Selection of Composable Web Services Driven by User Requirements

Zeina Azmeh*, Maha Driss†, Fady Hamoui*, Marianne Huchard*, Naouel Moha‡ and Chouki Tibermacine*
*LIRMM, CNRS, University of Montpellier, France
{azmeh, hamoui, huchard, tibermacin}@lirmm.fr
†IRISA, INRIA, University of Rennes I, France
{mdriss}@irisa.fr
‡Université du Québec à Montréal, Canada
{moha.naouel}@uqam.ca

***Abstract*—Building a composite application based on Web services has become a real challenge regarding the large and diverse service space nowadays. Especially when considering the various functional and non-functional capabilities that Web services may afford and users may require.**

In this paper, we propose an approach for facilitating Web service selection according to user requirements. These requirements specify the needed functionality and expected QoS, as well as the composability between each pair of services. The originality of our approach is embodied in the use of Relational Concept Analysis (RCA), an extension of Formal Concept Analysis (FCA). Using RCA, we classify services by their calculated QoS levels and composability modes. We use a real case study of `901` services to show how to accomplish an efficient selection of services satisfying a specified set of functional and non-functional requirements.

***Keywords*-Web service selection; Web service composition; Formal Concept Analysis (FCA); Relational Concept Analysis (RCA); User requirements; QoS.**

I. Introduction

Service-Oriented Computing (SOC) is an emerging paradigm for developing low-cost, flexible, and scalable distributed applications based on services [1]. Web services represent a realization of SOC. They are autonomous, reusable, and independent software units that can be accessed through the internet. SOC is becoming broadly adopted and in particular by organizations, which are more and more willing to open their information systems to their clients and partners over the Internet. The reason comes from the fact that SOC offers the ability to build efficiently and effectively added-value service-based applications by composing ready-made services. Web service composition addresses the situation when the functionality required by users (developers) cannot be satisfied by any available Web service, but by assembling suitably existing services [2]. When building Web service-based applications, we have to face several issues such as:

- Web service retrieval from the large number of existing services, and the lack of efficient indexing mechanisms. Current solutions are embodied in search engines, like: Seekda [3] and Service-Finder [4];
- a service's ability to meet the user's functional and non-functional requirements;
- a service's composability, or the degree to which a service can be composed with another, considering the needed adaptations;
- the last issue is how can we achieve a compromise for service selection in the light of the previous issues.

Discovering and selecting services that closely fit users' functional and non-functional requirements is an important issue highly studied in the literature as in [5], [6], [7]. Functional requirements define functionalities provided by Web services and non-functional requirements define Quality of Service (QoS) criteria such as availability, response time, and throughput [8]. However, discovering and selecting relevant composable services is another important issue that still need to be investigated, since few approaches focused on service composability problem [9], [10], [11]. Relevant composable services represent services that minimize the amount of adaptation among them while best fitting requirements.

In this paper, we propose an approach for the identification of services that best fit QoS and composition requirements. This approach is based on Relational Concept Analysis (RCA) [12], a variant of Formal Concept Analysis (FCA) [13], [14]. FCA has been successfully used as a formal framework for service substitution [15], [16], [17], [18], [19]. The approach also integrates a query mechanism that allows users to specify their required QoS and composability levels. Thereafter, the generated lattices help in identifying the services that match the specified queries.

The paper is organized as follows: Section II gives an overview of the FCA and RCA classification techniques. Section III describes our approach. Section IV describes the experiments performed on a real case study for validating our approach. The paper ends with the related work in Section V and the conclusion in Section VI.

II. Background

In this section, we give the basic definitions of FCA and RCA, along with a simple example of exploiting them.

A. Formal Concept Analysis (FCA)

We base our approach on FCA [13], [14] which is a classification method that permits the identification of groups

of objects having common attributes. It takes a data set represented as an n×m table (formal context) with objects as rows and attributes as columns. A cross "×" in this table means that the corresponding object has the corresponding attribute. An example of a formal context is shown in Table I, for a set of objects O={1,2,3,4,5,6,7,8,9,10} and a set of attributes A={odd,even,prime,composite,square}.

Table I: A formal context for objects O and attributes A.

	odd	even	prime	composite	square
1	×				×
2		×	×		
3	×		×		
4		×		×	×
5	×		×		
6		×		×	
7	×		×		
8		×		×	
9	×			×	×
10		×		×	

From a formal context, FCA extracts the set of all the formal concepts. A formal concept is a maximal set of objects (called extent) sharing a maximal set of attributes (called intent). For example, in Table I, a=({4,6,8,10},{even, composite}) is a formal concept because the objects 4, 6, 8, and 10 share exactly the attributes *even* and composite (and vice-versa). On the other hand, ({6},{even, composite}) is not a formal concept because the extent {6} is not maximal: other objects share the same set of attributes.

