

# Many-Valued Concept Lattices for Backing Composite Web Services

Zeina Azmeh<sup>1</sup>, Marianne Huchard<sup>1</sup>, Nizar Messai<sup>2</sup>, Chouki Tibermacine<sup>1</sup>,  
Christelle Urtado<sup>3</sup>, and Sylvain Vauttier<sup>3</sup>

<sup>1</sup> LIRMM - CNRS UMR 5506 - Université de Montpellier II - Montpellier (France)  
{azmeh, huchard, tibermacin}@lirmm.fr

<sup>2</sup> UMR 7503 LORIA, BP 239, 54506 Vandoeuvre-lès-Nancy, (France)  
messai@loria.fr

<sup>3</sup> LGI2P - Ecole des Mines d'Alès, Nîmes, (France)  
{Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr

**Abstract.** A Web service is a software functionality accessible through the network. Web services are intended to be composed to realize domain-specific applications. Achieving a required functionality by a service composition necessitates the discovery of a collection of Web services out of the enormous service space. Each service from this service space must be examined to verify its functionality, which makes the discovery task neither efficient nor practical. Moreover, when a service in a composition becomes unavailable or functionally broken, the whole composition may become broken too. Therefore, a functionally equivalent service must be discovered, in order to replace the broken service, thus spending more time and effort. In this paper, we propose an approach that facilitates the discovery of a web service and the identification of its candidate substitutes. We use many-valued concept lattices to classify web services, depending on the similarity estimated between their operations. This classification enables the identification of a needed web service as well as its possible alternatives. Thus, a Web service composition can be achieved more easily and can be supported with backup services, to recover the functionality of a broken service.

## 1 Introduction

A Web service is a software functionality accessible through the network. It exposes its functionality to the external world by an abstract interface expressed by Web Service Description Language, WSDL<sup>4</sup>. A WSDL interface is a uniform mechanism for describing a service's available operations, parameters, data types, and access protocols.

According to the functionalities described by the WSDL interfaces, web services can be assembled to serve a particular function, solve a specific problem, or deliver a particular solution. They represent the building blocks for creating composite applications. When creating a composite application, each selected

<sup>4</sup> <http://www.w3.org/TR/wsdl>

web service must fulfil a part of the application's functionality. Therefore, each service's WSDL must be analyzed to verify its provided operations, and so to decide whether to select the service or not. Due to the big number of web services that exist on the web nowadays, the task of finding an appropriate service to use becomes hard and time-consuming. After identifying the needed services, they can be assembled together in order to form the functionality of the aimed composite application. This application will be functional as long as each of its composing services is functional, but once one of its services breaks, the part of the application represented by this service will break too, causing a total or partial dysfunctionality to the application. The obvious solution in this case, will be to search for another web service to replace the broken one and recover the missing functionality. Thus, the task of finding an appropriate web service to use has to be repeated every time a service breaks and has to be replaced by an equivalent one.

In this paper, we propose an approach for web service browsing, in order to facilitate the discovery of a web service and the identification of its candidate backups. We use the formal concept analysis (FCA), a mathematical formalism that permits the identification of groups of objects having common attributes. We adopted FCA in a number of our previous work [2, 1]. Using FCA, we construct web service lattices according to their functionality domain, depending on the similarity estimated between their operations.

Our generated service lattices represent a browsing mechanism that facilitates the discovery of a needed service, along with a set of possible backup services.

The rest of the paper is organized as follows: Section 2 defines the problem statement. In Section 3, we give a quick overview about formal concept analysis with its basic definitions. In Section 4, we explain our approach along with examples and formal definitions. In Section 5, we demonstrate a case study to explain the use of our approach. In Section 6, we list and discuss the related work. Finally, in Section 7, we conclude our paper and briefly give some of our perspectives.

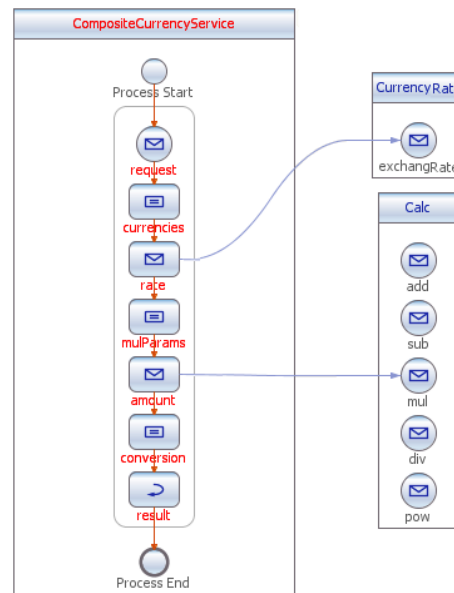
## 2 Problem Statement

Web services face several challenging issues coming from several factors. Since, they are offered by various providers, remotely accessed, and sometimes provided for free, there is no guarantee of a continuous execution. An available functioning web service may crash and become unavailable at any time, which necessitates finding an equivalent one to replace it. Unfortunately, this can be hard to achieve since there is a lack of WSDL organizing facilities, especially after the deficiency of the UDDI<sup>5</sup> registries: "UDDI did not achieve its goal of becoming the registry for all Web Services metadata and did not become useful in a majority of Web Services interactions over the Web" [18].

