The Journal of Systems and Software 86 (2013) 835–853

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

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

Communities of Web service registries: Construction and management

Mohamed Sellami[a,*], Olfa Bouchaala[a,b], Walid Gaaloul[a], Samir Tata[a]

[a] Institut TELECOM, TELECOM & Management SudParis, CNRS UMR Samovar, Evry, France
[b] ReDCAD Laboratory, National Engineering School of Sfax, Tunisia

ARTICLE INFO

Article history:
Received 26 April 2011
Received in revised form 8 November 2012
Accepted 13 November 2012
Available online 22 November 2012

Keywords:
Service oriented computing
Organization of Web service registries
Communities of registries
Managing communities of registries

ABSTRACT

The last few years have seen a democratization in the use of Internet technologies, mainly Web services, for electronic B2B transactions. This has triggered an increase in the number of companies' Web service registries. In this paper, we propose to use communities as a means to organize Web service registries in such a context. We provide an automatic and implicit approach to create communities of Web service registries using registries' WSRD descriptions. We also define the needed management operations to ensure the communities consistency during a registry/community life-cycle. Experiments we have made show the feasibility and validity of our community creation approach as well as the specified managing operations.

1. Introduction

According to the CIA's world Factbook, the services-based economy represents 62.9% of the World gross domestic product (GDP) (Washington, 2012). The importance of this market triggered an increase in the use of e-services to ensure electronic B2B transactions. In this context, more and more companies are using Web services to achieve transactions with their partners and/or offer on-line services. For instance, in a Mckinsey Quarterly survey conducted on more than 2800 companies worldwide, 80% are using or planning to use Web services. Among these companies, 78% says that the Web services technology is among the three most important technologies to their business (The Mckinsey Quarterly, 2007).

These companies have to make their Web services available for consultation through their own private Web service registries. As a result, the number of Web service registries that are made available for use can be as many as the large number of companies. This raises an old, search engine, problem in a new form: discovery mechanisms of Web services are not efficient both in response times and quality of results (Sioutas et al., 2009). Basically, a company interested in a service has to screen several registries to discover the service that best suits its needs. This task can be very cumbersome since the number of available registries, and also the services they advertise can be very large (Bentahar et al., 2007). In this context, if appropriate solutions are not considered, "traditional" Web services discovery mechanisms that consist of scanning all the registries would for instance slow down the democratization of Web services.

In our research, we are interested in Web services discovery in a distributed registry environment (Sellami et al., 2010c). As the number of registries can be very large, registry discovery acts as an initial filter to detect the adequate registries to a requester's query and thus reduce his/her search space. Indeed, to smooth the progress of the Web service registry discovery step, several approaches dealing with distributed registries (e.g. Sivashanmugam et al., 2004; Xu and Chen, 2007; Sellami et al., 2010c) have already proposed to structure their registries network into several groups according to business domain, functional, or non-functional properties of the services they advertise. To deal with this problem and to address the large number of Web service registries and their poorly organized network, we propose to organize Web service registries into communities according to the functionalities of the services they advertise.

1.1. Overview and motivations

In our work, we use communities as a means for a functionality-driven organization of a distributed Web service registry environment. The Oxford dictionary defines a community as "*a group of people living together in one place holding certain attitudes and interests in common*". In the Web services research field, Benatallah et al. (2003) define a Web service community as

* Corresponding author at: Telecom & Management SudParis, Computer Science Department, 9 rue Charles Fourier, 91011 Evry Cedex, France.
Tel.: +33 01 60 76 45 57; fax: +33 01 60 76 47 80.
E-mail addresses: mohamed.sellami@it-sudparis.eu (M. Sellami), olfa.bouchaala@it-sudparis.eu (O. Bouchaala), walid.gaaloul@it-sudparis.eu (W. Gaaloul), samir.tata@it-sudparis.eu (S. Tata).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.11.019

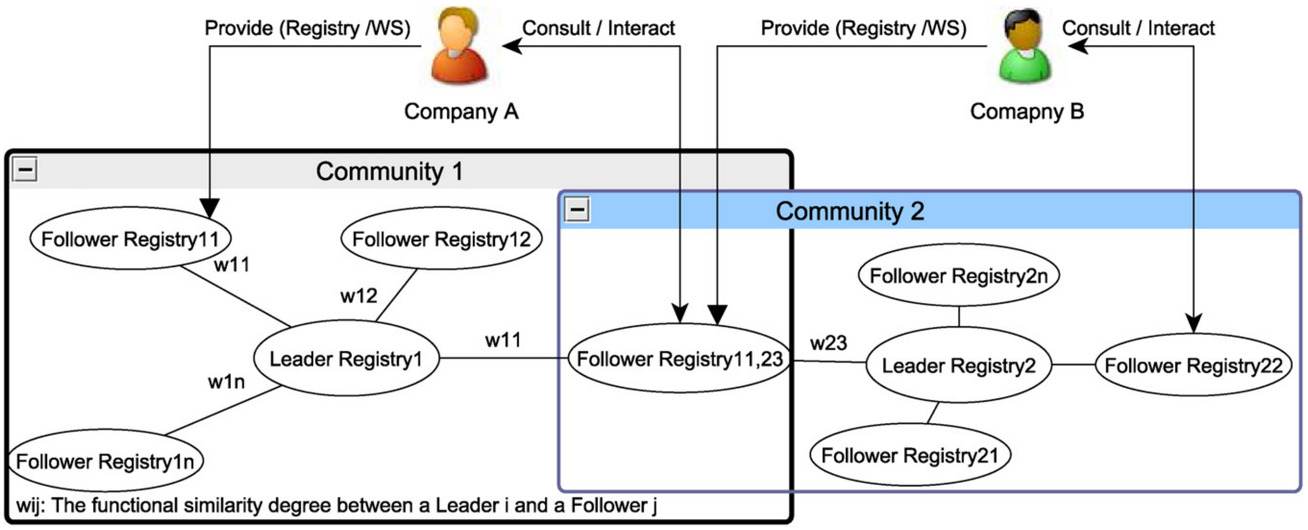


Fig. 1. Architecture of communities.

"*a collection of Web services with a common functionality although different non-functional properties*". Maamar et al. (2009) consider a community as "*a means for providing a common description of a desired functionality without explicitly referring to any concrete Web service that will implement this functionality at run-time*".

So, a distributed registry network can be virtually structured into communities and each registry belongs to at least one community with a certain extent. In our approach, a Web service registry community is defined as a set of registries offering Web services providing similar functionalities. For each registry, a set of membership degrees indicating its membership to the different communities will be assigned. In each community, one registry has the role of *leader* and the other members the role of *followers* (see Fig. 1). The *leader* registry is the most representative registry of the community functionality. Therefore, the *leader* plays a prominent role in managing its community and its members. In the same way, a *leader* or a *follower* for a community c_1 could be a *follower* of another community c_2.

To build communities of registries, we propose in this work to associate to each registry a signature, that we call Web Service Registry Description (WSRD) (Sellami et al., 2010a), reflecting its functionalities. These descriptions are the basis for building the communities of registry. To build a network of communities of Web service registries, we propose in this work to use a clustering technique rather than a classification. We also provide the management operations needed to guarantee the consistency of communities during their life-cycles. Below, we detail the technical motivations for our approach by answering these two main questions: *Why do we use a functionality driven organization?* and *Why do we need to manage communities after their creation?*

Functionality-driven organization: In a Web service discovery process, a service requester is usually interested by a functionality that a service can offer. Therefore, it is more obvious and appropriate to organize Web service registries based on their offered functionalities.[1] By using a service requester's query, representing the needed functionality, we can filter and reduce a service requester's search space to the community of registries advertising Web services offering the needed functionalities.

Need of management: Communities and Web service registries operate within a very dynamic environment where changes are frequent. In fact, a new Web service description can be published in a registry and others can be unpublished at any time. In the same way, a registry can join a community or leave it according to its convenience. However, we cannot re-run our community clustering approach when changes occur mainly for cost reasons. We should only use our communities building approach as a "cold starter" for a registry network organization and so we have to define management operations to handle the dynamic aspects of communities.

