Using Concept Lattices to Support Web Service Compositions with Backup Services

Zeina Azmeh, Marianne Huchard, Chouki Tibermacine
LIRMM - CNRS & Univ. Montpellier II - France
{*azmeh, huchard, tibermacin*}*@lirmm.fr*

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

Abstract—In SOA, composite applications can be developed on the basis of collections of interacting web services. A service's functionality is exposed to the external world by an abstract interface, described by the standard WSDL language, which must be published by service providers to public registries where service consumers can find them. Nowadays, web service discovery has become a real problem, because of the lack of public registries to publish and organize the fairly huge number of existing services. In this paper, we propose an approach based on formal concept analysis (FCA) for classifying and browsing web services. Using this approach, the web services are organized into a lattice structure, to facilitate their browse and selection. A service lattice reveals the invisible relations between the services, enabling the discovery of a needed service as well as the identification of its possible alternatives. Thus, service discovery may be achieved more easily using the service lattice. This facilitates the construction of service compositions and supports them with backup services to ensure a continuous functionality.

Keywords-Web service discovery; formal concept analysis; service backups.

I. Introduction

Web services represent an important realization of service-oriented architectures (SOA). In SOA, composite applications can be developed on the basis of collections of interacting services offering well-defined interfaces [1]. When a service is accessed and used through the internet and by open internet-based standards, it results in the concept of web services. A web service exposes its functionality to the external world by an abstract interface expressed by WSDL [23] (Web Service Description Language). 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 in 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 web service must fulfil a part of the application's functionality. Therefore, each service's WSDL interface 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 classification and browsing, in order to facilitate the discovery of a web service, the identification and conservation of its candidate backups. This facilitates the construction of service compositions as well as ensuring their continuous functionality, by supporting them with backup services. We use the formal concept analysis formalism to organize web services into lattices according to their functionality domain and depending on the similarity estimated between their operations. The constructed web service lattices group the services into sets of services with similar operations and order these sets according to the number of similar operations they have and according to how similar these operations are. In this way the generated service lattices offer 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 II gives the problem statement with an example of composing web services. Section III gives an overview about formal concept analysis with the basic definitions. In Section IV, we present our approach based on FCA to classify web services. In Section V, we give a case study to explain the use of our approach. Section VI lists and discusses the existing approaches concerning web service classification and discovery. Finally, Section VII concludes the paper and gives some perspectives.

II. 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 [24] 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" [3].

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

Let's consider the following travel scenario: a traveller needs to reserve a plane ticket to a desired city. Supposing that this traveller lives in a small city that has no airport, then, he should also travel to the city where the airport is located, in order to take the plane he reserved. Thus, he must also reserve a train ticket, from his home city to the airport city, taking into consideration the flight exact time with the time needed to travel between the two cities. This scenario can be achieved by a travel composite service (TCS), in which 3 functionalities must be satisfied:

- reserve a flight from home city, or airport city, towards a desired city,
- if a train reservation is needed (in case that the home city is not the airport city), then calculate the needed time (duration) to travel between the two cities by train,
- reserve a train ticket, regarding the exact flight time and the calculated duration.

The TCS can be realized by discovering services offering the described functionalities and composing them. It may look like the orchestration in Figure 1. In this orchestration, if the service TrainWS crashes, for example, an equivalent service offering at least the two used operations calcDuration() and resTrain() must be searched and discovered, in order to recover the missing functionality, and ensure the continuity of the composition.

In the following sections, we describe our FCA-based approach that facilitates the construction of such composition and illustrate how to support it with backup services.