FCA reveals the inheritance relations (super-concept and sub-concept) between the extracted concepts and organizes them into a partially ordered structure known as Galois lattice or concept lattice. The resulting concept lattice is illustrated in Figure 1(L).

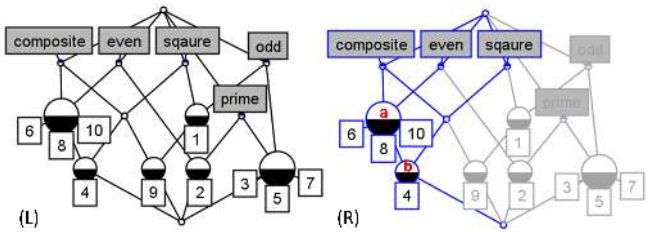

Figure 1: Formal concept lattice for the context in Table I (L); focus on the concept b (R). Lattices are built with Concept Explorer (ConExp) tool [20].

This lattice reveals phenomena that may not be recognized intuitively. For example, in Figure 1(R) appears the concept b=({4},{composite,even,square}) as a sub-concept of the concept a. It inherits a's attributes composite and even, and extends it by the square.

B. Relational Concept Analysis (RCA)

RCA [12] is an extension of FCA that takes into consideration the relations between the objects. Thus, it takes as input two types of contexts: (non-relational) ones that are previously used with FCA to classify objects by attributes, and inter-context (relational) ones that represent the relations between the objects. RCA generates lattices similar to the ones generated by FCA, but enriched with the information about the relation between the objects. We take as an example two sets of numbers, {1,2,3,4,5} and {11,12,13,14,15,16,17,18,19,20}. We build two non-relational contexts similar to the one in Table I. We consider a relation called Divides between the first and second sets of numbers, and we build the relational context in Table II. RCA takes the two non-relational contexts

Table II: The relational context Divides.

Divides	11	12	13	14	15	16	17	18	19	20
1	×	×	×	×	×	×	×	×	×	×
2		×		×		×		×		×
3		×			×			×		
4		×				×				×
5					×					×

(numbers×attributes), and the relational context Divides, then generates the two lattices in Figure 2.

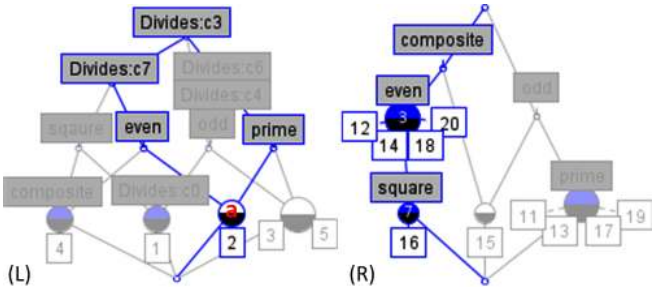

Figure 2: The enriched lattices generated by RCA.

These lattices are similar to FCA lattices, but one of them is enriched with the relation *Divides*. For example: by regarding the concept a=({2},{prime,even, Divides:c7,Divides:c3}) in lattice (L), we notice that numbers of its extent can divide numbers of the extents of the concepts 7 and 3 in lattice (R). In the general case, where relations form directed cycles between objects, RCA applies iteratively. During this iteration, several scaling operators can be used. Here we use the existential one (see [12] for more details).

III. Approach

Using our approach, a user can specify an abstract process described as a set of functional and non-functional requirements. The functional requirements describe the needed tasks, while the non-functional requirements describe the expected QoS and mode of composition for these tasks. The approach retrieves sets of Web services, filters and analyzes their data according to the user requirements. Then, it classifies them in concept lattices based on RCA classification

technique (II-B). This latter enables a simplified selection of Web services that best match the specified requirements and thus, allows the realization of a desired process with less time and effort.

We explain our approach in view of Figure 3, which illustrates the different components and their course of action that lead to service selection.
We describe these components respectively along with an example of an abstract process of three tasks {Task1, Task2, Task3}, as follows:

A. User Requirements Analyzer:

The approach starts by analyzing the requirements specified by the user via a description file (see Figure 4), which is composed of the following elements:

1) Functional requirements: this part is described by a set of tasks. Each task is described by its input and output parameters via their names and types. For each parameter, a user specifies a set of relevant keywords for its name, and another set for its type. For example, Task1 has one input parameter, which is defined by {InParamNameKeys1, InParamTypeKeys1}. We enabled the user of providing more than one keyword, in order to retrieve more relevant services. For example: if a user needs a "date" parameter, he/she may specify the possible keywords for its name as {date} and types (primitive or complex) as {string, date}.
2) Non-functional requirements: this part is composed of two other parts.
   - QoS specification, which specifies the requested QoS level for each task according to the supported QoS attributes (qos_x, qos_y, etc.).
   - Composition specification, which specifies the mode of composition (links) between each two consecutive tasks, for example: Task1 $\rightarrow$ Task2. The mode of composition describes whether Task1 (source) covers all the input of Task2 (target) or not. This notion is better described below.

```
Functional:                NonFunctional:
  Task: Task1                QoS:
    Input:                     Attribute: qos1
      Keywords: k1               Task: task1
      Type: t1                   Level: Good
    Output:                      ...
      ...                    Composition:
  ...                          Link:
  ...                            Source: task1
  ...                            Target: task2
  ...                            Mode: FC
  ...                          ..
```

Figure 4: An excerpt of the user's requirements file.

