Backing Composite Web Services Using Formal Concept Analysis

Zeina Azmeh, Fady Hamoui, Marianne Huchard, Nizar Messai, Chouki Tibermacine, Christelle Urtado, Sylvain Vauttier

Backing Composite Web Services Using Formal Concept Analysis

Zeina Azmeh[1], Fady Hamoui[1,3], Marianne Huchard[1], Nizar Messai[2], Chouki Tibermacine[1], Christelle Urtado[3], and Sylvain Vauttier[3]

[1] LIRMM - CNRS & Univ. Montpellier II, France,
{azmeh, hamoui, huchard, tibermacin}@lirmm.fr
[2] MAS - Ecole Centrale Paris, France
nizar.messai@ecp.fr
[3] LGI2P - Ecole des Mines d'Alès - Nîmes, France
{Fady.Hamoui, 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 into coarser-grained applications. Achieving a required composite functionality requires the discovery of a collection of Web services out of the enormous service space. Each service must be examined to verify its provided functionality, making the selection task neither efficient nor practical. Moreover, when a service in a composition becomes unavailable, the whole composition may become functionally broken. Therefore, an equivalent service must be retrieved to replace the broken one, thus spending more time and effort. In this paper, we propose an approach for Web service classification based on FCA, using their operations estimated similarities. The generated lattices make the identification of candidate substitutes to a given service straightforward. Thus, service compositions can be achieved more easily and with backup services, so as to easily recover the functionality of a broken service.

Key words: Web service classification, Formal Concept Analysis (FCA), service composition, service backups.

1 Introduction

A Web service is a software functionality accessible through the network. It exposes its functionalities to the external world by an abstract interface expressed in Web Service Description Language (WSDL) [1]. A WSDL interface is an XML-based document that describes a service's available operations, parameters, data types, and access protocols.

Web services represent the building blocks for creating composite applications. When creating a composite application, each selected Web service must fulfill 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. Then, after identifying the needed services, they can be assembled together in order to meet the desired functionality of the whole composite application.

The task of finding an appropriate service to use is hard and time-consuming, because of the large number of existing Web services nowadays. This may become even harder knowing that Web services are not guaranteed to have a continuous execution. This is due to their dynamic nature, being offered by various providers, remotely accessed, and having different quality of service (QoS) levels. Therefore, an available functioning Web service may crash and become unavailable at any time, which requires finding an equivalent one to replace it, in order to maintain the application functionality.

The real challenge lies in the fact that there is a lack of WSDL management facilities, especially after the deficiency of UDDI [2], which was originally proposed as a core Web service registry standard: *"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"* [3]. Thus, a mechanism for organizing and indexing Web services is significantly required. This leads us to our proposition for Web service classification, which is based on Formal Concept Analysis (FCA)[4].

In our proposed approach, we consider the objects to be Web services and the attributes to be the operations offered by these services. We construct Web service lattices using many-valued contexts of similarity values calculated for each pair of operations. The generated service lattices provide us with browsing and navigation capabilities. This allows the retrieval of more general to more specific sets of services [5, 6]. More general sets have lesser common operations while more specific sets have more common operations. Therefore, applying FCA to Web services provides us with a retrieval mechanism, which facilitates both selection of Web services and identification of their possible substitutes. Accordingly, it helps building composite applications as well as supporting them with backup services.

The rest of the paper is organized as follows: Section 2 defines how we adapt FCA to web services. Section 3 explains our approach along with examples and formal definitions. Section 4 demonstrates a case study using real web services. Section 5 lists and discusses the related work. Finally, Section 6 concludes the paper and describes some of our perspectives.

2 An FCA-Based Approach for Web Service Classification

In our approach, we use FCA [4] in order to construct a classification of Web services. We consider that the objects are Web services and the attributes are operations. In this way, a formal context of Web services and operations becomes $\mathbb{K} = (\mathbb{W}, \mathbb{O}, I)$, where: $\mathbb{W} = \{ws_i \mid 1 \leq i \leq n_{\mathbb{W}} \ , \ n_{\mathbb{W}} > 1\}$ 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 all of the services forms the total set of operations:

$$ws_i = \{op_{ij} \mid 1 \leq i \leq n_{\mathbb{W}} \ , \ 1 \leq j \leq n_{ws_i}\}$$

$$\mathbb{O} = \bigcup_{i=1}^{i=n_{\mathbb{W}}} ws_i$$

$(ws, op) \in I$ 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 $\mathbb{W} = \{Calc_1, Calc_2, Calc_3\}$ and $\mathbb{O} = \{add, sub, mul, div, pow\}$.

	add	*sub*	*mul*	*div*	*pow*
$Calc_1$	×	×			×
$Calc_2$	×	×		×	×
$Calc_3$	×		×	×	

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

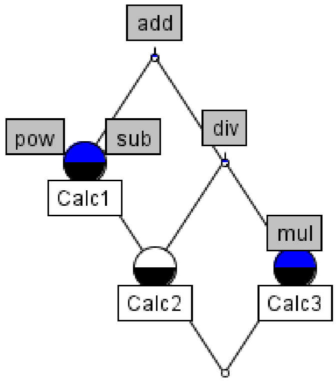


Fig. 1: The service lattice for the context in Table 1

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, $(\{Calc_1, Calc_2\})' = \{add, sub, pow\}$ and $(\{div\})' = \{Calc_2, Calc_3\}$.