III. 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 [6]. 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).
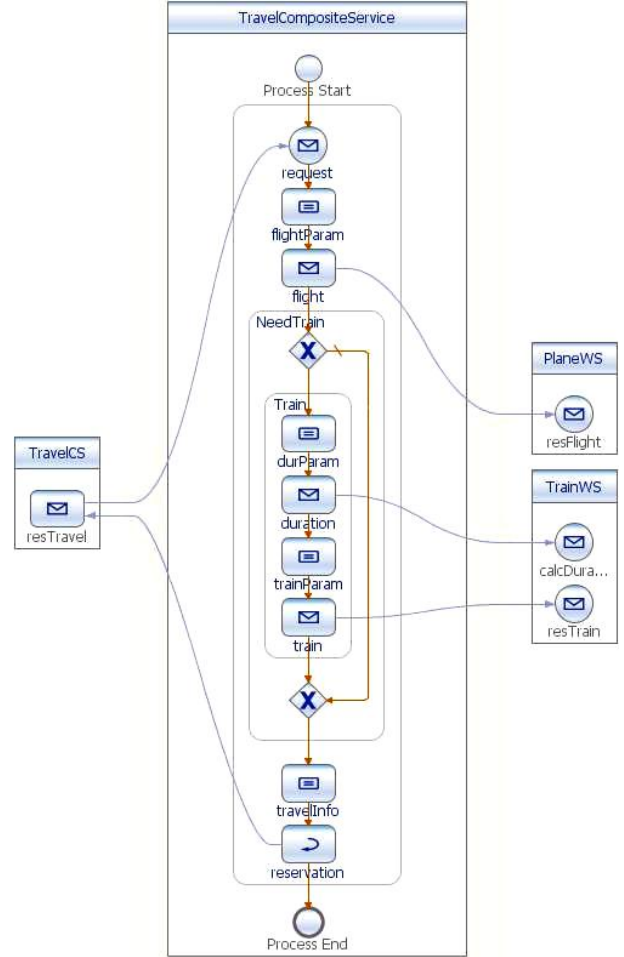

Figure 1.   The travel composite service (TCS).

Table I shows an example of a formal context (G, M, I) where $G = \{o_1, o_2, o_3\}$ and $M = \{a, b, c, d, e\}$.

Having a set $A \subseteq G$,

$$A' = \{m \in M \mid \forall\, g \in A : (g, m) \in I\}$$

is the set of common attributes. In the same way, having the set of attributes $B \subseteq M$,

$$B' = \{g \in G \mid \forall\, m \in B : (g, m) \in I\}$$

is the set of common objects.
In our example, $(\{o_1, o_3\})' = \{a, b, d\}$ and $(\{c\})' = \{o_2, o_3\}$.

	a	b	c	d	e
o_1	×	×		×	
o_2		×	×		×
o_3	×	×	×	×	

Table I
A FORMAL CONTEXT FOR $G \times M$

A concept is a pair of sets (A, B) where $A \subseteq G$ is called the extent, $B \subseteq M$ is called the intent, and $B = (A)'$,

$A = (B)'$. Meaning that, a concept is a maximal collection of objects sharing similar attributes. The set of all concepts is denoted by $\mathfrak{B}(G, M, I)$.

In our example, $(\{o_1, o_3\}, \{a, b, d\})$ is a concept while $(\{c\}, \{o_2, o_3\})$ is not. In fact, while $(\{c\})' = \{o_2, o_3\}$, $(\{o_2, o_3\})' = \{b, c\}$. A concept $(A1, B1)$ is a sub-concept of a concept $(A2, B2)$ if $A1 \subseteq A2$, which imposes a partial order relation on $\mathfrak{B}(G, M, I)$ expressed as $(A1, B1) \leq (A2, B2)$. The partial order relation $\leq$ can be used to build a structure, which is called a concept lattice and is denoted by $\underline{\mathfrak{B}}(G, M, I)$. A concept lattice defines a hierarchical representation of objects and attributes, in which a certain concept inherits all the extents (objects) of its descendants and all the intents (attributes) of its ascendants. Figure 2 illustrates the lattice built for the context[1] shown in table I, we notice that upper labels are the reduced intent sets (attributes) and bottom labels are the reduced extent sets (objects).
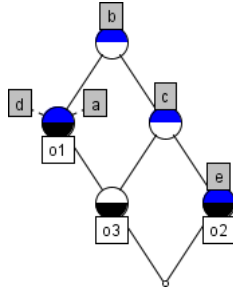

Figure 2. The concept lattice for the context in table I.