The analyzer extracts the information related to these functional and non-functional requirements, which is used by the remaining components as described below.

B. Web Service Retriever:

The requirements analyzer sends to this component the keywords provided for the parameter names input/output for each task {InParamNameKeys, OutParamNameKeys}. In our example, the retriever searches and retrieves a set of services: WS1.i, WS2.j, and WS3.k, corresponding respectively to each of the three tasks. It also gathers the QoS values (the supported attributes) for each retrieved service (qos_x.i, qos_y.i, etc.).

C. WSDL Parser:

Each set of the retrieved services is passed to this component[1], in order to extract for each service its operations with their input/output parameters.

Using the information extracted from the parsed services, we can check a service's compatibility.

D. Compatibility Checker:

This component checks whether a service provides an operation that can satisfy the corresponding task. An operation satisfies a task when it contains the requested input/output parameters names. We verified this by using the Jaro-Winkler string distance measure [21]. By doing so, we discovered three possible cases:

- compatible, there exists one operation at least that satisfies the corresponding task and has the same parameters types; or it may become:
- adaptable compatible, meaning that none of the satisfying operations has the same parameter types (either for input, or output, or both), thus type adaptations need to be done; otherwise:
- incompatible, the service does not satisfy the corresponding task.

The compatibility checker reduces the number of the retrieved services, by omitting the incompatibles ones (WS1.i becomes WS1.i'). Thus, it keeps a detailed list of the compatible services together with their satisfying operations.

Once we identified the compatible services, we can measure the mode of composition and the QoS levels using the two following components.

E. Composability Evaluator:

Composing two Web services is finding two operations (one of each) that can be linked. Two operations can be linked when the output parameters of the first (source) can cover the input parameters of the second (target). We can define two composition modes according to the coverage of the input parameters of a target operation, in addition to two other modes according to the needed adaptations that are discovered by the compatibility checker. They are:

[1]Available online: http://www.lirmm.fr/~azmeh/tools/WsdlParser.html

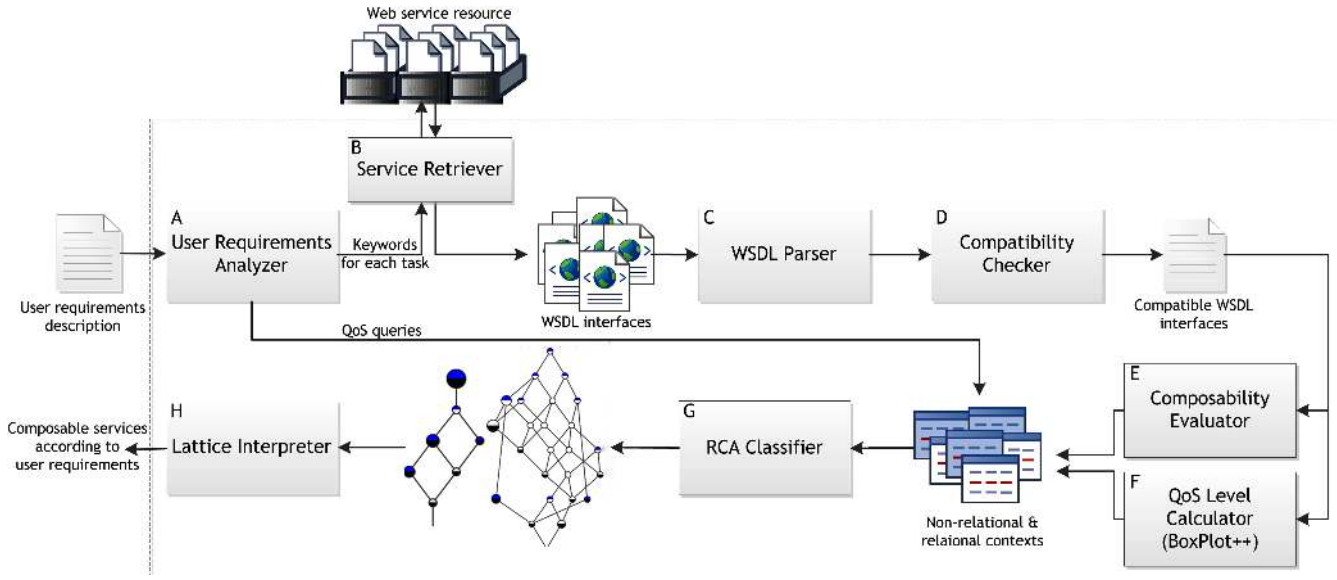

Figure 3: An overview of the approach's components in their course of action.