1.2. Roadmap of our approach

The main steps of our approach are summarized as follows:

1. Step 1: We characterize each Web service registry with a semantic WSRD description. This description is based on the descriptions of the Web services belonging to the considered registry and "semantically aggregate" its Web service functionalities. The registry description computing process that we propose is automatic and does not ask for any explicit knowledge from a registry provider. A registry description is implicitly created using as only input Web service descriptions of that registry.
2. Step 2: We use WSRD registry descriptions to build communities of registries. In fact, using the WSRD descriptions allows us to group into communities different Web service registries according to their offered functionalities.
3. Step 3: To handle the dynamic nature of communities and their members (i.e. Web service registries), we define the needed management mechanisms to monitor changes and reconcile potential conflicts. We identify the different capabilities of a registry (joining a community, updating functionalities, etc.) and a community (creation, dismantling, merging, etc.) life-cycle and we specify the associated management operations.

Preliminary parts of the community construction step are introduced in Sellami et al. (2010a), Sellami et al. (2010b), Sellami et al. (2011) and of the management step in Bouchaala et al. (2012). Nevertheless, in this paper we present the complete Web service registries construction process (including its associated management operations) promoting a complete, independent and dynamic approach for Web service registries organization.

The rest of the paper is structured as follows: Section 2 presents our registry description model (WSRD) and its computing process (step 1 of our approach). In Section 3, we first present a model for

[1] Functionalities of a Web service registry and functionalities of the Web services it advertise are interchangeably used.

formally representing a Web service registry, a community and a community network. Then we introduce the second step of our approach and we show how we use the WSRD descriptions to organize a registry network as communities. In Section 4 we introduce the management algorithms and operations for the registry and community life cycles that are used in the third step of our approach. The implementation, experimentation and usability of our approach are shown in Section 5. Section 6 discusses related work and Section 7 concludes the paper with an outlook to future work.

2. Step 1: description of registries

Our idea to describe a Web service registry consists in aggregating the WSDL descriptions of the Web services it advertises into one description called Web Service Registry Description (WSRD for short) (Sellami et al., 2010a). Hence a WSRD of a registry can give an overview of the functional properties of its Web services.

In Fig. 2, we introduce a graph representation of a WSRD description. Since a WSRD registry description is based on Web service descriptions published in that registry, the different nodes composing this graph are inspired by the WSDL format. As we provide a description of a registry (not of a service), we are only interested in the abstract section of the Web services description. To provide the semantic WSRD model, semantic descriptions of Web services are the only input that we use. In this work, we choose to use semantic Web service descriptions written in SAWSDL (Lausen and Farrell, 2007). Other semantic languages, such as OWL-S (Martin et al., 2004), WSMO (Roman et al., 2005) or YASA (Chabeb et al., 2010) can be adopted.

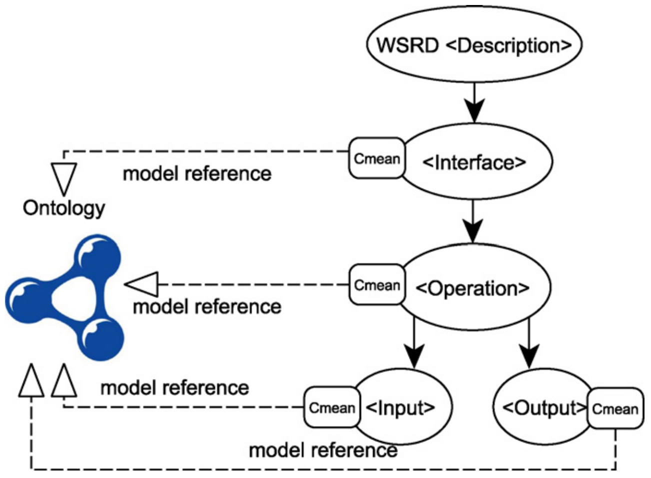


Fig. 2. The WSRD model.

WSRD defines a registry using the following WSDL elements: `interface`, `operation`, `Input` and `Output`. These elements give an abstract description of the mean functionalities offered by the Web services of a registry. We associate each WSRD element with a concept taken from a semantic model (domain ontology (DO)) using the SAWSDL modelReference extension attribute. In our work, we suppose that service requesters share the same semantic stack. This is done through common ontologies or ontology mediation mechanisms if different ones are used. And since we are dealing with private registries belonging to a specific company, we assume that all the services advertised by a registry are homogenous in term of

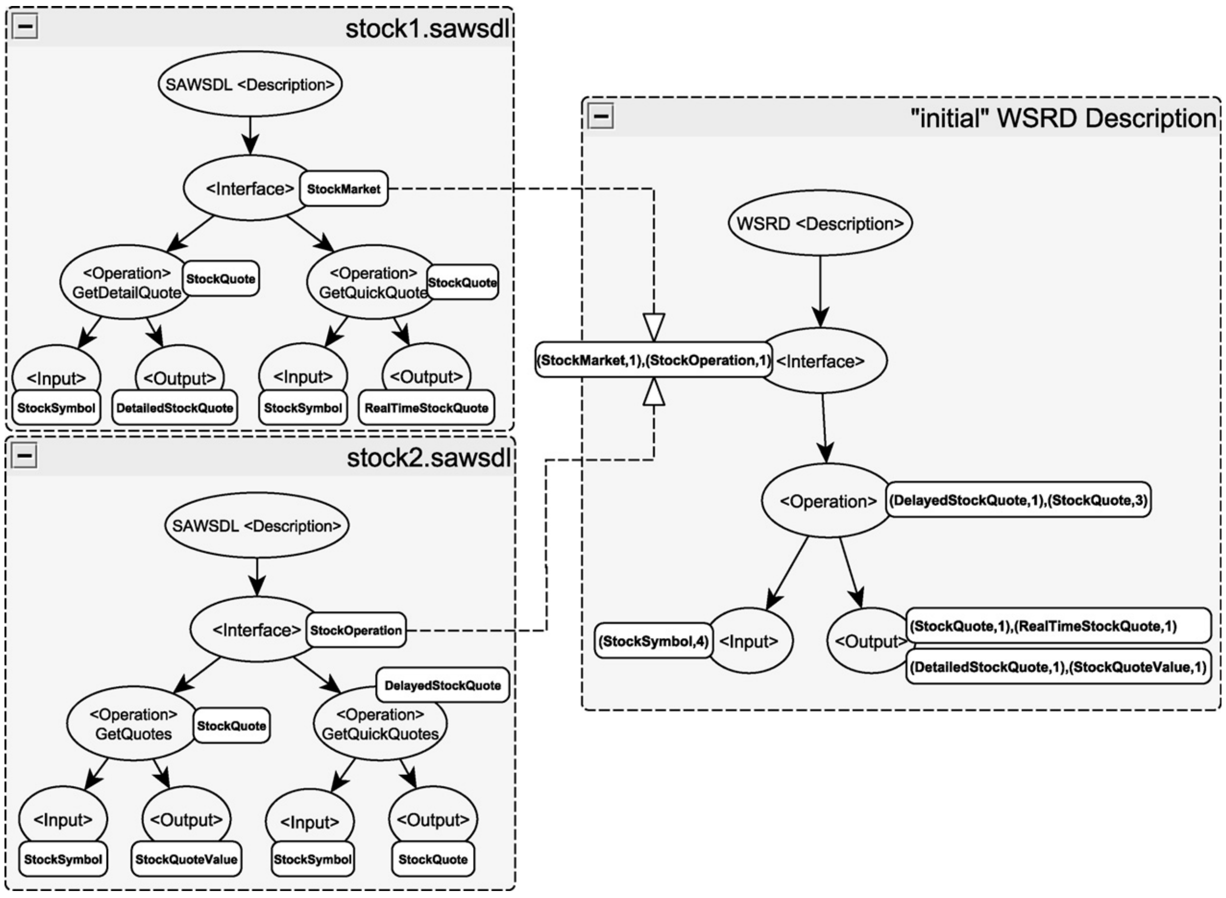


Fig. 3. Concept extraction.

their business domain and semantics. In addition, since those services have the same provider, we suppose that they use the same semantic description language.

Computing a registry's WSRD description goes through three steps:

- Step 1.1: We first extract the annotating concepts and the number of times they occur from the Web service descriptions published in the registry (Section 2.1).
- Step 1.2: We compute the groups of potential concepts, taken from the DO, to annotate the WSRD description (Section 2.2). To each concept we associate a value indicating its similarity degree to the whole set of extracted concepts.
- Step 1.3: Finally, we reduce the computed concepts' groups to only keep the concept(s) that will be used to annotate a registry's WSRD description (Section 2.3).

Compared to our initial WSRD computing approach introduced in Sellami et al. (2010a), these three steps are re-defined and we provide a formal definition of a WSRD description. This modification provides strong formal basis allowing a smoothly integration to our whole approach. We also added an example using two real world stock quotes Web services descriptions presented in the METEOR-S project to illustrate the different steps of the WSRD description computing process. The addition of this real world example enhances the paper understanding and shows that our approach is feasible in concrete use cases.