In this lattice we can reveal the relationships between the present objects, some of them are the following:

- o_1, o_2 and o_3 share the attribute $\{b\}$, thus, they can replace each other for this attribute,
- o_1 and o_3 share the attributes $\{a, b, d\}$,
- o_3 can replace o_1 since it shares all of its attributes in addition to c,
- o_2 and o_3 share together the attributes $\{b, c\}$,
- o_2 can replace o_1 for the attribute b.

IV. THE APPROACH

Figure 3 gives a general overview of the main steps of our approach, during which two lattices are generated consecutively:

- a lattice for indexing the services by keywords extracted from their WSDLs,
- a lattice for service classification, which organizes the services according to the similar operations they provide.

As a start, WSDL files coming from various repositories are analysed and automatically tagged with the significant
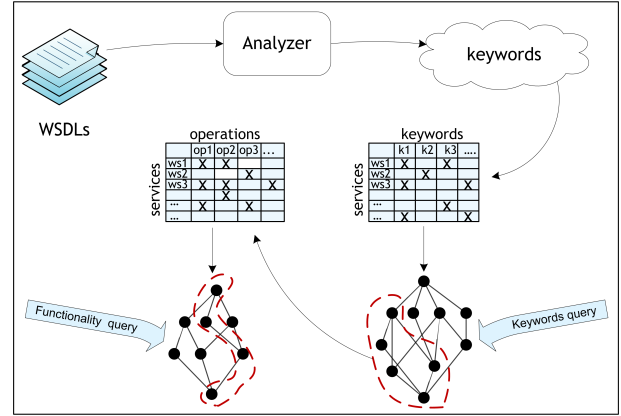

Figure 3. The main steps of our approach.

keywords [7]. The extracted keywords are used to generate the first lattice, which represents an indexation of services by keywords. Supposing a set of random web services for travel facilities, the corresponding lattice may resemble the lattice in Figure 4. In lattices, some intermediate concepts (white nodes) can be removed without losing information, which makes the structure more scalable [6].
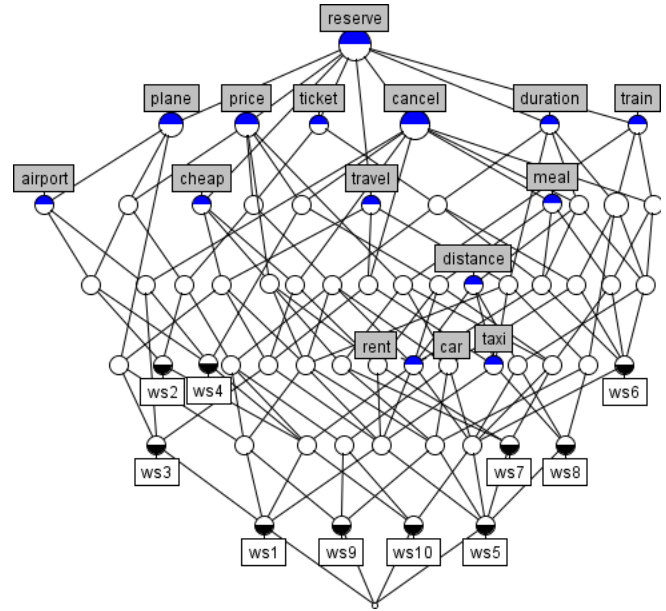

Figure 4. The concept lattice for a set of services and their keywords.

From this lattice, we can retrieve services in a certain domain by enquiring the lattice using keywords. Such a query returns a sub-lattice of services sharing the specified set of keywords, which are highly probable to provide similar functionalities. We extract these services, and apply a similarity measure on their operations, pair by pair. We will not consider the similarity between operations provided by the same service, because when a service becomes

dysfunctional, all of its operations become dysfunctional too.