A concept, for example $(\{Calc_1, Calc_2\}, \{add, sub, pow\})$, is thus a maximal collection of services offering similar operations. The concept lattice defines a hierarchical organization of services and operations, in which a certain concept inherits all the extents (services) of its descendants (subconcepts) and all the intents (operations) of its ascendants (super-concepts). Fig. 1 illustrates the lattice built for the context shown in Table 1, using the *ConExp* tool [7].

From the lattice in Fig. 1, we can reveal the relationships between the presented services. We list some of them as follows:

- $Calc_1$, $Calc_2$ and $Calc_3$ offer the operation $\{add\}$. Thus, they can replace each other for this operation;
- $Calc_1$ and $Calc_2$ offer the operations $\{add, sub, pow\}$;
- $Calc_2$ can replace $Calc_1$ since it offers all of its operations in addition to *div*;
- $Calc_2$ and $Calc_3$ offer together the operations $\{add, div\}$.

Using binary contexts to classify Web services by their operations reflects two cases: the service either offers a given operation or not. Substitution can only be handled when services offer strictly identical operations which is not the case for real Web services. This is why we need to introduce the notion of operation similarity and use many-valued contexts of similarity values, as in the following section.

3 Using Many-Valued Contexts

Web services in a certain business domain may offer similar operations. In order to classify these services by their operations using FCA, we need to calculate the operation similarity and to use many-valued contexts. We explain our approach using the set of services illustrated in Table 1. For clarity, we use only the first 3 operations of each service. We use the actual operations signatures. The set of services with their signatures are given unique identifiers, as listed in Table 2.

Table 2: A set of calculation services with their operations

Services	**Id**	**Operations**	**Id**
$Calc_1$	ws_1	add(a,b)	op_{11}
		sub(a,b)	op_{12}
$Calc_2$	ws_2	add(a,b,c)	op_{21}
$Calc_3$	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}

Next, a similarity measure must be chosen, and applied on pairs of operation signatures extracted from the WSDL files. There are several similarity measures for Web services that evaluate similarity according to syntax and semantics, such as [8–10]. 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, otherwise it will approach 0. The similarity measure is applied on pairs of operations provided by distinct services. We do not evaluate similarity between distinct operations provided by the same service (we suppose that it is equal to 0), because when a service becomes dysfunctional, all of its operations become dysfunctional too.

A similarity measure $Sim : \mathbb{O} \times \mathbb{O} \rightarrow [0,1]$ can be defined as follows:

$\forall\, op_{ij} \in \mathbb{O} \Longrightarrow Sim(op_{ij}, op_{ij}) = 1$ (an operation with itself)
$\forall\, op_{ij}, op_{ik} \in \mathbb{O}, j \neq k \Longrightarrow Sim(op_{ij}, op_{ik}) = 0$ (operations in the same service)
$\forall\, op_{ij}, op_{nm} \in \mathbb{O}, i \neq n \Longrightarrow Sim(op_{ij}, op_{nm}) \in [0,1]$ (operations in different services)

The calculated similarity values can be presented by a symmetric square matrix that we will call *SimMat*, as shown in Table 3. This matrix is of size $n = |\mathbb{O}|$, and its diagonal elements are all equal to 1 (similarity of an operation with itself).

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 θ 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*.

Table 3: The similarity matrix $(SimMat)$.

	op_{11}	op_{12}	op_{21}	op_{31}	op_{32}	op_{33}	op_{34}
op_{11}	1	0	0.75	0.5	0	0	1
op_{12}	0	1	0	0	0.75	0	0
op_{21}	0.75	0	1	0.75	0	0	0.75
op_{31}	0.5	0	0.75	1	0	0	0
op_{32}	0	0.75	0	0	1	0	0
op_{33}	0	0	0	0	0	1	0
op_{34}	1	0	0.75	0	0	0	1

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

	op_{11}	op_{12}	op_{21}	op_{31}	op_{32}	op_{33}	op_{34}
op_{11}	×		×				×
op_{12}		×			×		
op_{21}	×		×	×			×
op_{31}			×	×			
op_{32}		×			×		
op_{33}						×	
op_{34}	×		×				×

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

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

We use the *SimCxt* context to generate a lattice of operations (Fig. 2), $\underline{\mathfrak{B}}(\mathbb{O}, \mathbb{O}, RSim_\theta)$. This lattice helps in discovering groups of similar operations, which are used later on to construct the service 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 square concepts [11], 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.

	op_{11}	op_{34}	op_{21}	op_{31}	op_{12}	op_{32}	op_{33}
op_{11}	×	×	×				
op_{34}	×	×	×				
op_{21}	×	×	×	×			
op_{31}			×	×			
op_{12}					×	×	
op_{32}					×	×	
op_{33}							×

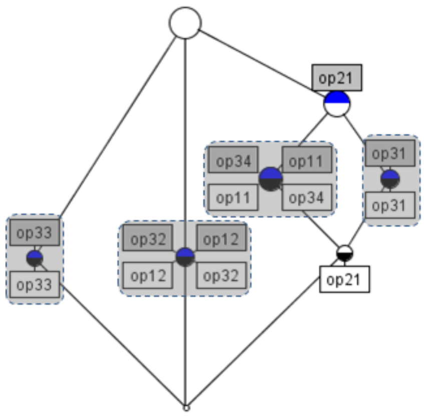


Fig. 2: The generated lattice for $(SimCxt)$ shown in Table 4.