2.1. Step 1.1: extracting the annotating concepts

The first step in a WSRD computing process is to extract the DO concepts annotating the different SAWSDL descriptions elements (i.e. `<interface>`, `<Operation>`, `<Input>` and `<Output>`) in the Web service registry. These concepts, as well as their number of occurrence constitute the "initial" WSRD description (Definition 2.1).

Definition 2.1 *("Initial" WSRD).* We define an "initial" WSRD as a quadruple (I, O, In, Out) of *hash maps*.[2] We call I (respectively O, In and Out) the *hash map* containing the extracted concepts of `<interface>` (respectively `<operation>`, `<input>` and `<output>`). These hash maps associate a value nb_i to a concept C_i where:

- C_i is the extracted annotating concept of an element (i.e. `<interface>`, `<Operation>`, `<Input>` or `<Output>`) of a SAWSDL Web service description.
- nb_i is the number of times the concept C_i was found in the corresponding description element of the Web service descriptions of the registry.

We rely on two real world stock quotes semantic Web services descriptions presented in the METEOR-S project[3] to provide an example for the concept extraction step (see Fig. 3). Without loss of generality, we considered a registry advertising only these two Web services and a DO[4] formed by the 8 concepts annotating the services descriptions. From those two descriptions, we extract the annotating concepts of the different description elements. These concepts and the number of times they were identified will be stored in the corresponding *hash maps* of the "initial" WSRD (I, O, In, Out) such that:

- $I = \{ <StockMarket, 1>, <StockOperation, 1> \}$
- $O = \{ <DelayedStockQuote, 1>, <StockQuote, 3> \}$
- $In = \{ <StockSymbol, 4> \}$
- $Out = \{ <StockQuote, 1>, <RealTimeStockQuote, 1>, <DetailedStockQuote, 1>, <StockQuoteValue, 1> \}$

2.2. Step 1.2: computing groups of potential concepts

In this second step, we create a group of weighted concepts for each element of a WSRD description (i.e. `<interface>`, `<Operation>`, `<Input>` and `<Output>`). These groups contain all the semantic concepts of the used DO that we consider as candidate concepts for annotating a WSRD element. Each of these concepts is associated to a value s reflecting its similarity to the concepts extracted in step 1.1.

A value s_j, associated to a concept C_j in a group of weighted concepts, is computed using the couples (C_i,nb_i), stored in the "initial" WSRD, and the similarity degrees between the different semantic concepts (see formula (1)). The formula, we propose, computes the average of the similarity factors ($Similarity[C_j, C_i]$) of C_j to the set of semantic concepts C_i extracted in step 1.1 weighted by nb_i (i.e. the number of times the concept C_i was identified). We use nb_i to weight the similarity factors since a concept identified many times is semantically more significant than a concept identified few times. A non-identified concept is weighted by zero. Based on these data, we create the "intermediate" WSRD (Definition 2.2).

Definition 2.2 *("Intermediate" WSRD).* An "intermediate" WSRD is a quadruple (H_I, H_O, H_{In}, H_{Out}). We call H_I (respectively H_O,H_{In} and H_{Out}) the hash map used to store the set of potential C_{mean} for the `<interface>` (respectively `<operation>`, `<input>` and `<output>`) element. These hash maps will contain the different concepts C_j of the used ontology mapped to a value s_j representing the sum of the similarity factors between C_j and the different concepts associated to an element of the "initial" WSRD. H_I (respectively H_O, H_{In} and H_{Out}) is computed using the following formula:

$$H_I[C_j] = s_j = \frac{\sum_{i=1}^{t}(I[C_i] \times Similarity[C_j, C_i])}{\sum_{i=1}^{t} I[C_i]}, \quad \text{for } j = 1, \ldots, t \tag{1}$$

where: t is the number of concepts in the used ontology. C_j, $j = 1, \ldots, t$: are the concepts of that used ontology. I (respectively O, In and Out): a hash map representing the extracted concepts of the `<Interface>` (respectively `<operation>`, `<input>` and `<output>`) element in step 1.1. Similarity: a matrix containing the similarity factors between all the concepts of the used ontology.

In order to compute the similarity matrix ($Similarity[]$), we are using the enhanced edge counting similarity measure proposed by Resnik (1995) as it is simple and widely used. This measure computes the similarity between two concepts based on the number of edges found on the path between them. We are aware of the impact of the choice of the similarity computing method on our approach and we are studying this issue by testing other similarity methods (Jiang and Conrath, 1997; Lin, 1998). To illustrate the groups of potential concepts computing step, we pick up again the example

[2] A hash map is a data structure mapping some identifiers or keys to some associated values.
[3] The METEOR-S project at the LSDIS Lab, University of Georgia aims to semantically extend Web services related standards (http://lsdis.cs.uga.edu/projects/meteor-s/).
[4] The OWL description of the used domain ontology can be found at http://www-inf.int-evry.fr/SIMBAD/tools/ontology/.

of the previous step (Fig. 3). The used Similarity matrix (Eq. 2) is shown below:

$$Similarity[] = \begin{pmatrix} 1 & 0.82 & 0.76 & 0.66 & 0.66 & 0.66 & 0.66 & 0.66 \\ 0.82 & 1 & 0.81 & 0.76 & 0.76 & 0.76 & 0.76 & 0.76 \\ 0.76 & 0.81 & 1 & 0.76 & 0.76 & 0.76 & 0.76 & 0.76 \\ 0.66 & 0.76 & 0.76 & 1 & 0.26 & 0.26 & 0.26 & 0.26 \\ 0.66 & 0.76 & 0.76 & 0.26 & 1 & 0.26 & 0.26 & 0.26 \\ 0.66 & 0.76 & 0.76 & 0.26 & 0.26 & 1 & 0.26 & 0.26 \\ 0.66 & 0.76 & 0.76 & 0.26 & 0.26 & 0.26 & 1 & 0.26 \\ 0.66 & 0.76 & 0.76 & 0.26 & 0.26 & 0.26 & 0.26 & 1 \end{pmatrix} \qquad (2)$$

Now, we compute the concepts' weights for each node of the WSRD description. According to Fig. 3, the `<Operation>` element in the "initial" WSRD is: $O = \{ < DelayedStockQuote, 1 >, < StockQuote, 3 > \}$. H_O is computed using formula (1) as follows:

$$H_O[StockMarket] = \frac{\sum_{i=1}^{8} O[C_i] \times Similarity[i,1]}{\sum_{i=1}^{8} O[C_i]} = 0.735$$

We apply the same formula on $H_O[C_j]$ for $j = 2, \ldots, 8$ and we obtain the following groups of concepts' weights for the `<Operation>` WSRD element: H_O ={<*StockMarket*, 0.735>, <*StockOperation*, 0.772>, <*StockQuote*, 0.94>, <*StockQuoteValue*, 0.635>, <*DelayedStockQuote*, 0.82>, <*StockSymbol*, 0.635>, <*RealTimeStockQuote*, 0.635>, <*DetailedStockQuote*, 0.635>}

By the same way, we compute the other concepts' weights related to the `<Interface>`, `<Input>` and `<Output>` elements.

2.3. Step 1.3: reducing the concepts

In the third step, we aim at selecting the median concept(s) C_{mean} (from the group of weighted concepts of step 1.2) which are the most similar to the ones identified in step 1.1 (stored in the "initial" WSRD). The selected C_{mean} will be used to annotate the corresponding WSRD element. To each selected C_{mean} we associate a weight indicating its relevance in the annotated WSRD element vis-à-vis the other C_{mean}. The resulting WSRD is called the "final" WSRD and is defined in Definition 2.3.

This reduction step can be achieved in two different ways: strong or weak.