Several similarity measures for web services exist in the literature. They evaluate the similarity according to the syntactic and semantic levels, such like [18], [4], [5].

Measuring the similarity between the operations enables us of identifying groups of similar operations, in order to build a context of $service \times operation$. This results in a new lattice that reveals the relations between the services according to the provided functionality. Searching for a certain operation returns a sub-lattice of services providing the operation and showing the replaceable services. We better explain the presented approach using a case study described in the next section.

V. CASE STUDY

In this section, we return to the scenario of TCS described in Section II, and we explain how to build such a composition and support it with backups using the generated lattices. Figure 4 illustrates the lattice of services and their keywords as explained before. We want to use this lattice to discover the required services in order to build the TCS. We begin by specifying the set of keywords describing the needed composite service. They are of two categories, plane reservation and train reservation. Thus, we form two queries, we perform a query on a lattice by expressing the set of keywords as a new line, added to the context $service \times keyword$. This results in a new concept in the lattice, labelled $query$. The services that answer the specified query are represented by the sub-concepts of the concept $query$. In our scenario, we perform two keyword queries: $query1 = \{reserve, plane\}$ and $query2 = \{reserve, train, duration\}$. This results in the lattice shown in Figure 5, in which, we extract the services answering the two queries and they are:

- for $query1$: $ws_1, ws_2, ws_3, ws_4, ws_9$ and ws_{10},
- for $query2$: $ws_5, ws_6, ws_8, ws_9, ws_{10}$.

We further analyse the two sets of services corresponding to the two queries. We apply a similarity measure on the operations according to each set and we identify the groups of similar operations. For example, an operation labelled $reserveF$ in the lattice, represents a group of similar operations for performing a flight reservation. This way, we can build a new context of $service \times operation$ for each set of services. The lattice that corresponds to $query1$ is shown in Figure 6 (top), and the one corresponding to $query2$ in Figure 6 (bottom).

Using these two lattices, the selection of services offering required operations is straightforward. In our scenario, we need three functionalities as indicated before in Section II: flight reservation, train reservation (if needed), and calculating the duration needed to travel by train to the aimed destination. By regarding the lattice in Figure 6 (top), we notice that all of the services offer an operation for plane reservation. In this case, a service selection might be done regarding the extra operations that the services provide,
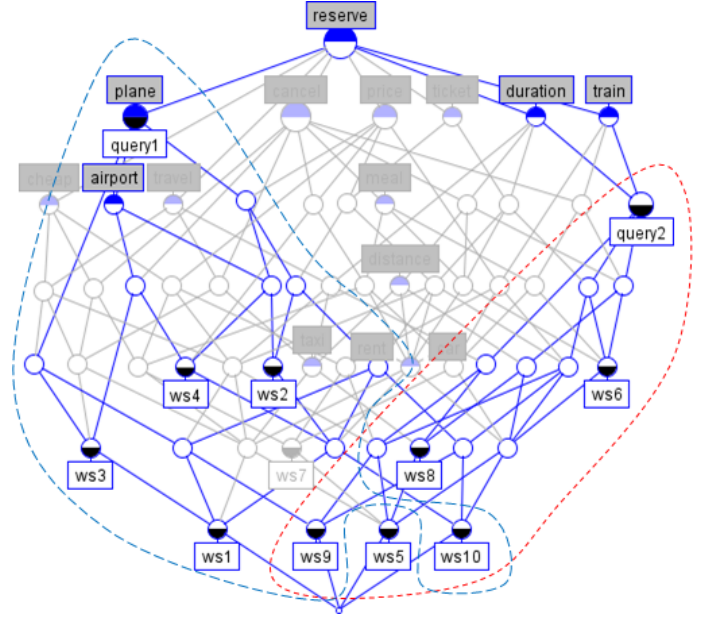

Figure 5.   Queries as concepts in the $service \times keyword$ lattice.