From the lattice in Fig. 2 as from the interchanged context in Table 5, we can identify the groups of similar operations, and they are the following:

- $\{op_{11}, op_{34}, op_{21}\}$ that we label $(11,34,21)$;
- $\{op_{21}, op_{31}\}$ labelled $(21,31)$;
- $\{op_{12}, op_{32}\}$ labelled $(12,32)$;
- $\{op_{33}\}$ labelled (33).

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 use the labels representing the groups of operations to build the final context, which is shown in Table 6. Using this context, we generate the corresponding service lattice that is shown in Fig. 3.

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

Table 6: The final services × groups context.

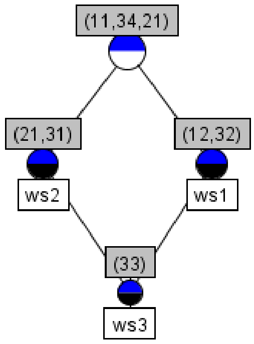


Fig. 3: The final service lattice with possible backups.

From the final generated service lattice, shown in Fig. 3, 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).

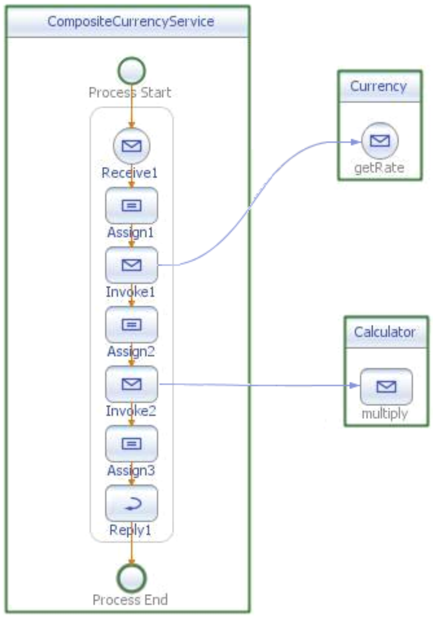
We can also infer immediately which services offer a specific functionality (denoted by a specific label), by considering the indices in the label. For example, the label $(11,34,21)$ makes it possible to directly deduce that (11) is provided by ws_1, (34) by ws_3 and (21) by ws_2.

4 Case Study

In this section, we demonstrate the use of service lattices for both building composite Web services and supporting them with backup services in a real world context.

We consider the example of a composite service for currency conversion, composed of two Web services: a currency converter service *Currency* and a calculation service *Calculator*. The *Currency* service offers an operation that returns the exchange rate between two entered currencies: *getRate(fromCurr,toCurr)*. The *Calculator* service offers an operation that calculates the multiplication of two entered numbers: *mul(a,b)*. We compose these two operations in order to build the composite currency service that converts a given amount from one currency to another. We describe a service composition using the Business Process Execution Language (BPEL) [12]. We use the BPEL editor of *NetBeans IDE* [13] to design and describe the specified *CompositeCurrencyService* as shown in Fig. 4.

We used the *Seekda* [14] and *Service-Finder* [15] Web service search engines to search for the needed services. We describe this case study on two parts: first we illustrate the use of the approach, then we validate it.



Recieve1: a start request performed by a requestor of the *CompositeCurrencyService*
Assign1: assigning values to the *getRate* operation's input parameters
Invoke1: invoking the *getRate* operation from the *Currency* service
Assign2: assigning the *getRate* operation's output value & an amount to the *mul* operation
Invoke2: invoking the *mul* operation from the *Calculator* service
Assign3: assigning the *mul* operation's output value to the *CompositeCurrencyService* response
Reply1: returning the final response of the *CompositeCurrencyService*

Fig. 4: The composite currency service.

4.1 Using the Approach

We use a set of services for currency conversion shown in Table 7 and another set for calculation 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.

Table 7: The set of currency converter services.