- `Fully-Composable (FC)`, when a source operation covers by its outputs all of the expected inputs of a target operation;
- `Partially-Composable (PC)`, when one or more input parameters of a target operation are not covered;
- `Adaptable-Fully-Composable (AFC)`, when the source and target operations have an `FC` mode, but need some type adaptations either for the output of the source or the input of the target, or both of them; and
- `Adaptable-Partially-Composable (APC)`, similar to `AFC` but when having a `PC` mode.

Thus, having the composition `Task1` $\rightarrow$ `Task2`, the composability evaluator determines the mode of composition for all the services in `WS1.i'` with all the services in `WS2.j'`. Then, it generates four relational contexts (Section II-B) between `WS1.i'` and `WS2.j'` according to the four previously defined modes of composition. These contexts are exploited by the RCA classifier to clarify which services can be composed and following which mode. This is illustrated in the experiments section IV.

F. QoS Levels Calculator:

This component takes into consideration the QoS values for all the sets of compatible services $(qos_x.i', qos_x.j', qos_x.k', etc.)$. It extracts these values from the ones returned by the service retriever, according to the list returned by the compatibility checker. Web services have many QoS attributes and different ranges of numerical values for each one of these attributes. In order to have a better overview of these values, we apply a statistical technique called BoxPlot++ [22] to cluster the convergent values together. The BoxPlot++ is an extension of the original boxplot [23] technique. It takes as input a given set of numerical values, and produces one to seven corresponding levels of values: `L = {BadOutlier,VeryBad,Bad,Medium,Good, VeryGood,GoodOutlier}`[2]. The technique is applied on each QoS attribute. Then, it generates for each set of services a non-relational context (Section II-A) having all of its QoS attributes levels. These contexts are exploited afterwards by RCA classifier, in order to classify the services according to these different QoS levels.

G. RCA Classifier:

This component takes into consideration the relational contexts of composition modes and the non-relational contexts of QoS levels. It also uses the non-functional requirements provided in the user requirements file. These requirements are considered as queries that are added to the two types of contexts as follows:
The QoS specifications for each task are integrated to the corresponding non-relational context as a new row. The composition specifications are also integrated to the corresponding relational context in the same way.

Finally, the RCA classifier [24] generates all the corresponding service lattices and passes them to the final component. This component is further detailed in [25].

H. Lattice Interpreter:

By integrating the non-functional queries into the contexts, they appear inside the concepts of the corresponding lattices. This enables this component of locating the services that satisfy the queries and to navigate between the different solutions. These services are present at the sub-concepts of the queries concepts. This is better illustrated in section IV.

[2]Available online: http://www.lirmm.fr/~azmeh/tools/BoxPlot.html

IV. EXPERIMENTS

We applied our approach on an abstract process, which is supposed to provide the weather information for a given ip address. It is described by a user requirements file (Section III) using three tasks: Task1, Task2, and Task3. Task1 takes as input an `ip address` and provides Task2 with the corresponding `city name`. Task2 takes the `city name` and returns to Task3 the corresponding `zipcode`. Finally, Task3 returns the corresponding `weather information`. From this file, the experiments are conducted on four steps as follows:

1. Collecting Services: We use the set of keywords describing each task to search and retrieve sets of corresponding Web services. We make use of the Service-Finder Web service search engine [4] to collect a set of corresponding endpoints (WSDL addresses). This engine also provides us with values of two QoS attributes: availability `(A)` and response time `(RT)`. We download the corresponding WSDL files after omitting repeated and invalid endpoints. We show in Table III each task with its keywords as well as the number of obtained endpoints, the number of retrieved WSDL files, and the sets identifiers. In this step, we make use of the requirements analyzer components (III-A) and the service retriever (III-B).

Table III: Summary of the retrieved services.

Task	Keywords	#Endpoints	#Services	SetID
1	{ip,ipAddress}, {city,cityName}	94	94	`WS1.i`
2	{city,cityName}, {zip,zipcode,postal,postalcode}	768	760	`WS2.j`
3	{zip,zipcode,postal,postalcode}, {weather,weatherInfo,forecast, weatherForecast,weatherReport}	39	37	`WS3.k`

2. Filtering the Services: In this step, we parse the WSDL files using the WSDL parser (III-C) and remove the invalid ones (Filter1). Then we calculate the compatible ones (Filter2) using the Compatibility checker (III-D). In Table IV, we can see the resulting number of filtered services for each set.