Thus, a mechanism for finding service backups becomes indispensable, especially when a service represents a part of a composite application.

<sup>5</sup> [http://www.uddi.org/pubs/uddi\\_v3.htm](http://www.uddi.org/pubs/uddi_v3.htm)

Let's take the two following services: a currency converter service and a calculation service. The currency converter service offers an operation that returns the exchange rate between two entered currencies. The calculation service offers several operations for calculation, one of them return the multiplication of two entered numbers. We will link the exchange rate operation with the multiplication operation in order to build a composite service of the two mentioned service, we will call it the composite currency service. This new composite service will give us a converted amount from a currency to another. Fig. 1 shows us an overview of this composite service.



**Fig. 1.** The composite currency service.

In the composition of Fig. 1, if the service *Calc* crashes, for example, a service offering a multiplication operation equivalent to *mul()* must be searched and discovered. Thus to recover the missing functionality, and ensure the continuity of the composition.

In the following sections, we illustrate some background definitions about FCA in order to describe the classification of Web services into concept lattices.

### 3 FCA Overview

In our approach, we use the FCA formalism in order to construct a classification of web services. FCA is a mathematical theory that permits the identification of

groups of objects having common attributes [11]. FCA takes as input a given data set represented as a formal context and produces the set of all the formal concepts which form a concept lattice. A formal context is denoted by  $\mathbb{K} = (G, M, I)$  where  $G$  is a set of objects,  $M$  is a set of attributes, and  $I$  is a binary relation between  $G$  and  $M$  ( $I \subseteq G \times M$ ).  $(g, m) \in I$  denotes the fact that object  $g \in G$  is in relation through  $I$  with attribute  $m \in M$  (also read as  $g$  has  $m$ ). We adapt the FCA for web services, by considering that the set of objects represent web services and the set of attributes represent operations. In this way, a formal context of web services and operations becomes  $\mathbb{K} = (\mathbb{W}, \mathbb{O}, I)$ , where:

$$\mathbb{W} = \{ws_i \mid 1 \leq i \leq |\mathbb{W}| \wedge |\mathbb{W}| > 1\}$$

is the set of web services, we suppose that it must contain more than one web service. Each service offers a set of one or more operations, and the union of all of the sets of operations offered by the services forms the total set of operations:

$$ws_i = \{op_{ij} \mid 1 \leq i \leq |\mathbb{W}| \wedge 1 \leq j \leq |ws_i|\}$$

$$\mathbb{O} = \bigcup_{i \geq 1} ws_i$$

$(ws, op) \in I$  denotes the fact that the service  $ws \in \mathbb{W}$  provides the operation  $op \in \mathbb{O}$  (also read as  $ws$  has  $op$ ). Table 1 shows an example of a formal context  $(\mathbb{W}, \mathbb{O}, I)$  where  $\mathbb{W} = \{ws_1, ws_2, ws_3\}$  and  $\mathbb{O} = \{op_1, op_2, op_3, op_4, op_5\}$ .

**Table 1.** A formal context for  $\mathbb{W} \times \mathbb{O}$ .

	$op_1$	$op_2$	$op_3$	$op_4$	$op_5$
$ws_1$	×	×		×	
$ws_2$		×	×		×
$ws_3$	×	×	×	×	

Having a set of web services  $X \subseteq \mathbb{W}$ ,

$$X' = \{op \in \mathbb{O} \mid \forall ws \in X : (ws, op) \in I\}$$

is the set of common operations. In the same way, having the set of operations  $Y \subseteq \mathbb{O}$ ,

$$Y' = \{ws \in \mathbb{W} \mid \forall op \in Y : (ws, op) \in I\}$$

is the set of common web services. In our example,  $(\{ws_1, ws_3\})' = \{op_1, op_2, op_4\}$  and  $(\{op_3\})' = \{ws_2, ws_3\}$ .

A concept is a pair of sets  $(X, Y)$  where  $X \subseteq \mathbb{W}$  is called the extent,  $Y \subseteq \mathbb{O}$  is called the intent, and  $Y = (X)'$ ,  $X = (Y)'$ . Meaning that, a concept is a maximal collection of services offering similar operations. The set of all concepts is denoted by  $\mathfrak{B}(\mathbb{W}, \mathbb{O}, I)$ .