- **Strong reduction**: This method consists in choosing a unique C_{mean} to annotate the associated WSRD element. A simple way of realizing strong reduction is to choose from its representative hash map H_e ={<C_i, s_i>} the concept C_i having the highest value s_i. The chosen concept corresponds to the mean concept of the WSRD element.
- **Weak reduction**: By employing weak reduction a WSRD element will be annotated using more than one C_{mean}. The issues here are to define which and how many concepts to choose. In our work, we choose the C_{mean} using Algorithm 1. Following this algorithm, we first select from the groups of potential C_{mean} (H_e) the concept having the highest value $s = s_{start}$ as a C_{mean} (lines 1–4). Then, we compute the absolute deviation σ between the values s of H_e (line 5). Then we choose another concept $C_{current}$ as C_{mean} such that its weight $s_{current}$ is less or equal than s (the weight of the previous chosen C_{mean} (line 6)) minus σ (line 10). This condition (the gap between the values of the chosen concepts is largest than the absolute deviation between them) allows us to avoid choosing two semantically similar concepts as C_{mean}. We then assign to s the value of the chosen concept and repeat the previous steps (lines 7–12) until $s - \sigma \leq 0$. Algorithm 1 allows us to efficiently choose the WSRD annotating C_{mean} by eliminating the less representative concepts while ensuring that no lost of knowledge happened.

Algorithm 1 *(WeakReduction(H_e))*.

```
Require: $H_e$        # The group of potential $C_{mean}$ of a WSRD element $e$ (i.e. $e$=interface, operation, input or output).
Ensure: $CM$          # Hash map of the weighted selected $C_{mean}$

 1: $H_e = Order(H_e)$                # Order: a function sorting in descending order the element $C_i$ of $H_e$ according to $s_i$
 2: $C_{start} = H_e.getElement()$
 3: $s_{start}=H_e[C_{start}]$;
 4: $CM$.add(($C_{start}, s_{start}$));   # Select the concept having the highest value as $C_{mean}$
 5: $\sigma$ =deviation($s_i$) ;
 6: $s = s_{start}$;                  # $s$ is the weight associated to the last chosen $C_{mean}$
 7: while ($s - \sigma \geq 0$) do    # Find in $H_e$ the concept having the highest value and distanced from $s$ by $\sigma$
 8:     $C_{current} = H_e.getElement()$
 9:     $s_{current} = H_e[C_{current}]$
10:     if ($s_{current} \leq s - \sigma$) then
11:         $CM$.add(($C_{current}, s_{current}$));
12:         $s = s_{current}$;
13:     end if
14: end while
15: return $CM$
```

Definition 2.3 *("Final" WSRD)*. The "final" WSRD is a quadruple ($CM_{H_I}, CM_{H_O}, CM_{H_{In}}, CM_{H_{Out}}$). CM_{H_I} (respectively $CM_{H_O}, CM_{H_{In}}, CM_{H_{Out}}$) is a hash map containing the C_{mean} annotating the WSRD `<interface>` (respectively `<operation>`, `<input>` and `<output>`) element associated with the value(s) $s = H_I[C_{mean}]$ (respectively $H_O[C_{mean}]$, $H_{In}[C_{mean}]$ and $H_{Out}[C_{mean}]$) indicating its representativeness.

Strong reduction is a simple technique for concepts reduction. However, weak reduction better reflects the functionalities offered by the Web services of a registry. We recommend using weak reduction as the computation load is almost the same for both techniques (Sellami et al., 2010a).

To illustrate the reduction step, we pick up again our example. In step 1.2, we computed the four groups of potential C_{mean}, namely H_I, H_O, H_{In} and H_{Out}. To select the best representative concepts for a WSRD element, we apply our reduction process:

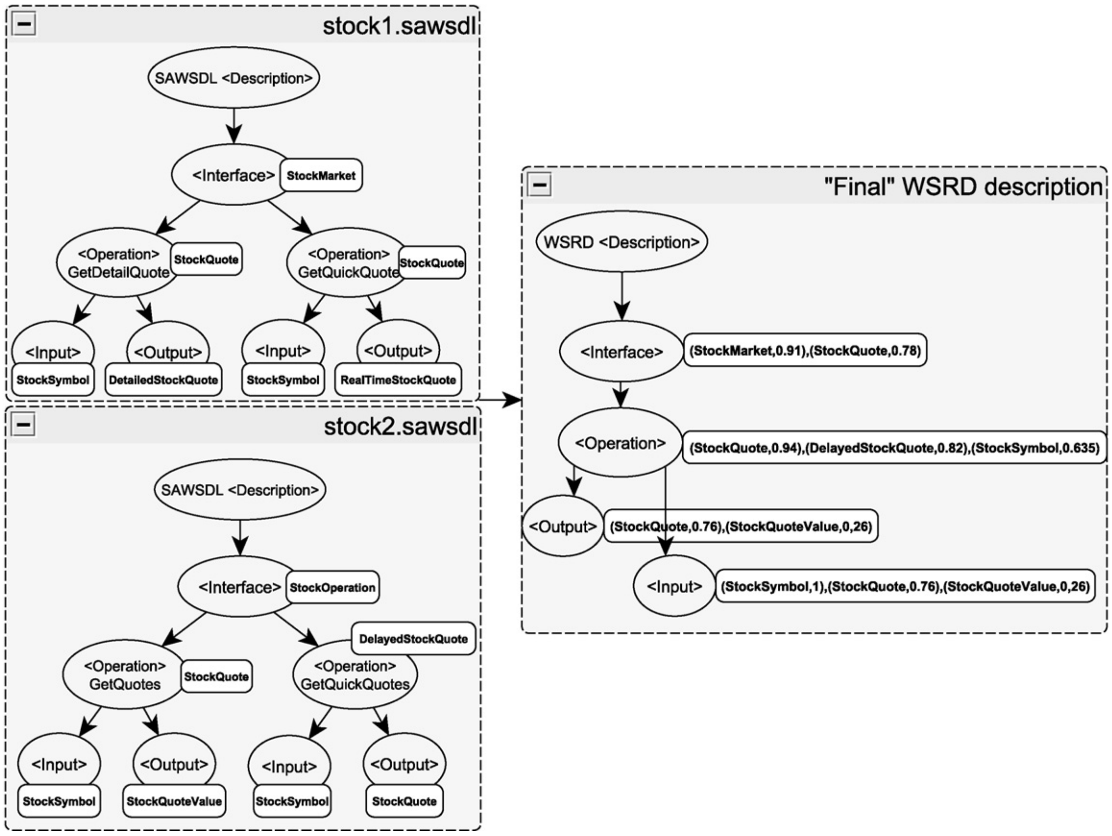


Fig. 4. A resulting WSRD graph using weak reduction.

- Using strong reduction, we choose from every group of potential C_{mean} the concept having the highest value s_i as the C_{mean}. In this way, *StockQuote* will be the mean concept of H_O.
- Using weak reduction, we associate with each WSRD element one or more C_{mean}. We start by applying our proposed weak reduction algorithm (Algorithm 1) on H_O. The first selected C_{mean} is <*StockQuote*, 0.94> as it presents the highest value. The absolute deviation of the s_i values of H_O is $\sigma = 0.11$ and $s_{start} = 0.94$. The next selected C_{mean} from H_O is the concept having the highest s_i value such that $s_i \le s_{start} - \sigma = 0.94 - 0.11 = 0.83$. The selected concept is <*DelayedStockQuote*, 0.82> and s will be equal to 0.82. By repeating the last step until $s - \sigma \le 0$, we obtain $C_{mean}(H_O) = \{<StockQuote, 0.94>, <DelayedStockQuote, 0.82>, <StockSymbol, 0.635>\}$. By executing this algorithm on the other groups of potential C_{mean}, we obtain the sets of potential C_{mean} for the other WSRD elements as shown in Fig. 4.

This description reflects the average functionality of both stock quotes Web services descriptions. For instance, the input element of this WSRD description is annotated by the semantic concept *StockSymbol* weighted by 1 and thus indicating that all the input elements the registry's services are annotated by *StockSymbol*. The WSRD input element is also annotated by <*StockQuote*, 0.76> and <*StockQuoteValue*, 0.26> indicating their semantic similarity to the concept annotating the registry's services input elements. This is due to the fact that *StockQuote* subsumes the *StockSymbol* concept and *StockQuoteValue* has the same parent concept as *StockSymbol* in the used DO.

3. Step 2: building of communities

In this section, we present the second step of our approach: based on WSRD descriptions computed in the first step, we organize a registry network into communities. We start by briefly introducing graph prerequisites and some of the special types of graphs playing prominent role in our work in Section 3.1. Then we present our model for representing a Web service registry, a community and a community network in Section 3.2. Finally, we introduce our communities building approach in Section 3.3.