Table IV: The number of filtered services for each set.

	`WS1.i`	`WS2.j`	`WS3.k`
Filter1 (Valid)	94	748	37
Filter2 (Compatible)	17	96	21

3. Composability and QoS: In this step, we calculate the composition modes for the compatible sets of services (Table V) as well as their QoS levels. We use the Composability evaluator (III-E) and the QoS level calculator (III-F).

The resulting composition modes and QoS levels are organized into non-relational and relational contexts (see [26]), and are used to classify the services in the next step.

4. RCA-Based Classification: During this step, the RCA classifier (III-G) takes the contexts formed in the previous step and integrates the QoS queries. The queries that we choose in this case study are specified to be `Good A` and `Good RT` levels for each task in the process. We also require an `FC` composition mode for both `(Task1,Task2)` and `(Task2,Task3)`. The generated lattices are illustrated in Figure 5.

Table V: Number of services per composition mode.

	`WS1.i'`	`WS2.j'`	`WS3.k'`
# `FC` services	12	3	12
# `PC` services	4	89	4
# `AFC` services	2	1	11
# `APC` services	0	3	2

These lattices are finally interpreted by the lattice interpreter (III-H), considering two rules:

- in each lattice, the services satisfying the corresponding query (QoS and composition) appear in the subconcepts of the concept where the query appears. Example: the services in `c0 (WS1.i)` satisfy `Query1`;
- the services located closer to the bottom of a lattice offer better QoS levels than the farther ones, for example: in the lattice `(WS2.j)`, the service `WS2.8` is better than service `WS2.198` because it has a `VeryGood A` (an inherited attribute). On the other hand, the services located in a same concept have convergent QoS levels.

Following the precedent rules, the lattice interpreter extracts the following services to be the best choice regarding the specified requirements:
{`WS1.59,WS1.5,WS1.3`} for `Task1` because they all appear in the same concept `(c0)`; {`WS2.8`} for `Task2` because it is better than {`WS2.198`}; and {`WS3.23`} for `Task3` because it is better than {`WS3.1`}. If we verify the actual services, we get the information in Table VI.

Table VI: Information about the services satisfying the queries with the selected ones (highlighted).

WS	Name	Operation	A(%)	RT(ms)
1.59	Ip2LocationWebService	IP2Location	100	257
		(in) IP:string (out) CITY:string,..		
1.5	GeoCoder	IPAddressLookup	100	328
		(in) ipAddress:string (out) City:string,..		
1.3	IP2Geo	ResolveIP	100	798
		(in) ipAddress:string (out) City:string,..		
2.198	MediCareSupplier	GetSupplierByCity (in) City:string (out) Zip:string,..	85	304
2.8	ZipcodeLookupService	CityToLatLong	100	439
		(in) city:string (out) Zip:string,..		
3.1	USWeather	GetWeatherReport (in) ZipCode:string (out) WeatherReport:string	85	384
3.23	Weather	GetCityForecastByZIP	100	237
		(in) ZIP:string (out) ForecastReturn:complex		