Services	Id	Operations	Id
CurrencyConverter	ws_1	GetConversionRate(fromCurrency,toCurrency)	op_{11}
CurrencyConvertor	ws_2	ConversionRate(FromCurrency,ToCurrency)	op_{21}
DOTSCurrencyExchange	ws_3	GetExchangeRate(ConvertFromCurrency,ConvertToCurrency)	op_{31}
		ConvertCurrency(Amount,ConvertFromCurrency,ConvertToCurrency)	op_{32}
		GetCountryCurrency(Country)	op_{33}
CurrencyRates	ws_4	GetRate(CurrencyCode)	op_{41}
		GetConversion(FromCurrencyCode,ToCurrencyCode)	op_{42}
RadixxFlights	ws_5	GetExchange(FromCurrency,ToCurrency)	op_{51}
		ConvertCurrency(Amount,FromCurrency,ToCurrency)	op_{52}
rates	ws_6	Convert(CurrencyFrom,CurrencyTo,ValueFrom)	op_{61}
Conversion	ws_7	CelciusToFahrenheit(fCelsius)	op_{71}
		FahrenheitToCelcius(fFahrenheit)	op_{72}
		Currency(fValue,sFrom,sTo)	op_{73}
CurConvert	ws_8	GetCurrencySign(CountryName)	op_{81}
		ConvertCurrency(FromCountry,ToCountry,Amount)	op_{82}
ConverterService	ws_9	Convert(sourceCurrency,targetCurrency,value)	op_{91}

Table 8: The set of calculation services.

Services	Id	Operations	Id
Calc	ws_1	add(a,b)	op_{11}
		div(a,b)	op_{12}
		mul(a,b)	op_{13}
		pow(b,a)	op_{14}
		sub(a,b)	op_{15}
Service	ws_2	add(a,b)	op_{21}
		sqrt(a)	op_{22}
		sub(a,b)	op_{23}
MathService	ws_3	Add(A,B)	op_{31}
		Divide(A,B)	op_{32}
		Multiply(A,B)	op_{33}
		Subtract(A,B)	op_{34}
CalculatorService	ws_4	add(y,x)	op_{41}
		divide(denominator,numerator)	op_{42}
		multiply(y,x)	op_{43}
		subtract(y,x)	op_{44}
CalcService	ws_5	Divide(A,B)	op_{51}
		Multiply(A,B)	op_{52}
		OperationAdd(A,B)	op_{53}
		Subtract(A,B)	op_{54}
Calculate	ws_6	Add(dbl1,dbl2)	op_{61}
		Divide(dbl1,dbl2)	op_{62}
		Multiply(dbl1,dbl2)	op_{63}
		Subtract(dbl1,dbl2)	op_{64}

For dealing with this illustration, we assess manually the similarity for the obtained services' operations of each set (an automatic approach is described later in the paper). This is achieved by comparing operation signatures (operation names, parameter names and types). Using the operations lattice and its square concepts, we identify the following groups of mutually similar operations for the currency services in Table 7:

- $\{op_{11}, op_{21}, op_{31}, op_{51}\}$ that we label $(CR : 11, 21, 31, 51)$;
- $\{op_{32}, op_{42}, op_{52}, op_{61}, op_{73}, op_{82}, op_{91}\}$ labelled $(CC : 32, 42, 52, 61, 73, 82, 91)$;
- $\{op_{33}, op_{81}\}$ labelled $(CS : 33, 81)$;
- $\{op_{41}\}$ labelled $(R : 41)$;
- $\{op_{72}\}$ labelled $(FC : 72)$;

– $\{op_{71}\}$ labelled $(CF:71)$.

We extract also the groups of mutually similar operations for the calculation services in Table 8, and they are as follows:

– $\{op_{15}, op_{23}, op_{34}, op_{44}, op_{54}, op_{64}\}$ labelled $(sub:15,23,34,44,54,64)$;
– $\{op_{11}, op_{21}, op_{31}, op_{41}, op_{53}, op_{61}\}$ labelled $(add:11,21,31,41,53,61)$;
– $\{op_{13}, op_{33}, op_{43}, op_{52}, op_{63}\}$ labelled $(mul:13,33,43,52,63)$;
– $\{op_{12}, op_{32}, op_{42}, op_{51}, op_{62}\}$ labelled $(div:12,32,42,51,62)$;
– $\{op_{14}\}$ labelled $(pow:14)$;
– $\{op_{22}\}$ labelled $(sqrt:22)$.

These extracted groups of similar operations lead to a binary context for each set of services as shown in Tables 9 and 10.

Table 9: The formal context corresponding to the currency converter services.

	(CR:11,21,31,51)	(CC:32,42,52,61,73,82,91)	(CS:33,81)	(R:41)	(FC:72)	(CF:71)
ws_1	×					
ws_2	×	×				
ws_3	×	×	×			
ws_4		×		×		
ws_5	×	×				
ws_6		×				
ws_7		×			×	×
ws_8		×	×			
ws_9		×				

Table 10: The formal context corresponding to the calculator services.