In our example,  $(\{ws_1, ws_3\}, \{op_1, op_2, op_4\})$  is a concept while  $(\{op_3\}, \{ws_2, ws_3\})$  is not. In fact, while  $(\{op_3\})' = \{ws_2, ws_3\}$ ,  $(\{ws_2, ws_3\})' = \{op_2, op_3\}$ . A concept  $(X1, Y1)$  is a subconcept of a concept  $(X2, Y2)$  if  $X1 \subseteq X2$ , which imposes a partial order relation on  $\mathfrak{B}(\mathbb{W}, \mathbb{O}, I)$  expressed as  $(X1, Y1) \leq (X2, Y2)$ . The partial order relation  $\leq$  can be used to build a structure, which is called a concept lattice. A concept lattice defines a hierarchical representation of services and operations, in which a certain concept inherits all the extents (services) of its descendants and all the intents (operations) of its ascendants. Fig. 2 illustrates the lattice built for the context shown in table 1, we notice that upper labels are the reduced intent sets (operations) and bottom labels are the reduced extent sets (services). In this lattice we can reveal the relationships between the present services, some of them are the following:

- $ws_1$ ,  $ws_2$  and  $ws_3$  offer the operation  $\{op_2\}$ , thus, they can replace each other for this operation
- $ws_1$  and  $ws_3$  offer the operations  $\{op_1, op_2, op_4\}$
- $ws_3$  can replace  $ws_1$  since it offers all of its operations in addition to  $op_3$
- $ws_2$  and  $ws_3$  offer together the operations  $\{op_2, op_3\}$
- $ws_2$  can replace  $ws_1$  for the operation  $op_2$

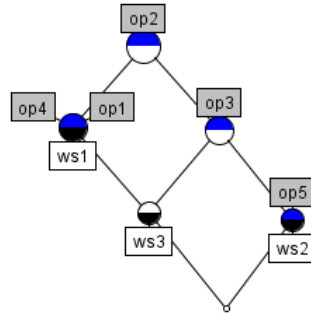


Fig. 2. The service lattice for the context in table1.

## 4 Classifying Web Services into Concept Lattices

In this section, we explain our approach using some basic formal definitions, along with an illustrative example. For clarity sake, we illustrate our approach using an imaginary set of Web services for performing calculations. Each service from this set is parsed by a WSDL parser to extract its signatures. The set of services with their signatures are given unique identifiers, as listed in table 2.

Next, a similarity measure must be chosen, and the operations signatures extracted from the WSDL files will be used by this similarity measure, according to its input format. Several similarity measures for web services exist in the

**Table 2.** A set of calculation services with their operations.

Services	Id	Operations	Id
Calculator1	$ws_1$	add(a,b)	$op_{11}$
		sub(a,b)	$op_{12}$
Calculator2	$ws_2$	add(a,b,c)	$op_{21}$
Calculator3	$ws_3$	add(a,b,c,d)	$op_{31}$
		sub(a,b,c)	$op_{32}$
		mult(a,b)	$op_{33}$
		add(a,b,c)	$op_{34}$

literature. They evaluate the similarity according to the syntactic and semantic levels, such like [10, 21, 14]. The similarity measure is applied on pairs of operations provided by different services. We do not consider the similarity between operations provided by the same service, because when a service becomes dysfunctional, all of its operations become dysfunctional too. The similarity is assessed in the form of values in the range [0,1]. If two operations are sufficiently similar, the similarity value will approach 1, or else it will approach 0.

A similarity measure (*Sim*) can be defined as follows:

$$\begin{aligned}
& Sim : \mathbb{O} \times \mathbb{O} \rightarrow [0, 1] \\
& \forall op_{ij} \in \mathbb{O} \implies Sim(op_{ij}, op_{ij}) = 1 \\
& \quad \text{(an operation with itself)} \\
& \forall op_{ij}, op_{ik} \in \mathbb{O} \implies Sim(op_{ij}, op_{ik}) = 0 \\
& \quad \text{(operations in the same service)} \\
& \forall op_{ij}, op_{nm} \in \mathbb{O} \implies Sim(op_{ij}, op_{nm}) \in [0, 1] \\
& \quad \text{(operations in different services)}
\end{aligned}$$

The calculated similarity values can be presented by a symmetric square matrix that we will call *SimMat*, as shown in table 3<sup>6</sup>. This matrix is of size  $n = |\mathbb{O}|$ , and its diagonal elements are all equal to 1, since we consider that the similarity of an operation with itself is equal to 1 as declared above.

From the similarity matrix *SimMat*, we can extract several binary contexts, by specifying threshold values  $\theta \in ]0, 1]$ . Thus, the values of *SimMat* that are greater or equal to the chosen threshold  $\theta$  are scaled to 1, while other values are scaled to 0. The binary context that corresponds to  $\theta = 0.75$  is shown in table 4, we call it *SimCxt*.