These service lattices offer a browsing mechanism that

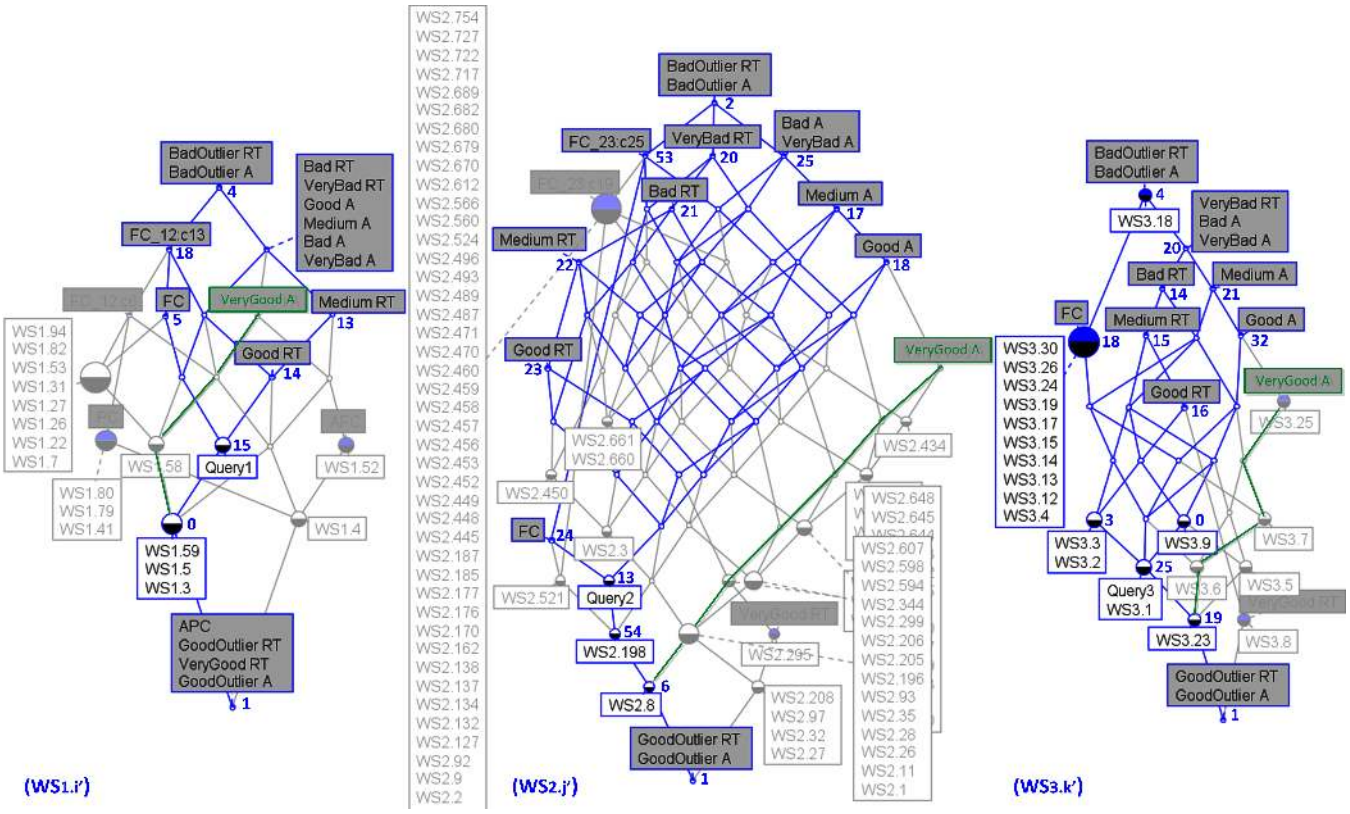

Figure 5: The concept lattices for the compatible sets of services with `Good A`, `Good RT` queries, and `FC` mode.

facilitates services selection according to user requirements. In each lattice, services are classified by their QoS levels as well as their composition modes with services in a following lattice. A lattice reveals two relations between the services regarding QoS: *hasSimilarQoS* when services are located in a same concept and *hasBetterQoS* when a service is a descendant of another service(s). Services having a *hasSimilarQoS* or *hasBetterQoS* relations with a selected service, are considered to be its alternatives. User requirements (queries) can be expressed as new services to be classified in the corresponding lattices. This locates the part of the lattice that meets the user requirements and thus represent an efficient lattice navigation mechanism. In the case where no services could be found for the specified requirements, the query mechanism enables the user to identify the next best service selection. This service will have lesser QoS than the requested and it represents the direct ascendant of the query concept. For example: in the third lattice of Figure 5, we could take `WS3.2` or `WS3.3` in concept `c3` to be the next best selection. They have a `Medium RT` and a `Good A`.

In this experiment, we had several functional and non-functional requirements needed to build a simple process of three tasks as described previously. The Service-Finder enabled us to find a total of `901` (Table III) Web service addresses, among which we have to identify and retrieve the services meeting our requirements. Using our approach, we efficiently identified out of 901 endpoints a set of five services that best match our requirements (VI). The total time required to extract these services is equal to `103` sec[3], starting from the `WSDL Parser` (component `C`) till the end.

V. Related Work

We list the related work according to three categories:

Web Service Composability: The Web service Composability problem have been addressed by many works [10], [27], [28], [9], [11]. Ernst *et al.* [10] present an approach based on syntactic descriptions of Web services. This approach detects the matching between Web service's operations by analyzing the results obtained after multiple invocations of these services. The input and output parameter values are compared syntactically and matchings are deduced. Contrarily to our approach, which focuses on the syntactic, semantic, and QoS descriptions of Web services, this work is based on the experimental usage of Web services, and is perfectly complementary to ours. In this research area, much work has been done also on semantic Web services [27], [28], [9], [11]. In [27], Sycara *et al.* present DAML-S, an ontology for semantic Web service description. They show the use of this ontology for service

[3]Calculated by `NetBeans (6.9.1)` using `Intel Core 2 Duo (1.80GHz)` and a `RAM` of `(2.00 GB)`.