like for example the operation $rentCar$. When we select a certain service, we can immediately extract the set of backup services that are able to replace it if it fails. In the same way, we can select a service for train reservation with obtaining the duration information. Thus, the composition in Figure 1 can be easily achieved and supported with backups, as in Figure 6. Supposing that we selected the service
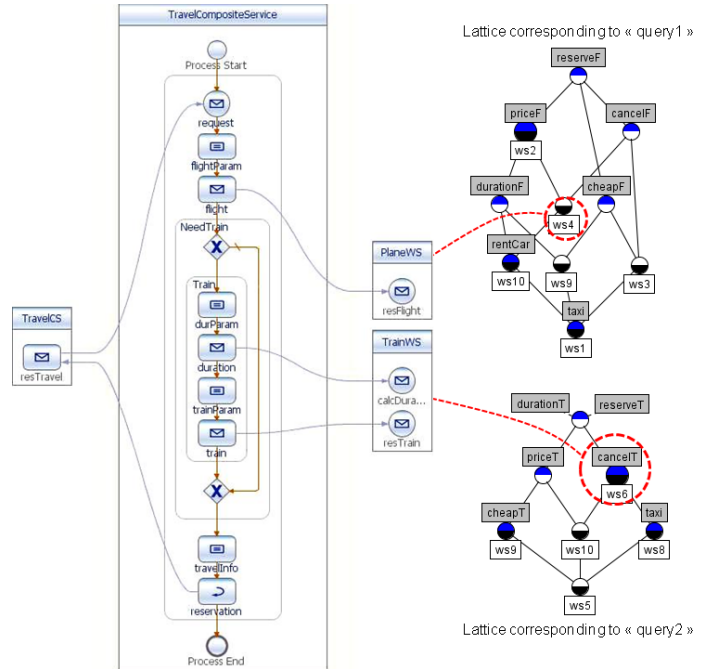

Figure 6.   The TCS composition with its corresponding backups.

ws_4 named $PlaneWS$, and used its operation $resFlight$

(which is grouped with other similar operations under the name $reserveF$ in the lattice). We can notice that for the operation $resFlight$, any other service in the lattice can be a backup for ws_4. In case where all of the operations of ws_4 were used, we notice that only ws_{10} and ws_1 can be backups for ws_4. Similarly, we selected the service ws_6, named $TrainWS$, and used two of its provided operations: $calcDuration$ and $resTrain$. We notice that ws_6 can be replaced by the services ws_{10}, ws_8 and ws_5. Thus, we have discovered immediate backups for the service $TrainWS$ as we did for the service $PlaneWS$. We can notice that the service ws_{10} exists in the two lattices, as a backup for $TrainWS$ and $PlaneWS$ together. In this case, if both of these services crash, we can replace them both by only one service, which is ws_{10}. This service provides the same functionalities of the two services, with three extra operations that are $priceT$, $rentCar$ and $durationF$ as can be seen in the lattices in Figure 6.

Our approach has enabled us of an easy service discovery and selection, in order to build our aimed scenario. It has also facilitated the discovery of backup services to support our service composition and ensure its continuous functionality.

VI. 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 [8], [9]. Below, we describe a selection of works, classified according to their adapted techniques.

A. Using formal concept analysis (FCA)

In [10], 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 [11], 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 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. [12], similarity values are calculated for service operations, and depending on a chosen threshold, a service lattice is built.

B. Using machine learning

Many approaches adapt techniques from machine learning field, in order to discover and group similar services. In [13], [14], 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 [15], CPLSA approach is defined that reduces a services set then clusters it into semantically related groups.

C. 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 [17], 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 [18], 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 [19], tags coming from folksonomies are used to discover and compose services.

D. Using vector space model techniques

The vector space model is used for service retrieval in several existing works as in [20], [21], [22]. 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 [21], or by partitioning the space into subspaces to reduce the searching space as in [22].

E. 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 VI-B, VI-C and VI-D). 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.

