. 367–374.
Console, L., Team, W.D., 2007. WS-diamond: an approach to web services-diagnosability, monitoring and diagnosis. In: International e-Challenges Conference, The Hague, October 2007.
Crasso, M., Zunino, A., Campo, M., 2008. Query by example for web services. In: Proceedings of the 2008 ACM Symposium on Applied Computing. ACM, New York, NY, USA, pp. 2376–2380. http://doi.acm.org/10.1145/1363686.1364251.
Crasso, M., Zunino, A., Campo, M., 2011. A survey of approaches to web service discovery in service-oriented architectures. J. Database Manage. (JDM) 22 (1), 102–132.
Dai, Y., Yang, L., Zhang, B., 2009. QoS-driven self-healing web service composition based on performance prediction. J. Comput. Sci. Technol. 24 (2), 250–261.
De Antonellis, V., Melchiori, M., Plebani, P., 2006. An approach to web service compatibility in cooperative processes. In: International Symposium on Applications and the Internet Workshops (SAINTW'06). IEEE Computer Society, p. 95.
Ding, Y., Xiang, R., 2013. A service-oriented exception handling method based on exception classification. In: International Conference on Sensor Network Security Technology and Privacy Communication System (SNS & PCS). IEEE, pp. 63–68.
Dobson, G., 2006. Using WS-BPEL to implement software fault tolerance for web services. In: 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, 2006. SEAA'06. IEEE, pp. 126–133.
Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J., 2004. Similarity search for web services. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases – Volume 30, VLDB '04. VLDB Endowment, pp. 372–383.
Driss, M., Moha, N., Jamoussi, Y., Jézéquel, J.M., Ghézala, H.H.B., 2010. A requirement-centric approach to web service modeling, discovery, and selection. In: Service-Oriented Computing. Springer, pp. 258–272.
Fan, J., Kambhampati, S., 2005. A snapshot of public web services. SIGMOD Rec. 34 (1), 24–32.
Fenza, G., Senatore, S., 2010. Friendly web services selection exploiting fuzzy formal concept analysis. Soft Comput. 14 (8), 811–819.
Fugini, M.G., Mussi, E., 2006. Recovery of faulty web applications through service discovery. In: Proceedings of the 1st SMR-VLDB Workshop, Matchmaking and Approximate Semantic-based Retrieval: Issues and Perspectives, 32nd International Conference on Very Large Databases, pp. 67–80.
Ganter, B., Stumme, G., Wille, R., 2005. Formal Concept Analysis: Foundations and Applications, vol. 3626. Springer.
Ganter, B., Wille, R., 1999. Contextual attribute logic. Springer.
Garriga, M., Flores, A., Mateos, C., Zunino, A., Cechich, A., 2013. Service selection based on a practical interface assessment scheme. Int. J. Web Grid Serv. 9 (4), 369–393.
Kokash, N., 2006. A comparison of web service interface similarity measures. Front. Artif. Intell. Appl. 142, 220.
Levenshtein, V., 1966. Binary codes capable of correcting deletions, insertions and reversals. Sov. Phys. Dokl. 10, 707.
Li, Y., Melliti, T., Dague, P., 2007. Modeling BPEL web services for diagnosis: towards self-healing web services. In: WEBIST (1), pp. 297–304.
Liu, F., Shi, Y., Yu, J., Wang, T., Wu, J., 2010. Measuring similarity of web services based on wsdl. In: IEEE International Conference on Web Services (ICWS). IEEE, pp. 155–162.
Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F., 2007. Semi-automated adaptation of service interactions. In: Proceedings of the 16th International Conference on World Wide Web. ACM, pp. 993–1002.
Ouederni, M., Salan, G., Pimentel, E., 2011. Measuring the compatibility of service interaction protocols. In: Proceedings of the 2011 ACM Symposium on Applied Computing. ACM, pp. 1560–1567.
Peng, D., Chen, Q., 2008. An efficient approach for managing replaceability of web services. In: Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid. IEEE Computer Society, Washington, DC, USA, pp. 388–391. 10.1109/skg.2008.21.
Pirró, G., 2009. A semantic similarity metric combining features and intrinsic information content. Data Knowl. Eng. 68, 1289–1308. 10.1016/j.datak.2009.06.008.
Plebani, P., Pernici, B., 2009. URBE: web service retrieval based on similarity evaluation. IEEE Trans. Knowl. Data Eng. 21 (11), 1629–1642.
Psaier, H., Dustdar, S., 2011. A survey on self-healing systems: approaches and systems. Computing 91 (1), 43–73.
Rodriguez, J.M., Crasso, M., Zunino, A., Campo, M., 2010. Improving web service descriptions for effective service discovery. Sci. Comput. Program. 75 (11), 1001–1021.
Simmonds, J., Ben-David, S., Chechik, M., 2010a. Guided recovery for web service applications. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, pp. 247–256.
Simmonds, J., Ben-David, S., Chechik, M., 2010b. Monitoring and recovery of web service applications. In: Smart Internet, LNCS. Springer, pp. 1–35.
Stoilos, G., Stamou, G., Kollias, S., 2005. A string metric for ontology alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (Eds.), Proceedings of the 4th International Semantic Web Conference (ISWC). Springer, Berlin, Heidelberg, pp. 624–637.
Tibermacine, O., Tibermacine, C., Cherif, F., 2013. WSSim: a tool for the measurement of web service interface similarity. In: French-Speaking Conference on Software Architectures (CAL'13).
Tibermacine, O., Tibermacine, C., Cherif, F., 2014. A practical approach to the measurement of similarity between WSDL-based web services. RNTI: Revue des Nouvelles Technologies de l'Information Special Issue CAL 2013 (RNTI-L-7), pp. 3–18.
Wille, R., 2009. Restructuring lattice theory: an approach based on hierarchies of concepts. In: Ferré, S., Rudolph, S. (Eds.), Formal Concept Analysis. Volume 5548: Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 314–339.
Winkler, W.E., 1990. String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. In: Proceedings of the Section on Survey Research, pp. 354–359.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell, MA, USA.
Yevtushenko, S.A., 2000. System of data analysis concept explorer. In: Proceedings of the 7th National Conference on Artificial Intelligence KII, vol. 2000.
Zhang, Y., Zheng, Z., Lyu, M.R., 2011. WSPred: a time-aware personalized QoS prediction framework for web services. In: IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE). IEEE, pp. 210–219.