3.1. Background on graph theory

We define our distributed registry network based on the notations and concepts offered by graph theory. Indeed, graphs are highly flexible models for analyzing a wide range of practical problems, especially networks, through a collection of nodes and connections between them. A network is then formalized with a graph G, defined as a pair of sets $G = (V, E)$. V is the set of vertices (or nodes) and E is the edge set representing the network connections. Also, a graph can be weighted by a weight function $w : E \to \mathbb{R}$ assigning a weight on each edge. A weighted graph is then denoted $G = (V, E, w)$. Throughout the rest of the paper, we use these notations. In the following, we introduce the main concepts used in this paper:

- Adjacency matrix: The adjacency matrix of a graph G with n vertices is a $n \times n$ matrix $A_G = (a_{i,j})$ where $a_{i,j}$ is equal to 1 if $(i, j) \in E$ and 0 otherwise. In a weighted graph, the adjacency matrix can be used to store the weights of the edges (McConnell, 2008).
- Bipartite graph: A graph G is called bipartite (Bondy and Murty, 2007) if its vertex set can be divided into two subsets X and Y such that every edge joins a vertex in X to another in Y.
- Complete bipartite graph: A bipartite graph is complete if every vertex in X is joined to every vertex in Y. This is denoted $K_{n,m}$ with n, m respectively the cardinality of X and Y.
- Star graph: A star graph is a complete bipartite graph $K_{1,k}$ or $K_{k,1}$.

- Complement of a graph: The complement of a graph $G=(V, E)$ is a graph $\overline{G}=(V, V\times V\backslash E)$ with the same vertices as G but with only edges that are not in G.

3.2. Modeling of communities

Modeling a Web service registry: In this work, we refer to each WSRD description of a Web service registry by *f*. A registry can belong to different communities at the same time. Thus, we assign to a registry a set of membership degrees that we call *MEM*. This set contains its membership degrees to each community in the network of communities (see Definition 3.5).

Definition 3.5 *(Web service Registry).* A registry is defined as a triple $r=(id, f, MEM)$ where:

- *id* is the registry identifier,
- *f* is a vector representing the average functionality offered by the advertised Web services within the registry,
- *MEM* represents the registry membership degrees to the different communities in the network. It is represented by a binary relation defined as follows. $MEM=\{(c, d)|c\in C, d\in[0, 1]\}$ where:
  - *C* is the community set,
  - *d* is the membership degree of the registry *r* to the community *c*.

We define the domain and range of $MEM\subseteq C\times[0, 1]$ as:

- $dom(MEM)=\{c|(c, d)\in MEM$ for some $d\in[0, 1]\}$
- $ran(MEM)=\{d|(c, d)\in MEM$ for some $c\in C\}$

We also define the function U_j that computes the degrees of membership of a registry r_i to a community c_j as follows:

- $U_j(r_i)=d\Rightarrow(d, c_j)\in r_i.MEM$

Modeling a community: A community is mainly characterized by its mean functionality *f* which represents the aggregation of the community registries functionalities. As we said in Section 1.1, we distinguish two kinds of registries (*leader* and *follower*) based on their role inside a community. Therefore, the set of community members (nodes) can be divided into a singleton $L=\{l\}$ representing the *leader* and a set $Fl=\{fl_i|i:1, \ldots, n\}$ where *n* is the number of the community *followers*. Thus, the community nodes are modeled as a star graph *G* where nodes are registries and each edge represents the functional similarity between the *leader* and a *follower* fl, $fl\in Fl$. The similarity between the functionalities offered by the *leader* and a *follower* can be computed using the cosine function (see Section 3.3, formula (3)). A definition of a community of registries is given in Definition 3.6.

Definition 3.6 *(Community of Registries).* A community is a triple $c=(id, f, G)$ where:

- *id* is the community identifier,
- *f* is a vector representing the mean functionality of the community c,
- $G=(L\cup Fl, E, w)$ is an undirected weighted star graph where:
  - *L* is the community *leader* (the registry having the highest membership degree inside the community c),
  - *Fl* is the set of community *followers* such as $L\cap Fl=\{\}$,
  - $E\subseteq L\times Fl$ is the set of edges,
  - $w: E\rightarrow[0, 1]$ is a weighting function, each weight represents the functional similarity between nodes.

Modeling the community network: So far, our distributed registry environment which is a set of communities is modeled by a set of star graphs. As the number of registries (nodes) can be very large and a single registry can belong to many communities, the community management is a cumbersome task. To deal with this problem and to have a global view of the network, we define another graph *CG*, called *Community Graph*, in which nodes represent communities and edges are the relationships between them. If two communities have at least one registry in common, then there is an edge joining them. In this case, we compute the distance between the vectors *f* of these communities. This distance represents the weight of the edge relating these two communities and can be computed using formula (4). We present the model of our community network in Definition 3.7.

Definition 3.7 *(Community network).* The community network is represented by an undirected weighted graph $CG=(C, E, w)$

- *C* is a finite set of nodes. Each single node represents a community of registries,
- $E\subseteq C\times C$ is the set of edges (representing the relationships between communities),
- $w: E\rightarrow[0, 1]$ is a weighting function representing the distance between two given nodes.

3.3. Organizing a registry network as communities

A community of Web service registries will bring together registries offering similar functionalities. Since a Web service registry generally offer services proposing different functionalities, it is difficult to properly define in advance classes categorizing the functionalities of the different registries. To organize Web service registries into communities, we use a clustering technique (where the different communities will be deduced from the registry descriptions) rather than a classification technique (where the different communities have to be defined in advance). When using a dynamic clustering technique, the different clusters (i.e. the communities of Web service registries) will be identified from the given input data (i.e. the WSRD descriptions) and each data point (i.e. Web service registry) will be assigned to one community. We already promoted the idea of using functionalities to organize Web service registries in previous work (Sellami et al., 2010b, 2011). However, in Sellami et al. (2010b) the necessary clustering technique was not concretely defined and in both papers the dynamic aspect of our environment was not considered. In this paper, we define a clustering technique to build a network of communities and the needed management operations to guarantee the evolution of this organization. This clustering technique, which was sketchily introduced in Sellami et al. (2011), is better developed, detailed and especially integrated to the whole approach in the current paper.

Since a registry can belong to different communities at the same time, the use of an exclusive clustering is inadequate for building communities of registries. In exclusive clustering, data are grouped exclusively. Thus, if a certain datum belongs to one cluster then it is automatically excluded from the others. Therefore we propose to use an overlapping clustering method, also known as fuzzy clustering (Zadeh, 1965), to organize our distributed registries into communities. In the following, we present our fuzzy clustering approach to organize our distributed registries into communities which was inspired from the fuzzy C-means method (Dunn, 1973; Bezdek, 1981). The fuzzy C-means is a method of unsupervised clustering often used in the fields of data analysis and pattern recognition. In several works (Jursic and Lavrac, 2008; Saraçoğlu et al., 2007), this method is employed for document clustering. A document contains terms and is represented by a vector which dimensions are the document's terms. By using fuzzy C-means to cluster a set of documents, each document belongs to two or more clusters. Concretely, we proceed in three steps to build communities of registries:

1. Step 2.1: WSRD descriptions are processed to map to the term-document structure (see Section 3.3.1).
2. Step 2.2: we propose a method to measure the distance between two registry vectors as needed by the fuzzy C-means algorithm (see Section 3.3.2).
3. Step 2.3: using the results of the two previous steps we apply a fuzzy clustering technique to build communities of registries (see Section 3.3.3).

3.3.1. Step 2.1: defining the registry vector space

In order to apply the fuzzy C-means method to our context, we consider each *registry as a document* and the set of its *ontology annotating concepts as terms* in the document. The data points are Web service registries r_i represented by their WSRD descriptions. We use the vector space model (Salton et al., 1975) to represent these descriptions as vectors. Each WSRD description will be represented by a vector $r_i \cdot f = [w_1, w_2, \ldots, w_t]$ where t is the number of concepts in the ontology. The weights of the different w_i are computed as follows:

1. For the `<interface>` (respectively `<Operation>`, `<Input>` and `<Output>`) element we extract the associated weight (see Section 2) with each C_{mean} and we store these values in a vector $v_I = [e_1, e_2, \ldots, e_t]$ (respectively v_O, v_{In} and v_{Output}). These vectors have the same size t as the vector $r_i \cdot f$. If a concept from the common ontology does not occur in the WSRD description, its value in the vectors is zero.
2. Since v_I, v_O, v_{In} and v_{Out} have different meanings, they should be associated with weights. We define four weights: α for v_I, β for v_O, δ for v_{In} and λ for v_{Out} such that $\alpha + \beta + \delta + \lambda = 1$.
3. Finally, a registry WSRD vector $r_i \cdot f$ is computed as follows: $r_i \cdot f = \alpha \times v_I + \beta \times v_O + \delta \times v_{In} + \lambda \times v_{Out}$

The coefficients α, β, δ and λ depend on the weights that the service consumer or provider want to give to the interface, the operation, the input or the output of a service. Bigger the coefficient is, more important the related element is considered for the WSRD computation. Normally, different value setting for α, β, δ and λ should impact the clustering result to some extent. Therefore service consumer and provider should agree on these coefficients based on the given environment. Indeed, if we have a capability centric service (Zaremba et al., 2011) then a greater weight should be given to the interface. Else, a greater weight should be given to the input/output if we have a data-centric service (Zhou et al., 2011).