VII. CONCLUSION

In this paper, we proposed an approach based on formal concept analysis (FCA) for classifying web services. 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. An obvious drawback of current web services is the lack of a common definition, which hardens the identification of similar services and necessitates more adaptations when replacing a service with another.

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 corresponds 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] M. P. Papazoglou, *Web Services: Principles and Technology*. Pearson, Prentice Hall, 2008.

[2] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *VLDB '04: Proc. of the 30th int. conf. on Very large data bases*, pages 372–383. 2004.

[3] E. Newcomer and G. Lomow, *Understanding SOA with Web Services (Independent Technology Guides)*. Addison-Wesley Professional, 2004.

[4] E. Stroulia and Y. Wang. Structural and semantic matching for assessing web-service similarity. *Int. J. Cooperative Inf. Syst.*, 14(4):407–438, 2005.

[5] 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.

[6] B. Ganter and R. Wille, *Formal Concept Analysis - Mathematical Foundations*, Springer, 1999.

[7] J.-R. Falleri, Z. Azmeh, M. Huchard, and C. Tibermacine, "Automatic tag identification in web service descriptions," in *proceedings of WEBIST2010. To appear.*

[8] S. Brockmans, M. Erdmann, and W. Schoch, "Service-finder deliverable d4.1. research report about current state of the art of matchmaking algorithms," Tech. Rep., October 2008.

[9] H. Lausen and N. Steinmetz, "Survey of current means to discover web services," Semantic Technology Institute (STI), Tech. Rep., August 2008.

[10] L. Aversano, M. Bruno, G. Canfora, M. D. Penta, and D. Distante, "Using concept lattices to support service selection," *Int. J. Web Service Res.*, vol. 3, no. 4, pp. 32–51, 2006.

[11] M. Bruno, G. Canfora, M. D. Penta, and R. Scognamiglio, "An approach to support web service classification and annotation," in *EEE*. IEEE Computer Society, 2005, pp. 138–143.

[12] 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.

[13] M. Crasso, A. Zunino, and M. Campo, "Awsc: An approach to web service classification based on machine learning techniques," *Revista Iberoamericana de Inteligencia Artificial*, vol. 12, No 37, pp. 25–36, 2008.

[14] A. Heßand N. Kushmerick, "Learning to attach semantic metadata to web services," in *International Semantic Web Conference*, 2003, pp. 258–273.

[15] J. Ma, Y. Zhang, and J. He, "Efficiently finding web services using a clustering semantic approach," in *CSSSIA '08*. New York, NY, USA: ACM, 2008, pp. 1–8.

[16] J. Lu and Y. Yu, "Web service search: Who, when, what, and how," in *WISE Workshops*, 2007, pp. 284–295.

[17] A. Günay and P. Yolum, "Structural and semantic similarity metrics for web service matchmaking," in *EC-Web*, 2007, pp. 129–138.

[18] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services," in *VLDB '04: Proceedings of the international conference on Very large data bases*. VLDB Endowment, 2004, pp. 372–383.

[19] E. Bouillet, M. Feblowitz, H. Feng, Z. Liu, A. Ranganathan, and A. Riabov, "A folksonomy-based model of web services for discovery and automatic composition," in *(SCC)*. IEEE Computer Society, 2008, pp. 389–396.

[20] C. Platzer and S. Dustdar, "A vector space search engine for web services," in *Third IEEE European Conference on Web Services, ECOWS 2005.*, 2005, pp. 62–71.

[21] Y. Wang and E. Stroulia, "Semantic structure matching for assessing web service similarity," in *1st International Conference on Service Oriented Computing (ICSOC03)*. Springer-Verlag, 2003, pp. 194–207.

[22] 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*. New York, NY, USA: ACM, 2008, pp. 2376–2380.

[23] http://www.w3.org/TR/wsdl, last access 10-02-24

[24] http://www.uddi.org/pubs/uddi_v3.htm, last access 10-02-24