discovery, interaction, and composition. Two implementations are proposed: the DAML-S/UDDI Matchmaker that provides semantic capability matching and the DAML-S Virtual Machine that manages the interaction with Web services. Contrarily to our approach, this work do not deal with QoS properties to compose Web services. In [28] and [9], Medjahed *et al.* present a model for checking the composability of semantic Web services at different levels: syntactic, semantic, and QoS. Two kinds of composability are defined: horizontal (normal composability) and vertical (substitutability). This work deals with three QoS properties: fees, security, and privacy. In our approach, we may consider any QoS property. We used availability and response time provided by Service-Finder. In [11], Lécue *et al.* present a method which combines semantic and static analysis of messages in semantic Web services. Data types (XML schemas) and semantic description (domain ontologies and SA-WSDL specifications) of parameters are used to deduce mappings. These mappings are then transformed into adapters (XSL documents). In this work, the focus is on the generation of adapters of Web service data flows. This enhances greatly the composability of Web services which are not directly composable. In our work, we concentrated on the selection of services that best fit the user's QoS and composability requirements. The two approaches are complementary.

QoS-Based Web Service Selection: Many approaches like [5], [6], [7] have been proposed to solve the problem of QoS-based Web service selection. In [5], Zeng *et al.* present a middle ware platform that enables the quality-driven composition of Web services. In this platform, the QoS is evaluated by means of an extensible multidimensional model, and the selection of Web services is performed in such a way as to optimize the composite service's QoS. Aggarwal *et al.* [6] present a constraint driven Web services composition tool that enables the selection and the composition of Web services considering QoS constraints. Like Zeng *et al.* [5], a linear integer programming approach is proposed for solving the optimization problem. In [7], Yu *et al.* propose heuristic algorithms to find a near-to-optimal solution more efficiently than exact solutions. They model the QoS-based service selection and composition problem in two different ways: a combinatorial model and a graph model. A heuristic algorithm is introduced for each model. The QoS-based service selection problem is viewed by all the works detailed above [5], [6], [7] as an optimization problem that aims to find the service that best fit QoS requirements from the set of candidate services. The advantage of our approach compared to these works is that FCA provides equivalence classes of services (the extents formal concepts provide classes of services that share the same characteristics). If a service in an application fails, one of the other services in the class can replace it. This provides considerable enhancement for the dynamic composition since it reduces the reaction time.

Web Service Classification Using Concept Lattices: Many works in the literature like [15], [16], [17], [18], [29], [19], [30] have addressed the classification of Web services using concept lattices. Peng *et al.* [15] present an approach to classify and select services. They build lattices upon contexts where individuals are Web services and properties represent the operations of these services. The approach allows similar services clustering by applying similarity search techniques that compare operation descriptions and input/output messages data type. Aversano *et al.* [15] use FCA, to understand relationships between services, as well as between operations of a complex service, by analyzing service interfaces and documentation. Concept lattices are built upon a context obtained from keywords extracted from service descriptions or operation parameters. They cluster similar services, highlight hierarchical relationships, commonalities, and differences between services. Azmeh *et al.* [17] present a similar approach to classify and select services using the FCA. They propose WSPAB tool that permits the discovery, the automatic classification, and the selection of Web services. Classification is accomplished by defining a binary relation between services and operation signatures. In [18], Azmeh *et al.* uses FCA to classify Web services by keywords extracted from their WSDL description files, to identify relevant services and their substitutes. Fenza and Senatore [29] describes a system for supporting the user in the discovery of semantic Web services that best fit personal requirements and preferences. Through a concept-based navigation mechanism, the user discovers conceptual terminology associated to the Web services and uses it to generate an appropriate service request which syntactical matches the names of input/output specifications. The approach exploits the fuzzy FCA for modelling concepts and relative relationships elicited from Web services. After the request formulation and submission, the system returns the list of semantic Web services that match the user query. Contrarily to our approach, these works [15], [16], [17], [18], [29] do not deal with QoS properties to classify Web services. In [19], Chollet *et al.* propose an approach based on FCA to organize the services registry at runtime and to allow the best service selection among heterogeneous and secured services. The services registry is viewed as a formal context where the services are the individuals and the services types, functional, and non-functional characteristics (security characteristics) are properties. In [30], Driss *et al.* propose a requirement-centric approach to Web services, modelling, discovery, and selection. They consider formal contexts with services as individuals and QoS characteristics as properties. The obtained lattices are used to check out relevant (that best fit functional requirements) and high QoS Web services. All the works detailed above [15], [16], [17], [18], [29], [19], [30] are based on FCA, and not on RCA. In addition, they do not deal with service composition since they propose to classify and select only individual services.

VI. Conclusion

In this paper, we presented an approach for facilitating Web service selection according to user functional and non-functional requirements. This approach is based on four principal steps; service collecting, validity and compatibility filtration, QoS levels calculating, and RCA classification. The resulting lattices group services that have common QoS and composition levels. User requirements are expressed as new services and are classified in the corresponding lattices. This locates the part of the lattice that meets the user requirements. We validated our approach on a set of 901 real-world Web services obtained by querying Service-Finder. Experimental results show that our approach allows an efficient selection of services satisfying the specified functional and non-functional requirements.