The *SimCxt* context is a triple  $(\mathbb{O}, \mathbb{O}, RSim_\theta)$ , where *RSim<sub>θ</sub>* is a binary relation indicating whether an operation is similar to another operation or not.

$$(op_{ij}, op_{nm}) \in RSim_\theta \iff Sim(op_{ij}, op_{nm}) \geq \theta$$

We use the *SimCxt* context to generate a lattice of operations,  $\mathfrak{B}(\mathbb{O}, \mathbb{O}, RSim_\theta)$ . This lattice helps in discovering groups of similar operations, which are used

<sup>6</sup> We can calculate the similarity values using any Web services similarity measure. Here we used a string distance measuring method, since similarity measuring is not the main objective of this paper

**Table 3.** The similarity matrix (*SimMat*).

	<i>op</i> <sub>11</sub>	<i>op</i> <sub>12</sub>	<i>op</i> <sub>21</sub>	<i>op</i> <sub>31</sub>	<i>op</i> <sub>32</sub>	<i>op</i> <sub>33</sub>	<i>op</i> <sub>34</sub>
<i>op</i> <sub>11</sub>	1	0	0.75	0.5	0	0	1
<i>op</i> <sub>12</sub>	0	1	0	0	0.75	0	0
<i>op</i> <sub>21</sub>	0.75	0	1	0.75	0	0	0.75
<i>op</i> <sub>31</sub>	0.5	0	0.75	1	0	0	0
<i>op</i> <sub>32</sub>	0	0.75	0	0	1	0	0
<i>op</i> <sub>33</sub>	0	0	0	0	0	1	0
<i>op</i> <sub>34</sub>	1	0	0.75	0	0	0	1

**Table 4.** The binary context (*SimCxt*) for  $\theta = 0.75$ .

	<i>op</i> <sub>11</sub>	<i>op</i> <sub>12</sub>	<i>op</i> <sub>21</sub>	<i>op</i> <sub>31</sub>	<i>op</i> <sub>32</sub>	<i>op</i> <sub>33</sub>	<i>op</i> <sub>34</sub>
<i>op</i> <sub>11</sub>	x		x				x
<i>op</i> <sub>12</sub>		x			x		
<i>op</i> <sub>21</sub>	x		x	x			x
<i>op</i> <sub>31</sub>			x	x			
<i>op</i> <sub>32</sub>		x			x		
<i>op</i> <sub>33</sub>						x	
<i>op</i> <sub>34</sub>	x		x				x

later on to construct the services lattice.

In the resulting operation lattice, groups of mutually similar operations can be identified by the concepts having equal extent and intent sets. We call such concepts as square concepts, because they form square gatherings on the binary context matrix. We define a group  $G_{op}$  of mutually similar operations  $Op_{Sim}$  as:

$$G_{op} = \{Op_{Sim} \mid (Op_{Sim}, Op_{Sim}) \in \mathfrak{B}(\mathbb{O}, \mathbb{O}, RSim_{\theta})\}$$

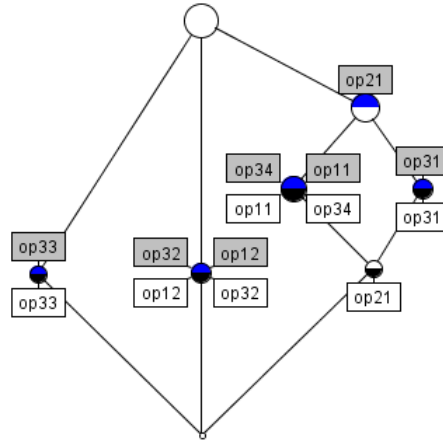
The notion of square concepts can be better recognized by performing a mutual column-line interchange in the *SimCxt*, the resulting interchanged context is shown in table 5.

**Table 5.** The interchanged (*SimCxt*) context.

	<i>op</i> <sub>11</sub>	<i>op</i> <sub>34</sub>	<i>op</i> <sub>21</sub>	<i>op</i> <sub>31</sub>	<i>op</i> <sub>12</sub>	<i>op</i> <sub>32</sub>	<i>op</i> <sub>33</sub>
<i>op</i> <sub>11</sub>	x	x	x				
<i>op</i> <sub>34</sub>	x	x	x				
<i>op</i> <sub>21</sub>	x	x	x	x			
<i>op</i> <sub>31</sub>			x	x			
<i>op</i> <sub>12</sub>					x	x	
<i>op</i> <sub>32</sub>					x	x	
<i>op</i> <sub>33</sub>							x

From the lattice in Fig. 3 as from the interchanged context in table 5, we can identify the groups of similar operations, and they are the following:

- $\{op_{11}, op_{34}, op_{21}\}$
- $\{op_{21}, op_{31}\}$
- $\{op_{12}, op_{32}\}$
- $\{op_{33}\}$



**Fig. 3.** The generated lattice for  $(SimCxt)$  shown in table 4.

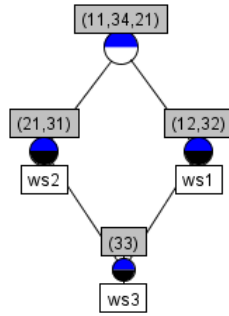
The groups of similar operations, denoted as  $\mathbb{G}$ , are used to define the final binary context. This context is a triple  $(\mathbb{W}, \mathbb{G}, R)$ , in which the relation  $R$  indicates whether or not a service offers the functionality represented by the corresponding group of similar operations.