To illustrate a WSRD description representation according to our vector space, we pick up the WSRD example of Fig. 4. $r_i \cdot f$ is computed as follows:

1. We first create the vectors describing the different elements of the WSRD description:

v_I	= [0.91	0	0.78	0	0	0	0	0]
v_O	= [0	0	0.94	0	0.82	0.635	0	0]
v_{In}	= [0	0	0.76	0.26	0	1	0	0]
v_{Out}	= [0	0	0.76	0.26	0	0	0	0]

2. To specify the different weights of these vectors, we consider that the WSRD `<interface>` and `<Operation>` elements have the same importance in representing a registry's functionality which is more important than the `<Input>` and `<Output>` elements. We suppose that $\alpha = \beta = 0.4 > \delta = \lambda = 0.1$.
3. Finally, the WSRD representing vector $r_i \cdot f$ is the weighted sum of the different WSRD elements vectors. $r_i \cdot f$= [0.364 0 0.84 0.052 0.328 0.354 0 0].

3.3.2. Step 2.2: computing registry distance

A distance measure is used to establish the degrees of membership of each data point to the different clusters. In our work, the set of data points to cluster (the Web service registries) are represented by vectors computed from the WSRD descriptions. To compute the distance between two vectors, we use the cosine similarity measure (Salton and Buckley, 1987) to establish the similarity between two vectors since it is adequate for high dimensional data (Qian et al., 2004). The cosine function ranges between 0 (no similarity between vectors) and 1 (identical vectors). Given two vectors $r_1 \cdot f$ and $r_2 \cdot f$ that represent the functionalities of two Web service registries, the cosine similarity is computed using formula (3):

$$cosine(r_1 \cdot f, r_2 \cdot f) = \frac{r_1 \cdot f \times r_2 \cdot f}{\|r_1 \cdot f\| \times \|r_2 \cdot f\|} \quad (3)$$

Since the cosine function leads to the similarity between two vectors, and not to a distance as needed for the fuzzy C-means algorithm, we use formula (4) to deduce the distance from the cosine similarity function:

$$distance(r_1 \cdot f, r_2 \cdot f) = 1 - cosine(r_1 \cdot f, r_2 \cdot f) \quad (4)$$

3.3.3. Step 2.3: building communities

To build communities of Web service registries, we use a clustering technique. We proceed in two repetitive phases: (i) computing the communities' centers also called centroids (represented by f the mean functionality of the community) and (ii) assigning the different registries to these centers. Each registry is assigned to a community with a various degree of membership. The total sum of the membership's values for a registry should be equal to 1.

The Web service registry clustering is based on the minimization of the following objective function (5):

$$J = \sum_{j=1}^{C} \sum_{i=1}^{N} [U_j(r_i)]^m distance(r_i \cdot f, c_j \cdot f)^2 \quad (5)$$

where C is the number of communities, N the number of registries to organize, U_j the membership function of the community j, r_i is the ith registry of our registries set, m (the fuzziness coefficient) is any real number greater than 1 and *distance* is our defined distance measure giving the similarity between a registry r_i and a community c_j. The clustering is carried out through an iterative optimization of the function J until it tends to be stabilized. The update of the membership $U_j(r_i)$ vectors and the community centroids $c_j \cdot f$ is done by respectively using formulas (6) and (7).

$$U_j(r_i) = \frac{1}{\sum_{k=1}^{C} (distance(r_i \cdot f, c_j \cdot f)/distance(r_i \cdot f, c_k \cdot f))^{2/(m-1)}} \quad (6)$$

$$c_j \cdot f = \frac{\sum_{i=1}^{N} U_j(r_i)^m \cdot r_i}{\sum_{i=1}^{N} U_j(r_i)^m} \quad (7)$$

After this step, we get C communities represented by their centroids. These centroids represent the mean functionality f of these communities in accordance with Definition 3.6. In addition, to each community c_j are associated N membership vectors $U_j(r_i)$, $i = 1, \ldots, N$ indicating the degrees of membership of the N registries to the community. These data allow us to infer the membership degrees *MEM* (see Definition 3.5) of the N registries to organize them into communities.

However, using this technique, the membership degrees of some registries to some communities may be very low. We thus define a threshold th for the membership degrees. If the membership degree

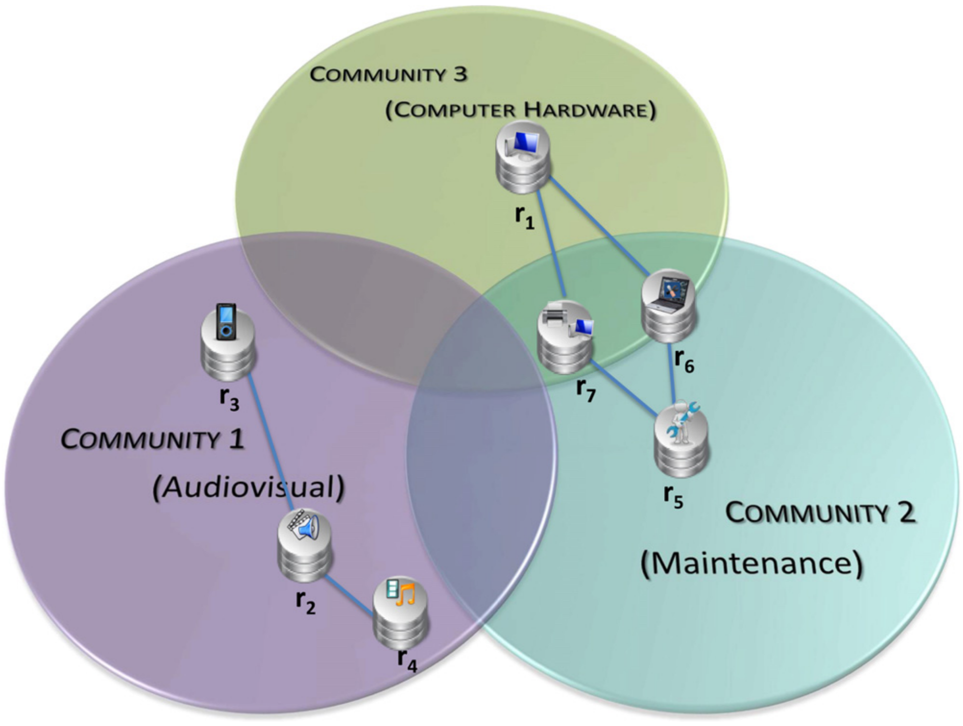


Fig. 5. Example of a registries network organization as communities.

of a registry to a community is below this threshold, it will not be considered as a member. This threshold is necessary to preserve the "reputation" of a community and to ensure that the functionalities of its members are not too different than those announced by the community. A complete scenario of building communities according to our approach is provided in Section 5.

4. Step 3: management of communities

Communities and Web service registries operate within a dynamic environment where changes are mainly initiated by service and registry providers. A service provider can publish or delete a Web service. Similarly, a registry provider can register its Web service registry or dismantle it at any moment. To keep the consistency of our communities' network against these events, management operations are needed (step 3 of our approach). In this aim, we use a community management approach that we introduced in previous work (Bouchaala et al., 2012). Nevertheless, in the current paper the management approach is used as a complement to our community clustering technique whereas in Bouchaala et al. (2012) this management approach was not integrated to the clustering approach. In addition, we added two algorithms to formally define the community creation and merging steps of the community life-cycle (see Section 4.3). Finally, we added a realistic example to illustrate the community life cycle in Section 4.1 and we use it to illustrate the necessary management operations to handle Web service registries (Section 4.2) and communities of Web service registries (Section 4.3) during their life-cycles.

4.1. Illustrative example

To illustrate our community management approach, we consider a registry network organized into communities according to our communities building approach. This network is formed by 7 registries organized into 3 communities as shown in Fig. 5.