Future work includes: (i) enhancing the composability evaluator component by considering advanced semantic and syntactic similarity techniques; (ii) proposing keywords to the user to help her/him in specifying the needed tasks more efficiently; and (iii) performing experiments on more complex compositions.

References

[1] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *IEEE Internet Computing*, vol. 9, no. 1, pp. 75–81, 2005.

[2] J. Rao and X. Su, *A Survey of Automated Web Service Composition Methods*. LNCS, Springer, 2005.

[3] Seekda, http://webservices.seekda.com/.

[4] Service-Finder, http://www.service-finder.eu/.

[5] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[6] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint driven web service composition in meteor-s," in *Proc. of SCC '04*, pp. 23–30, 2004.

[7] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Transactions on the Web*, vol. 1, no. 1, p. 6, 2007.

[8] D. A. Menascé, "Qos issues in web services," *IEEE Internet Comp.*, vol. 6, no. 6, pp. 72–75, 2002.

[9] B. Medjahed and A. Bouguettaya, "A multilevel composability model for semantic web services," *IEEE Trans. on Knowledge and Data Eng.*, vol. 17, no. 7, pp. 954–968, 2005.

[10] M. D. Ernst, R. Lencevicius, and J. H. Perkins, "Detection of web service substitutability and composability," in *WS-MaTe 2006*, pp. 123–135, 2006.

[11] F. Lécue, S. Salibi, P. Bron, and A. Moreau, "Semantic and syntactic data flow in web service composition," in *Proc. of ICWS'08*, pp. 211–218, 2008.

[12] M. Huchard, M. R. Hacene, C. Roume, and P. Valtchev, "Relational concept discovery in structured datasets," *Annals of Mathematics and Artificial Intelligence*, vol. 49, no. 1-4, pp. 39–76, 2007.

[13] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Inc., 1999.

[14] R. Wille, "Restructuring lattice theory: an approach based on hierarchies of concepts," *Ordered Sets*, vol. 83, pp. 445–470, Sep. 1982.

[15] D. Peng, S. Huang, X. Wang, and A. Zhou, "Management and retrieval of web services based on formal concept analysis," in *Proc. of CIT'05*, pp. 269–275, 2005.

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

[17] Z. Azmeh, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier, "Wspab: A tool for automatic classification and selection of web services using formal concept analysis," in *Proc. of ECOWS'08*, 2008.

[18] Z. Azmeh, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier, "Using concept lattices to support web service compositions with backup services," in *Proc. of ICIW'10*, 2010.

[19] S. Chollet, V. Lestideau, P. Lalanda, P. Colomb, and D. Moreno, "Heterogeneous service selection based on formal concept analysis," in *Proc of Int. W. on Net-Centric Service Enterprises: Theory and Application (NCSE2010)*, 2010.

[20] Conexp. [Online]. Available: http://conexp.sourceforge.net/

[21] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *IIWeb*, S. Kambhampati and C. A. Knoblock, Eds., pp. 73–78, 2003.

[22] Z. Azmeh, F. Hamoui, and M. Huchard, "BoxPlot++," Tech. Rep., 01 2011. [Online]. Available: http://hal-lirmm.ccsd.cnrs.fr/lirmm-00557222/en/

[23] R. McGill, J. W. Tukey, and W. A. Larsen, "Variations of Box Plots," *The American Statistician*, vol. 32, pp. 12–16, 1978.

[24] GaLicia, "Galois lattice interactive constructor," 2002, http://www.iro.umontreal.ca/~galicia - accessed on Sept. 22, 2008.

[25] Z. Azmeh, M. Driss, M. Huchard, N. Moha, and C. Tibermacine, "QoS-Driven Selection of Composable Web Services," Tech. Rep., 06 2010. [Online]. Available: http://hal-lirmm.ccsd.cnrs.fr/lirmm-00565357/en/

[26] RCA contexts, http://www.lirmm.fr/~azmeh/icws11/rca-contexts.html.

[27] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated discovery, interaction and composition of semantic web services," *Journal of Web Semantics, Elsevier*, vol. 1, pp. 27–46, 2003.

[28] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing web services on the semantic web," *The VLDB Journal*, vol. 12, pp. 333–351, 2003.

[29] G. Fenza and S. Senatore, "Friendly web services selection exploiting fuzzy formal concept analysis," *Soft Comput.*, vol. 14, pp. 811–819, June 2010.

[30] M. Driss, N. Moha, Y. Jamoussi, J.-M. Jézéquel, and H. Hajjami Ben Ghézala, "A requirement-centric approach to web service modeling, discovery, and selection," in *Proc. of ICSOC'10*, 2010.