We will represent each group of operations by an identifier that corresponds to the indices of the operations. The new context is shown in table 6: From the

**Table 6.** The final services  $\times$  groups context.

	(11,34,21)	(21,31)	(12,32)	(33)
$ws_1$	x		x	
$ws_2$	x	x		
$ws_3$	x	x	x	x

final binary context, we can generate the corresponding service lattice, which is shown in Fig 4.



**Fig. 4.** The final service lattice with possible backups.

### Lattice interpretation

From the final generated lattice, shown in Fig. 4, we can notice the following:

- $ws_1$ ,  $ws_2$ , and  $ws_3$  offer the functionality denoted by  $(11, 34, 21)$ , so they can replace each other for this specific functionality
- $ws_3$  can replace  $ws_1$  and  $ws_2$ , and it offers an additional functionality  $(33)$

We can also infer immediately which services offer a specific functionality (denoted by a specific label), by regarding the indices in the label.

## 5 Case Study

In this section, we demonstrate the use of the service lattices for the construction of composite web services and supporting them with backup services. We return to the example of the composite currency service, described in Section 2. We use BPEL<sup>7</sup> to orchestrate the two services by their operations, as shown in Fig. 1. We use many-valued contexts of similarity values as explained previously, in order to build two service lattices, a lattice for currency exchange services, and a lattice for calculation services. We retrieved a set of services<sup>8</sup> for currency conversion as shown in table 7 and for calculations as shown in table 8. We limit the number of services in this example, in order to simplify it and clearly explain the idea of lattice use.

We choose manually a value for similarity threshold in order to get a binary context. Then we obtain two lattices corresponding to each set of services. The two lattices are shown in the right side of Fig. 5. We can use the service lattices to build our composite service as well as to support it with backup services. We select operation  $op_{11}$ (CR:11) from the service s1 for exchange rate (currency lattice), and operation  $op_{13}$ (mul:13) from service s1 for multiplication (calculation

<sup>7</sup> <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>

<sup>8</sup> The services are obtained from the Seekda web service search engine, <http://www.seekda.com>

**Table 7.** The set of currency converter services.

Services	Id	Operations	Id
CurrencyConverter	<i>ws</i> <sub>1</sub>	GetConversionRate(fromCurrency,toCurrency)	<i>op</i> <sub>11</sub>
CurrencyConvertor	<i>ws</i> <sub>2</sub>	ConversionRate(FromCurrency,ToCurrency)	<i>op</i> <sub>21</sub>
DOTSCurrencyExchange	<i>ws</i> <sub>3</sub>	GetExchangeRate(ConvertFromCurrency,ConvertToCurrency)	<i>op</i> <sub>31</sub>
		ConvertCurrency(Amount,ConvertFromCurrency,ConvertToCurrency)	<i>op</i> <sub>32</sub>
		GetCountryCurrency(Country)	<i>op</i> <sub>33</sub>
CurrencyRates	<i>ws</i> <sub>4</sub>	GetRate(CurrencyCode)	<i>op</i> <sub>41</sub>
		GetConversion(FromCurrencyCode,ToCurrencyCode)	<i>op</i> <sub>42</sub>
RadixxFlights	<i>ws</i> <sub>5</sub>	GetExchange(FromCurrency,ToCurrency)	<i>op</i> <sub>51</sub>
		ConvertCurrency(Amount,FromCurrency,ToCurrency)	<i>op</i> <sub>52</sub>
rates	<i>ws</i> <sub>6</sub>	Convert(CurrencyFrom,CurrencyTo,ValueFrom)	<i>op</i> <sub>61</sub>
Conversion	<i>ws</i> <sub>7</sub>	CelciusToFahrenheit(fCelcius)	<i>op</i> <sub>71</sub>
		FahrenheitToCelcius(fFahrenheit)	<i>op</i> <sub>72</sub>
		Currency(fValue,sFrom,sTo)	<i>op</i> <sub>73</sub>
CurConvert	<i>ws</i> <sub>8</sub>	GetCurrencySign(CountryName)	<i>op</i> <sub>81</sub>
		ConvertCurrency(FromCountry,ToCountry,Amount)	<i>op</i> <sub>82</sub>
ConverterService	<i>ws</i> <sub>9</sub>	Convert(sourceCurrency,targetCurrency,value)	<i>op</i> <sub>91</sub>