Okba Tibermacine is a lecturer and Ph.D. student in computer science at Biskra University (Algeria), graduated from the same university with B.Sc. (engineer) degree, and received his M.S. (magister) degree in computer science from Batna University, Algeria. His current research interests include SOA architectures, Reliability analysis, Error-handling and Self-healing Web service compositions.

Chouki Tibermacine is an associate professor at Montpellier University (France) since fall 2007. He received his Ph.D. from the University of South Brittany (France) in 2006 and his M.Sc. in Distributed Systems from the University of Paris VI (France) in 2003. His current research focuses on the specification, evolution and transformation of component-based and service-oriented software architectures and programs. He supervised several Ph.D. and Master theses which have been successfully defended these last years. He participated to several research projects with industrial (IBM, among others) and academic partners. He co-authored about thirty peer-reviewed articles and received the ACM SIGSOFT Distinguished Paper Award at CBSE'11 and CBSE'14. He is holding (for the period 2012–2016) the scientific excellence fellowship from the University of Montpellier.

Cherif Foudil is an Associate Professor of computer science at Computer Science Department, Biskra University, Algeria. Dr. Cherif holds Ph.D. degree in computer Science. The topic of his doctoral dissertation is Behavioral Animation: simulation of a crowd of virtual humans. He also possesses B.Sc. (engineer) in computer science from Constantine University 1985, and an M.Sc. in computer science from Bristol University, UK 1989. He is currently the head of LESIA Laboratory. His current research interest is in artificial intelligence, artificial life, crowd simulation and software engineering.