The community 1 (respectively 2 and 3) contains registries offering services with functionalities around the audiovisual (respectively maintenance and computer hardware) domain. Definition 3.6 is used to represent the registries organization inside the communities in Table 1. For instance, r_6 and r_7 that offer services related to both computer hardware and maintenance domains (e.g. computer's screen reparation) are members of community 2 and community 3.

4.2. The registry life-cycle

Fig. 6 illustrates the communities and registries management process. This process does not include communities building as it is a "cold starter" step and executed once in our approach. A registry life-cycle can start when a registry provider decides to register its Web service registry in the community network (operation (1) in Fig. 6).

We use the example introduced in Section 4.1 to illustrate the possible operations that can occur when a company adds a new registry r_8. r_8 will join the communities network (operation (2)) and try to adhere to some existing communities according to the functionalities it advertises. The registry's WSRD is computed and two cases can be observed:

- r_8 offers services related to the maintenance domain and will join community 2 (operation (3)).
- r_8 offers services related to the shipping domain that is not represented by the existing communities and will create a new community to represent its functionalities (operation (10)).

Table 1
The communities of our example.

Community	id	f	G = (L ∪ Fl, E, w)
community1	c_1	f_1	$(r_2 \cup \{r_3, r_4\}, E_1, w)$
community2	c_2	f_2	$(r_5 \cup \{r_6, r_7\}, E_2, w)$
community3	c_3	f_3	$(r_1 \cup \{r_6, r_7\}, E_3, w)$

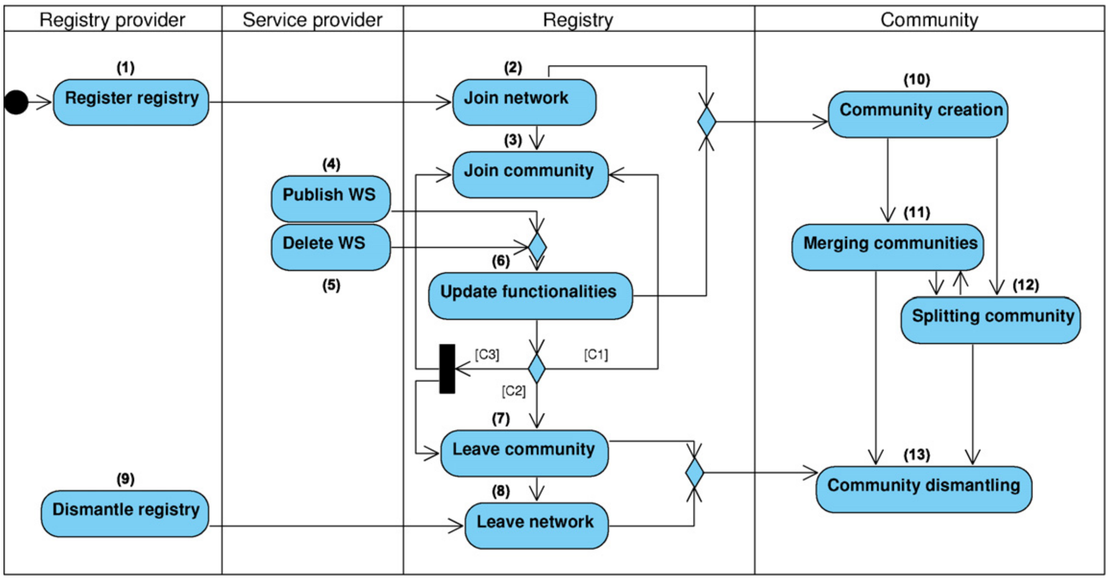


Fig. 6. Communities and registries management process.

In addition, service providers can publish (operation (4)) or delete (operation (5)) Web services within their registries thus leading to an update in the registry's mean functionality (operation (6)). In such scenario, the membership degrees of the updated registry to the existing communities can change and a suitability check of the registry memberships should be done. We provide three scenarios to illustrate some possible effects of an update in a registry's functionalities:

- The owner of the registry r_4 belonging to c_1, offering services related to the audiovisual domain, adds some services related to the maintenance domain (operation (4)). The registry's mean functionality will be updated (operation (6)) and r_4 can become a new member in c_2, offering services related to the maintenance domain,(operation (3)) and remains a member of c_1.
- The owner of the registry r_4 belonging to c_1 adds a single service related to the maintenance domain (operation (4)). The registry's mean functionality will be updated (operation (6)) but r_4 will stay a member in c_1.
- The owner of the registry r_4 belonging to c_1 adds some services related to the maintenance domain (operation (4)) and deletes all services related to the audiovisual domain (operation (5)). The registry's mean functionality will be updated (operation (6)), r_4 leaves c_1 (operation (7)) and become a new member in c_2 (operation (3)).

Finally, a registry can leave the whole network (operation (8)) if its provider decides to dismantle it (operation (9)). In the following, we only detail the operations (2), (3) and (6) of the registry life-cycle as the others are either trivial or does not trigger a change requiring management operations.

4.2.1. Joining the network

When a new Web service registry *r* joins our distributed registry environment, first its WSRD description is computed. After that, the registry will be guided to the adequate communities according to its set of membership degrees *MEM*. *MEM* is computed by the *CommunitySelection* algorithm (Algorithm 2). This algorithm takes as input the current registry's WSRD description. The membership degree is computed using Eq. (6). This algorithm outputs the set of membership *MEM* containing the membership degrees of the current registry to the different communities in the network. Taking into account that the membership degree must be greater than the community acceptance threshold `th` (line 4), the registry will be guided to the adequate communities (i.e. r will be added to the node set of the graph $c.G$ representing the community (line 5)). If all membership degrees are lower than `th`, a new community will be created (see Section 4.3.1).

Algorithm 2 *(CommunitySelection).*

```
Require: $r$ :registry
1: for each community $c_i \in C$ do
2:     $m \longleftarrow U_i(r)$                      # Compute the membership degree of $r$ to the communities $c_i$
3:     $r.MEM \longleftarrow r.MEM \cup \{(c_i, m)\}$
4:     if $m > \mathtt{th}$ then
5:         $V(c_i.G) \longleftarrow V(c_i.G) \cup \{r\}$      # Add $r$ to the adequate communities
6:     end if
7: end for
```

4.2.2. Joining a community

When a new registry joins a community, or after an update of the functionalities of an existent one, the registry can join some communities. However, when joining a community a registry may have a membership degree greater than the community *leader*. To handle this situation, we apply the *LeaderReselection* algorithm (Algorithm 3) to check whether or not a registry *r* will take the role of *leader* in one of the communities it joined. This is done by comparing the new registry's membership degree d_r to a community *c*, to d_l the membership degree of the *leader l* of the same community *c* (line 5). If the *leader*'s membership degree is still the greatest, the current registry will be a *follower* in this community (lines 6 and 7). Otherwise ($d_r > d_l$), we remove all the links between the *leader l* and the *followers* of that community (lines 9–11), add the *leader l* to the *followers* set (line 12), define the registry *r* as the new *leader*

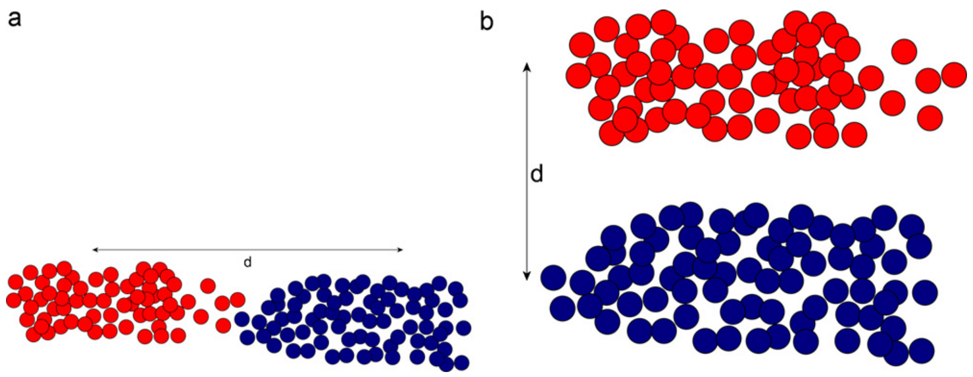


Fig. 7. Distance between clusters centers.

(line 13) and link the community *followers* to the new community *leader* (lines 14–16).