lattice). From the lattices in Fig. 5, we can extract some backup services for our composite service. For example, we used operation *op*<sub>11</sub> from service *s*<sub>1</sub>, this operation has 3 equivalents: *op*<sub>21</sub>, *op*<sub>31</sub> and *op*<sub>51</sub> appearing clearly in the lattice. This means that if service *s*<sub>1</sub> breaks down, we can replace it by the services *s*<sub>2</sub>, *s*<sub>3</sub> or *s*<sub>5</sub>. Moreover, if we go down in the lattice, we get the set of services that cover the used operations and offer other operations, like service *s*<sub>5</sub> and service *s*<sub>3</sub>. In the same way, we can extract the backup services for the calculation service *s*<sub>1</sub> that we are using. According to the calculation services lattice, the service *s*<sub>1</sub> as a whole set of operations cannot be replaced by any service. But, regarding the multiplication functionality, *op*<sub>13</sub>(mul:13), it can be replaced by the operations *op*<sub>33</sub>, *op*<sub>43</sub>, *op*<sub>52</sub>, *op*<sub>63</sub>, which are offered by the services *s*<sub>3</sub>, *s*<sub>4</sub>, *s*<sub>5</sub>, *s*<sub>6</sub> respectively.

The service lattices present the table of services in an organized way, which reveals the relations between the services. We can verify the correctness of the lattice by verifying the data in the table. The chosen threshold value determines the necessity of human intervention, when it is small, the precision becomes lesser, then a service's substitutes may need few adaptations in order to replace a service. For example, while doing our experiments, we had an operation for rate exchange that takes an additional date parameter. For a specific threshold value, lesser than 1, we got this operation grouped with the other operations, which do not have a date parameter. We need to do a little adaptation, in order to use this operation, which can be putting a default value like today's date.

**Table 8.** The set of calculation services.

Services	Id	Operations	Id
Calc	<i>ws</i> <sub>1</sub>	add(a,b)	<i>op</i> <sub>11</sub>
		div(a,b)	<i>op</i> <sub>12</sub>
		mul(a,b)	<i>op</i> <sub>13</sub>
		pow(b,a)	<i>op</i> <sub>14</sub>
		sub(a,b)	<i>op</i> <sub>15</sub>
Service	<i>ws</i> <sub>2</sub>	add(a,b)	<i>op</i> <sub>21</sub>
		sqrt(a)	<i>op</i> <sub>22</sub>
		sub(a,b)	<i>op</i> <sub>23</sub>
MathService	<i>ws</i> <sub>3</sub>	Add(A,B)	<i>op</i> <sub>31</sub>
		Divide(A,B)	<i>op</i> <sub>32</sub>
		Multiply(A,B)	<i>op</i> <sub>33</sub>
		Subtract(A,B)	<i>op</i> <sub>34</sub>
CalculatorService	<i>ws</i> <sub>4</sub>	add(y,x)	<i>op</i> <sub>41</sub>
		divide(denominator,numerator)	<i>op</i> <sub>42</sub>
		multiply(y,x)	<i>op</i> <sub>43</sub>
		subtract(y,x)	<i>op</i> <sub>44</sub>
CalcService	<i>ws</i> <sub>5</sub>	Divide(A,B)	<i>op</i> <sub>51</sub>
		Multiply(A,B)	<i>op</i> <sub>52</sub>
		OperationAdd(A,B)	<i>op</i> <sub>53</sub>
		Subtract(A,B)	<i>op</i> <sub>54</sub>
Calculate	<i>ws</i> <sub>6</sub>	Add(dbl1,dbl2)	<i>op</i> <sub>61</sub>
		Divide(dbl1,dbl2)	<i>op</i> <sub>62</sub>
		Multiply(dbl1,dbl2)	<i>op</i> <sub>63</sub>
		Subtract(dbl1,dbl2)	<i>op</i> <sub>64</sub>