	(sub:15,23,34,44,54,64)	(add:11,21,31,41,53,61)	(mul:13,33,43,52,63)	(div:12,32,42,51,62)	(pow:14)	(sqrt:22)
ws_1	×	×	×	×	×	
ws_2	×	×				×
ws_3	×	×	×	×		
ws_4	×	×	×	×		
ws_5	×	×	×	×		
ws_6	×	×	×	×		

We generate the two corresponding lattices as shown in the right side of Fig. 5. We can exploit these service lattices to build our composite service as well as to support it with backup services. Thus, we decide to select operation $op_{11}:(CR:11)$ from service ws_1 for exchange rate (currency lattice), and operation $op_{13}:(mul:13)$ from service ws_1 for multiplication (calculation lattice). From these lattices (Fig. 5), we can also

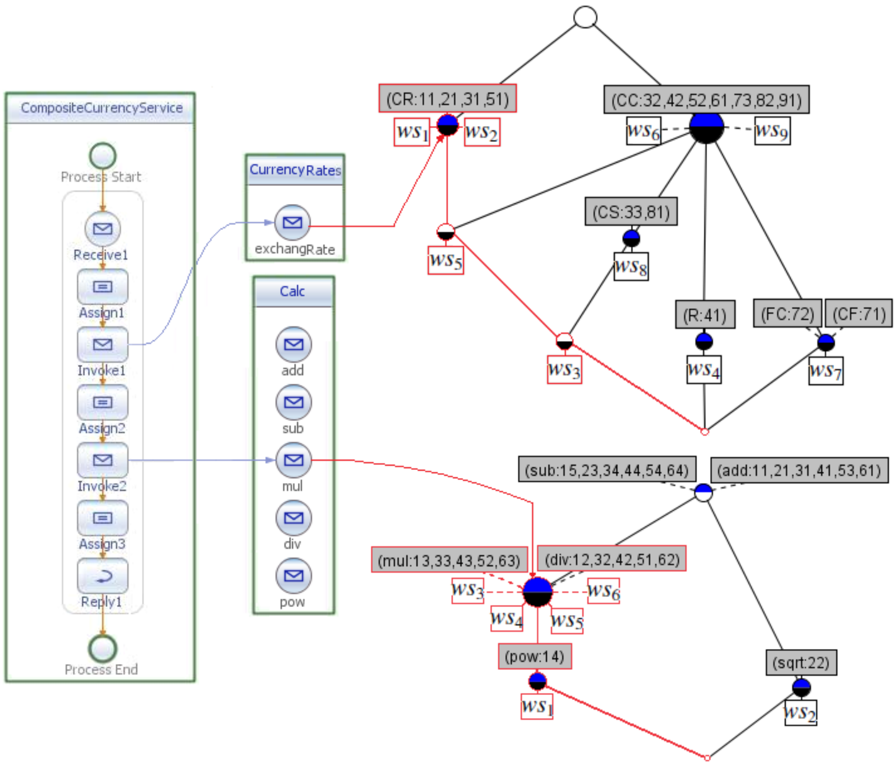


Fig. 5: The composite currency service, supported by backups from the service lattices.

extract some backup services for our composite service according to the selected operations. For example, we used operation $op_{11} : (CR : 11)$ from service ws_1, which has 3 equivalent operations: $op_{21} : (CR : 21)$, $op_{31} : (CR : 31)$ and $op_{51} : (CR : 51)$ appearing clearly in the lattice. This means that if service ws_1 breaks down, we can replace it by any of the services ws_2 (equivalent to ws_1 being in the same concept), ws_3 or ws_5 (services introduced in subconcepts).

Moreover, if we go down in the lattice, we get the set of services that provide the operations used together with extra operations, like service ws_5 and service ws_3. They can help if the composite service evolves and needs other operations. In the same way, we can extract the backup services for the calculation service ws_1 that we are using. According to the calculation service lattice, service ws_1 as a whole set of operations cannot be replaced by any service. But, regarding the multiplication functionality, $op_{13}(mul : 13)$, it can be replaced by operations $op_{33} : (mul : 33)$, $op_{43} : (mul : 43)$, $op_{52} : (mul : 52)$, and $op_{63} : (mul : 63)$, which are offered by services ws_3, ws_4, ws_5, and ws_6 respectively. This gives us a replacement possibility in case of unavailability of ws_1 in the framework of the composite currency service.

4.2 Validation

In this section, we validate our approach using the entire number of retrieved $Calculator$ and *Currency* services[4]. We queried $Service-Finder$ to collect service endpoints (addresses), then we downloaded the corresponding WSDL interfaces via $Seekda$. For the $Calculator$ service, we searched using $multiply$ as keyword. This returned a set $WS1$ of 29 services, among which we found one unrelated service.

For the *Currency* service, we used a combination of the following keywords exchange, rate, currency, converter. After eliminating the repeated services, we found a set of 81 services. From this set, we also eliminated the services that we were unable to parse. This resulted in a set $WS2$ of 64 services.

We parsed each service of the two sets (WSDL parser[5]), to extract its operation signatures. The set $WS1$ has a total of 142 operations, while $WS2$ has 935 operations.

In order to calculate the *SimMat* (explained in Section 3) for both sets of services, we make use of $Jaro-Winkler$ [16] similarity measure, to assess the similarity between the extracted signatures according to each set. This metric gave convenient similarity values that were calculated efficiently, compared to another tested technique that used a combination of syntactic and semantic metrics. After a number of experiments, we found that a relatively pertinent similarity value starts from 80%. By applying this threshold on the *SimMat*, we obtained the binary *SimCxt* corresponding to each set.

We tried to compute the lattices corresponding to each *SimCxt* using Galicia [17]. The lattices could not be generated due an "out of memory" error (on a machine with limited resources). Therefore, we computed the Galois Sub Hierarchy (GSH), (order induced by attribute and object concepts). Using GSH, we obtained a suborder of 155 concepts for *SimCxt* (142×142) and another suborder of 1724 concepts for *SimCxt* (935×935). The second suborder may be reduced depending on the functionality filtering techniques.

Hereby, we restrict our analysis on $WS1$ regarding the limited paper space. From the GSH calculated for *SimCxt* (142×142), we extracted 65 square concepts. Among these 65 square concepts, we had 13 non-trivial concepts and 52 concepts reduced to one operation. Each square concept represents a functionality, for example: $c82$ represents the $multiply$ functionality. It contains $\{op15.2, op18.3, op2.3, op6.3, op8.2\}$, which are mutually substitutable operations for calculating the multiplication of two numbers.

Afterwards, we constructed the lattice of services (as objects) and these square concepts (as attributes). The generated lattice is shown in Fig. 6, and contains 21 concepts. By regarding the right half of the lattice, we can notice services that can be entirely replaced by other ones. For example: if we consider $ws15$, it contains the $multiply$ functionality (being a subconcept of $c82$). This service can be replaced by three other services: $ws18$, $ws6$ and $ws2$.

5 Related Work

Software engineering research has long benefitted from using FCA-based techniques in various ways, as attested by [18] which indexes and classifies 42 such scientific papers

[4] Retrieved services: http://www.lirmm.fr/∼azmeh/icfca11/CaseStudy.html
[5] Available online: http://www.lirmm.fr/∼azmeh/tools/WsdlParser.html

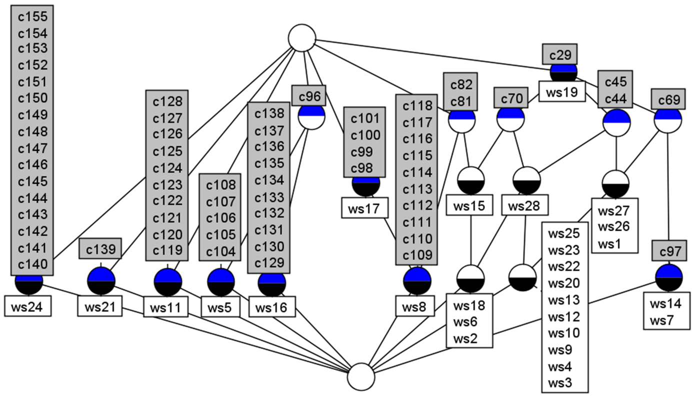


Fig. 6: The lattice corresponding to the *Calculator* services set.

published between 1992 and 2003 using FCA. These works go from early development phases (requirement engineering) to late ones (maintenance or legacy system analysis). Many have focused on refactoring and reingineering, especially in object-oriented languages [19–22]. Although our approach can be used and understood as a Web service refactoring method (since operations are factorized in the lattice), this paper chooses to focus on the classification of Web services inside backup service libraries. Among the works that ambition to browse or request software libraries using FCA, some rely mainly on syntax [23, 24], extending type theory [25] to recent paradigms (Component-based development or SOA). Others have studied the use of FCA [26] to structure keyword-based indexes that enable to browse software libraries [27, 28]. In the literature, we can find several works that more specifically focus on Web service classification and selection. A quick overview can be obtained from [29, 30]. In sections 5.1 and 5.2 we list a selection of works based respectively on FCA and on other techniques. Then we discuss comparatively our contribution in Section 5.3.

5.1 Approaches Based on FCA

In [31], Web services are classified using FCA to facilitate WSDL browsing. The 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 [32], FCA is used together with keywords extracted from services' interfaces to build a Web services lattice. The extracted words are processed using WordNet and other IR techniques, and are classified into vectors using support vector machines (SVM). The obtained vectors categorize the services into domains, and service lattices are obtained for each category using FCA.

In [33, 34], pairs of similar operations, depending on a chosen threshold, are merged together and the services are described by a representative operation of the pair in order to build the service lattice. They do not approach the issue that in a set of operations, op_1 can be similar to op_2 and op_2 similar to op_3 but op_1 might be not similar to op_3 because similarity is in general not a transitive operation. We solve this issue using an intermediate operation lattice based on the *SimCxt* to merge the maximal sets of mutually similar operations. Our mining of mutually similar operations is another application of the use of tolerance relations jointly with FCA, as is also done in [35].

5.2 Approaches Based on Other Techniques

Many approaches use machine learning techniques, in order to discover and group similar services. In [36, 37], service classifiers are defined depending on sets of previously categorized services. The resulting classifiers are then used to deduce relevant categories for new services. In case there are no predefined categories, unsupervised clustering is used. In [38], the CPLSA approach is defined that reduces a service set then clusters it into semantically related groups.

In [39], 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 [40], 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 [8], the Woogle 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 [41], tags coming from folksonomies are used to discover and compose services.