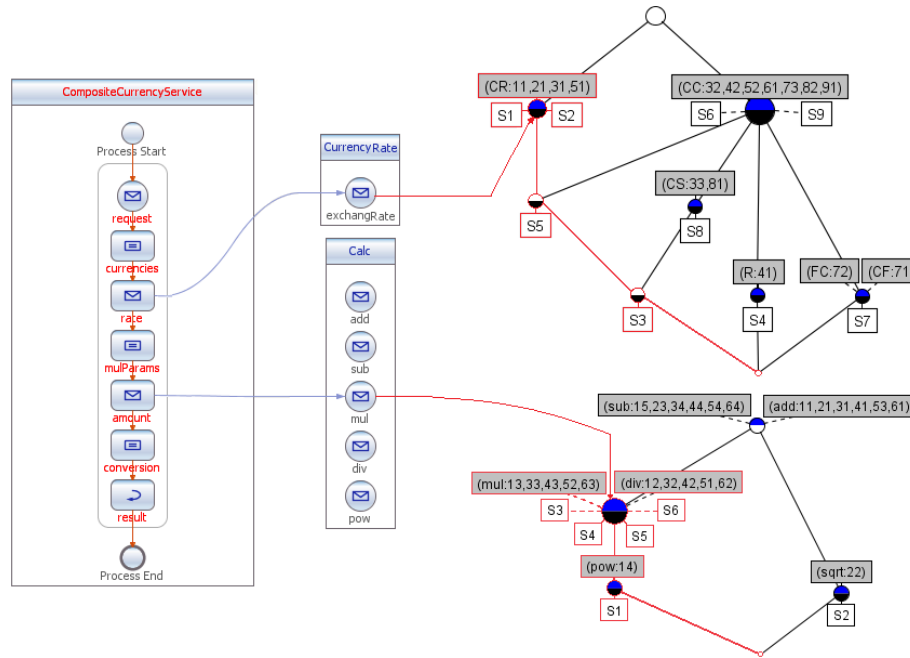
## 6 Related Work

Several works have been proposed for web service classification, in order to facilitate browsing, discovery and selection. A quick overview of some of the works can be obtained from [6, 15]. Below, we describe a selection of works, classified according to their adapted techniques.

### 6.1 Using formal concept analysis (FCA)

In [3], Aversano et al. classify web services using FCA as a means for WSDL browsing. Their formal contexts are composed according to three levels, service level, operation level and type level, together with keywords. These keywords are identified from the WSDL files by applying vector space metrics with the help of WordNet to discover the synonyms. The resulting service lattice represents an indexing of web services, it highlights the relationships between the services and permits the identification of different categorizations of a certain service.

In [7], Bruno et al. also use keywords extracted from services' interfaces together with FCA to build a web services lattice. They analyze the extracted words, process them using WordNet and other IR techniques. Then, they classify



**Fig. 5.** The composite currency service, supported by backups from the service lattices.

them into vectors using support vector machines (SVM). The obtained vectors categorize the services into domains, then service lattices can be obtained for each category using FCA.

In Peng et al. [5], similarity values are calculated for service operations, and depending on a chosen threshold, a service lattice is built.

## 6.2 Using machine learning

Many approaches adapt techniques from machine learning field, in order to discover and group similar services. In [8, 13], service classifiers are defined depending on sets of previously categorized services. Then the resulting classifiers are used to deduce the relevant categories for new given services. In case there were no predefined categories, unsupervised clustering is used. In [17], CPLSA approach is defined that reduces a services set then clusters it into semantically related groups.

## 6.3 Using service matching

In [16], a web service broker is designed relying on approximate signature matching using XML schema matching. It can recommend services to programmers in order to compose them. In [12], a service request and a service are represented

as two finite state machines then they are compared using various heuristics to find structural similarities between them. In [10], the Woogole web service search engine is presented, which takes the needed operation as input and searches for all the services that include an operation similar to the requested one. In [4], tags coming from folksonomies are used to discover and compose services.

#### 6.4 Using vector space model techniques

The vector space model is used for service retrieval in several existing works as in [20, 22, 9]. Terms are extracted from every WSDL file and the vectors are built for each service. A query vector is also built, and similarity is calculated between the service vectors and the query vector. This model is sometimes enhanced by using WordNet, structure matching algorithms to ameliorate the similarity scores as in [22], or by partitioning the space into subspaces to reduce the searching space as in [9].

#### 6.5 Discussion on the related work

In FCA approaches based on keywords, similar operations can't be determined and thus, web service substitutes can't be identified either. In our approach, we use the lattice based on keywords as a preliminary index together with operation similarity measuring, in order to generate our concept lattices. We have proposed several uses of the generated lattices, as for service discovery, selection and supporting service compositions with backup services. One of our main contributions is the idea of supporting the continuity of service compositions.

A service lattice is an organization of services that reveals the relations between them according to the operations provided in common. It offers a structure of navigation that enables better discovery and browsing than in structures such as lists and sets in other approaches (described in subsections 6.2, 6.3 and 6.4). New services can be classified in existing lattices by incremental algorithms for lattice generation. Thus, there is no need to regenerate the whole lattice, if a new service is to be added. A query represents a new concept in the lattice, and the services that offer the minimum required functionality represent the concepts that are closest to the query concept, while further situated services offer extra functionalities. In the lattice, when selecting a service, a sub-lattice that is descendant from this service can be extracted. This sub-lattice contains the possible backups that can replace this service to ensure a recovered functionality.