The vector space model is used for service retrieval in several existing works as in [42–44]. Terms are extracted from every WSDL file and vectors are built for describing 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 similarity scores as in [43], or by partitioning the search space into smaller subspaces as in [44].

5.3 Discussion

In FCA approaches based on keywords, similar operations can not be determined and thus, Web service substitutes can not be identified either. In our approach, we generate an intermediary lattice to group mutually similar operations. Thus, sets of equivalent operations appear in each concept of the final lattice. This serves for several purposes such as service retrieval, selection and support for service compositions with backup services. Indeed, one of our main contributions is the idea of supporting the continuity of service compositions. 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.

A service lattice is a structure that reveals relations between services according to the operations provided in common. It offers a navigation facility that enables better discovery and browsing than in structures such as lists and sets used in the other approaches. New services can be classified in existing lattices using incremental lattice generation algorithms. Thus, there is no need to regenerate the whole lattice.

Moreover, our approach can be tuned to have similarity thresholds set to consider finer-grained to coarser-grained comparisons. Indeed, the threshold values set during the process set the sizes of the sieves used to keep similar operations. These thresholds are set empirically. They condition the number of candidate backups our approach will discover. If there are too much candidate services, selection might as well be harder (only very similar services should be kept): the threshold can be raised. If there are few services proposed as backups, the threshold can be lowered. Backup candidates will be more dissimilar, probably requiring some little manual adaptations, but such setting would find backup possibilities where others would not. Finally, systematically calculating lattices for several threshold values might provide an interesting zoom-in / zoom-out capability that would provide several finer to coarser-grained classifications as in [45].

6 Conclusion and Future Work

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 [8–10] 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 responds 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. Besides, the construction of a composite web service could benefit from Relational Concept Analysis [46]. Several context families could be considered that would encode relations between operations, operations and services, or services in the composition.

Another challenge is the dynamic update of the classification. As algorithms exist that incrementally build lattices, we believe adding services might be possible without reconsidering the whole calculus. When the disappearance of services is concerned, dismissing the indexing information immediately might not be a good idea as services might be frequently unavailable for temporary periods of time (as a crashed web server reboots, for instance). More observation still is necessary for us to evaluate if disappearances should be handled as immediate removals (or, maybe, as lazy removals, based on their being unavailable for too long). This dynamic aspect is an interesting field for future research.

Acknowldegements Authors would like to thank anonymous reviewers for their relevant comments that helped to clarify and enrich our paper.

References

1. Web Services Description Language (WSDL) 1.1. (http://www.w3.org/TR/wsdl)
2. UDDI Version 3.0.2. (http://www.uddi.org/pubs/uddi_v3.htm)
3. Newcomer, E., Lomow, G.: Understanding SOA with Web Services (Independent Technology Guides). Addison-Wesley Professional (2004)
4. Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical foundations edn. Springer (1999)
5. Godin, R., Mineau, G.W., Missaoui, R.: Méthodes de classification conceptuelle basées sur les treillis de Galois et applications. Revue d'intelligence artificielle **9** (1995) 105–137
6. Carpineto, C., Romano, G.: A lattice conceptual clustering system and its application to browsing retrieval. Machine Learning **24** (1996) 95–122
7. The Concept Explorer. (http://conexp.sourceforge.net/)
8. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In Proc. of VLDB '04, VLDB Endowment (2004) 372–383
9. Stroulia, E., Wang, Y.: Structural and semantic matching for assessing web-service similarity. Int. J. Cooperative Inf. Syst. **14** (2005) 407–438
10. Kokash, N.: A comparison of web service interface similarity measures. In Proc. of STAIRS 2006, Amsterdam, The Netherlands, IOS Press (2006) 220–231
11. Azmeh, Z., Huchard, M., Messai, N., Tibermacine, C., Urtado, C., Vauttier, S.: Many-Valued Concept Lattices for Backing Composite Web Services. Technical report, LIRMM (2010)
12. Web Services Business Process Execution Language Version 2.0. (http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html)
13. NetBeans IDE. (http://www.netbeans.org/)
14. Seekda Web Services Search Engine. (http://webservices.seekda.com)
15. Service-Finder Web Services Search Engine. (http://demo.service-finder.eu)
16. Cohen, W.W., Ravikumar, P.D., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In Kambhampati, S., Knoblock, C.A., eds.: IIWeb. (2003) 73–78
17. Galicia. (http://www.iro.umontreal.ca/ galicia/)
18. Tilley, T., Cole, R., Becker, P., Eklund, P.: A survey of formal concept analysis support for software engineering activities. In Ganter, B., Stumme, G., Wille, R., eds.: Formal Concept Analysis: Foundations and Applications. LNCS–LNAI (3626). Springer (2005) 250–271
19. Godin, R., Mili, H.: Building and maintaining analysis-level class hierarchies using galois lattices. In: OOPSLA. (1993) 394–410
20. Snelting, G., Tip, F.: Understanding class hierarchies using concept analysis. ACM Trans. Program. Lang. Syst. **22** (2000) 540–582
21. Huchard, M., Dicky, H., Leblanc, H.: Galois lattice as a framework to specify building class hierarchies algorithms. ITA **34** (2000) 521–548
22. Godin, R., Valtchev, P.: Formal concept analysis-based class hierarchy design in object-oriented software development. In Ganter, B., Stumme, G., Wille, R., eds.: Formal Concept Analysis. Volume 3626 of Lecture Notes in Computer Science., Springer (2005) 304–323
23. Aboud, N.A., Arévalo, G., Falleri, J.R., Huchard, M., Tibermacine, C., Urtado, C., Vauttier, S.: Automated architectural component classification using concept lattices. In Proc. of WICSA/ECSA'09, Cambridge, UK, IEEE Computer Society Press (2009)
24. Arévalo, G., Desnos, N., Huchard, M., Urtado, C., Vauttier, S.: FCA-based service classification to dynamically build efficient software component directories. International Journal of General Systems **38** (2009) 427–453

25. Liskov, B.: Keynote address - data abstraction and hierarchy. SIGPLAN Not. **23** (1987) 17–34
26. Lindig, C.: Concept-based component retrieval. In Köhler, J., Giunchiglia, F., Green, C., Walther, C., eds.: Working Notes of the IJCAI'95 Workshop: Formal Approaches to the Reuse of Plans, Proofs, and Programs, Montréal, Canada (1995) 21–25
27. Fischer, B.: Specification-based browsing of software component libraries. In Proc. of ASE'98, Honolulu, USA (1998) 74–83
28. Sigonneau, B., Ridoux, O.: Indexation multiple et automatisée de composants logiciels. Technique et Science Informatiques **25** (2006) 9–42
29. Brockmans, S., Erdmann, M., Schoch, W.: Service-finder deliverable d4.1. research report about current state of the art of matchmaking algorithms. Technical report (2008)
30. Lausen, H., Steinmetz, N.: Survey of current means to discover web services. Technical report, Semantic Technology Institute (STI) (2008)
31. Aversano, L., Bruno, M., Canfora, G., Penta, M.D., Distante, D.: Using concept lattices to support service selection. Int. J. Web Service Res. **3** (2006) 32–51
32. Bruno, M., Canfora, G., Penta, M.D., Scognamiglio, R.: An approach to support web service classification and annotation. In: EEE, IEEE Computer Society (2005) 138–143
33. Peng, D., Huang, S., Wang, X., Zhou, A.: Concept-based retrieval of alternate web services. In Zhou, L., Ooi, B.C., Meng, X., eds.: DASFAA. Volume 3453 of Lecture Notes in Computer Science., Springer (2005) 359–371
34. Azmeh, Z., Huchard, M., Tibermacine, C., Urtado, C., SylvainVauttier: Using concept lattices to support web service compositions with backup services. In: Proc. of ICIW'10, IEEE Computer Society (2010) 363–368
35. Kaytoue, M., Assaghir, Z., Napoli, A., Kuznetsov, S.O.: Embedding tolerance relations in formal concept analysis: an application in information fusion. In Huang, J., Koudas, N., Jones, G., Wu, X., Collins-Thompson, K., An, A., eds.: CIKM, ACM (2010) 1689–1692
36. Crasso, M., Zunino, A., Campo, M.: Awsc: An approach to web service classification based on machine learning techniques. Inteligencia Artificial, Revista Iberoamericana de Interligencia Artificial **12, No 37** (2008) 25–36
37. Heß, A., Kushmerick, N.: Learning to attach semantic metadata to web services. In: International Semantic Web Conference. (2003) 258–273
38. Ma, J., Zhang, Y., He, J.: Efficiently finding web services using a clustering semantic approach. In Proc. of CSSSIA '08, New York, USA, ACM (2008) 1–8
39. Lu, J., Yu, Y.: Web service search: Who, when, what, and how. In: WISE Workshops. (2007) 284–295
40. Günay, A., Yolum, P.: Structural and semantic similarity metrics for web service matchmaking. In: EC-Web. (2007) 129–138
41. Bouillet, E., Feblowitz, M., Feng, H., Liu, Z., Ranganathan, A., Riabov, A.: A folksonomy-based model of web services for discovery and automatic composition. In: IEEE International Conference on Services Computing (SCC), IEEE Computer Society (2008) 389–396
42. Platzer, C., Dustdar, S.: A vector space search engine for web services. In: Third IEEE European Conference on Web Services, 2005. ECOWS 2005. (2005) 62–71
43. Wang, Y., Stroulia, E.: Semantic structure matching for assessing web service similarity. In Proc. of ICSOC'03, Springer-Verlag (2003) 194–207
44. Crasso, M., Zunino, A., Campo, M.: Query by example for web services. In Proc. of SAC '08, New York, NY, USA, ACM (2008) 2376–2380
45. Messai, N., Devignes, M.D., Napoli, A., Smaïl-Tabbone, M.: Using domain knowledge to guide lattice-based complex data exploration. In Proc. of ECAI 2010, IOS Press, 847–852
46. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Ann. Math. Artif. Intell. **49** (2007) 39–76