## 7 Conclusion

In this paper, we proposed an approach based on formal concept analysis (FCA) for building web service lattices according to functionality domains. We make use of similarity measures for web services to form our formal contexts in order to build the lattices according to threshold values.

A web service lattice reveals the invisible relations between web services in a certain domain, showing the services that are able to replace other ones. Thus, facilitating service browsing, selecting and identifying possible substitutions. We explained how to exploit the resulting lattices to build orchestrations of web services and supporting them with backup services.

The quality of our generated lattices depends on the chosen similarity measure and the similarity threshold. The more accurate the measure is, the more precise the obtained lattice is. The chosen values of threshold will give us a variation of lattices, and they reflect the level of the required adaptations. Thus, a high value of threshold means similar services with a low number of required adaptations.

Our work in progress is to enrich the service lattices with quality of service (QoS) aspects, in order to enable an automatic selection of a service that respond to a requested level of QoS. We are also working on the dynamic substitution of a web service by one of its backups, to ensure a continuous functionality of a service orchestration.

## References

1. G. Arévalo, N. Desnos, M. Huchard, C. Urtado, and S. Vauttier. Precalculating component interface compatibility using fca. In P. W. Eklund, J. Diatta, and M. Liquiere, editors, *CLA*, volume 331 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
2. G. Arévalo, N. Desnos, M. Huchard, C. Urtado, and S. Vauttier. Fca-based service classification to dynamically build efficient software component directories. *International Journal of General Systems*, May 2009. To appear.
3. L. Aversano, M. Bruno, G. Canfora, M. D. Penta, and D. Distanto. Using concept lattices to support service selection. *Int. J. Web Service Res.*, 3(4):32–51, 2006.
4. E. Bouillet, M. Febowitz, H. Feng, Z. Liu, A. Ranganathan, and A. Riabov. A folksonomy-based model of web services for discovery and automatic composition. In *IEEE International Conference on Services Computing (SCC)*, pages 389–396. IEEE Computer Society, 2008.
5. D. Peng, S. Huang, X. Wang, and A. Zhou, “Concept-based retrieval of alternate web services,” in *DASFAA*, ser. Lecture Notes in Computer Science, L. Zhou, B. C. Ooi, and X. Meng, Eds., vol. 3453. Springer, 2005, pp. 359–371.
6. S. Brockmans, M. Erdmann, and W. Schoch. Service-finder deliverable d4.1. research report about current state of the art of matchmaking algorithms. Technical report, October 2008.
7. M. Bruno, G. Canfora, M. D. Penta, and R. Scognamiglio. An approach to support web service classification and annotation. In *EEE*, pages 138–143. IEEE Computer Society, 2005.
8. M. Crasso, A. Zunino, and M. Campo. Awsc: An approach to web service classification based on machine learning techniques. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 12, No 37:25–36, 2008.
9. M. Crasso, A. Zunino, and M. Campo. Query by example for web services. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2376–2380, New York, NY, USA, 2008. ACM.

10. X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 372–383. VLDB Endowment, 2004.
11. B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, mathematical foundations edition, 1999.
12. A. Günay and P. Yolum. Structural and semantic similarity metrics for web service matchmaking. In *EC-Web*, pages 129–138, 2007.
13. A. Heß and N. Kushmerick. Learning to attach semantic metadata to web services. In *International Semantic Web Conference*, pages 258–273, 2003.
14. N. Kokash. A comparison of web service interface similarity measures. In *Proceeding of the 2006 conference on STAIRS 2006*, pages 220–231, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.
15. H. Lausen and N. Steinmetz. Survey of current means to discover web services. Technical report, Semantic Technology Institute (STI), August 2008.
16. J. Lu and Y. Yu. Web service search: Who, when, what, and how. In *WISE Workshops*, pages 284–295, 2007.
17. J. Ma, Y. Zhang, and J. He. Efficiently finding web services using a clustering semantic approach. In *Proceedings of CSSSIA '08*, pages 1–8, New York, NY, USA, 2008. ACM.
18. E. Newcomer and G. Lomow. *Understanding SOA with Web Services (Independent Technology Guides)*. Addison-Wesley Professional, 2004.
19. M. P. Papazoglou. *Web Services: Principles and Technology*. Pearson, Prentice Hall, 2008.
20. C. Platzer and S. Dustdar. A vector space search engine for web services. In *Third IEEE European Conference on Web Services, 2005. ECOWS 2005.*, pages 62–71, 2005.
21. E. Stroulia and Y. Wang. Structural and semantic matching for assessing web-service similarity. *Int. J. Cooperative Inf. Syst.*, 14(4):407–438, 2005.
22. Y. Wang and E. Stroulia. Semantic structure matching for assessing web service similarity. In *1st International Conference on Service Oriented Computing (IC-SOC03)*, pages 194–207. Springer-Verlag, 2003.