Algorithm 3 *(LeaderReselection).*

```
Require: r: registry
1: for all Communities $c \in dom(r.MEM)$ do
2:     $\{l\} \longleftarrow c.G.L$
3:     $d_r \in [0, 1]$ such that $(c, d_r) \in r.MEM$
4:     $d_l \in [0, 1]$ such that $(c, d_l) \in l.MEM$
5:     if $d_l > d_r$ then          # Compare the leader's membership degree ($d_l$) to the registry's membership degree ($d_r$)
6:         $c.G.Fl \longleftarrow c.G.Fl \cup \{r\}$          # Add $r$ to the followers list if $d_l > d_r$
7:         $E(c.G) \longleftarrow E(c.G) \cup \{(r, l)\}$
8:     else
9:         for all $fl \in c.G.Fl$ do
10:            $E(c.G) \longleftarrow E(c.G) - \{(l, fl)\}$
11:        end for
12:        $c.G.Fl \longleftarrow c.G.Fl \cup \{l\}$          # Add the leader to the followers list if $d_l < d_r$
13:        $c.G.L \longleftarrow \{r\}$          # Define $r$ as the new leader
14:        for all $fl \in c.G.Fl$ do
15:            $E(c.G) \longleftarrow E(c.G) \cup \{(r, fl)\}$
16:        end for
17:    end if
18: end for
```

4.2.3. Updating registry functionalities

We recall that the Web services advertised within a registry frequently change (new Web services arrive, others leave). Therefore, the registry functionalities have to be regularly updated. When a change occurs, the registry can stay in the same community, move from a community to another or create a new community. Therefore, its membership degree should be updated. If this degree is still lower to the community acceptance threshold *th*, the registry stay in the same community, otherwise it will be rejected.

After functionalities update of one or more registries in a community, the following events can happen:

- $E = E'$, i.e. no changes occur in the set of community members, where E and E' are two sets of members of a given community c, respectively before and after updating registries functionalities.
- $E \subset E'$, i.e. some new registries join the set of community members (Fig. 6[C1])
- $E' \subset E$, i.e. some registries leave the set of community members (Fig. 6[C2])
- $E \not\subseteq E' and E' \not\subseteq E$, i.e. some new registries join the set of community members and some others leave (Fig. 6[C3])

4.3. The community life-cycle

The main steps describing a community life-cycle revolve around community creation, dismantling, merging and splitting. We use the example introduced in Section 4.1 to illustrate these operations. For example, a new community creation (operation (10) in Fig. 6) can be triggered by:

- a new registry r_8 in the communities network offering functionalities that are not represented by c_1, c_2 and c_3 (e.g. services related to the shipping domain).
- the owner of the registry r_4 belonging to c_1, offering services related to the audiovisual domain, deletes all its proposed services and adds some new services related to the shipping domain. The registry's mean functionality will be updated, r_4 is no longer member of any existing community and a new community reflecting its proposed services has to be created.

Also, a community will be dismantled (operation (13)) if it becomes empty. This can happen for example when the companies providing r_1, r_6 and r_7 decide to only specialize in services related to maintenance domain and thus deleted their services related to the computer hardware domain. Throughout a registry life-cycle, we check the similarity inside and between communities to ensure the principle goal of clustering: minimizing the similarity between clusters while maximizing it within each cluster. To guarantee this goal, a community can be merged (operation (11)) to another one or split (operation (12)) into two communities. In the following, we detail these steps and we define their triggers, preconditions and effects.

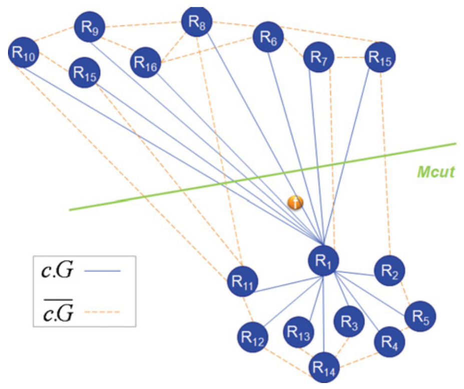


Fig. 8. Splitting a community using *Mcut* algorithm.

4.3.1. *Community creation*

When the membership degrees of a registry r to all the existing communities are lower than the threshold `th`, it provides necessarily a new functionality in the network. This situation can happen when a new registry joins the network or after an update of the registry's functionalities.

In this case, a new community $c_{new} = (id, f, G)$ is established automatically. The registry r that triggered the community creation, will get the role of *leader* for the new community c_{new}. The community mean functionality $c_{new} \cdot f$ will be the same as the functionality $r \cdot f$ proposed by the registry. Afterwards, some *followers* are recruited for the new community. To do so, the membership degrees of existing registries to the new community are computed. Every registry having a membership degree higher than the threshold th will be a member of c_{new}. The community creation as well as the necessary precondition and effect are displayed in Algorithm 4.

Algorithm 4 *(CommunityCreation).*

```
Require: $r$ : the registry that triggers the community creation
Ensure: $c_{new}$ : the new community
 1: precondition: $\forall d \in ran(r.MEM), d < \mathtt{th}$
 2: Community $c_{new}$;
 3: List followers;
 4: $c_{new}.f \longleftarrow r.f$          # The mean functionality of $c_{new}$ is the same as the functionalities offered by $r$
 5: $c_{new}.G.L \longleftarrow \{r\}$     # The leader of $c_{new}$ is the registry $r$
 6: for all Communities $c \in C$ do      # Recruit followers for the new community
 7:     for all Registries $r_c \in V(c.G)$ do
 8:         $m \longleftarrow U_{new}(r_c)$
 9:         $r_c.MEM \longleftarrow r.MEM \cup (c_{new}, m)$
10:         if $m > \mathtt{th}$ then
11:             $followers \longleftarrow followers \cup \{r_c\}$
12:             $E(c_{new}.G) \longleftarrow E(c_{new}.G) \cup \{(l, r_c)\}$
13:         end if
14:     end for
15: end for
16: effect: $c_{new} \in V(CG) \wedge c_{new}.G.L = \{r\} \wedge c_{new}.f = r.f \wedge c_{new}.G.Fl = followers$
17: return $c_{new}$
```

4.3.2. *Community dismantling*

A community c is automatically dismantled; when it becomes empty $|V(c.G)| = 0$ (all of its members left or no longer exist). This is the only condition that triggers the disappearance of a community. This *precondition* is modeled as follows: $c \in V(CG) \wedge |V(c.G)| = 0$

After deleting a community c, it is no longer the extremity of any edge in the community graph CG: $c \notin V(CG) \wedge \forall c_1 \in V(CG)$, $(c, c_1) \notin E(CG)$

4.3.3. *Merging communities*

The first idea that comes to mind when deciding which communities to merge is closeness. This condition can be specified as follows: two communities c_1 and c_2 such as $(c_1, c_2) \in E(CG)$, where CG is the graph modeling the community network, can be merged if $w(c_1, c_2) < \xi$, where: $w(c_1, c_2)$ is the weight of the edge (c_1, c_2) and represents the distance between both communities centers (see Definition 3.7) and $\xi \in [0, 1]$ is the threshold beyond of which two communities can be merged.

However, this closeness condition is computed using a geometrical distance without taking into account the registries dispersion. Thus, an exception can take place when communities' centers are close to each other but with a weak density between centers. This happens when there are few registries in the communities' intersection (Fig. 7a) or when the communities are completely separated (Fig. 7b). As a consequence the closeness condition is necessary to check the similarity between communities functionalities but not sufficient.

Thus, we define the communities merging precondition by adding another condition to the closeness one defined above. This second condition checks if a community is included in another one. When this precondition is satisfied for two communities, they will be merged into a new one called c_{merg}. The center of c_{merg} is computed as the weighted average of both communities' centers $c_1 \cdot f$ and $c_2 \cdot f$:

$$c_{merg} \cdot f = \frac{c_1 \cdot f \times nb_1 + c_2 \cdot f \times nb_2}{nb_1 + nb_2} \tag{8}$$

where:

- $nb_1 = \#\{r | (c_1, d_1) \in r.MEM \wedge (c_2, d_2) \in r.MEM \wedge d_1 \geq d_2\}$, the number of registries in the intersection of c_1 and c_2 and having a greater membership degree to c_1.
- $nb_2 = \#\{r | (c_1, d_1) \in r.MEM \wedge (c_2, d_2) \in r.MEM \wedge d_2 \geq d_1\}$, the number of registries in the intersection of c_1 and c_2 and having a greater membership degree to c_2.

As a consequence of the merging step, the community c_{merg} is added to the graph CG and both communities c_1 and c_2 are deleted. Thereby, all edges whose ends are one of these two communities are removed too. The merging of two communities as well as the necessary precondition and effect are presented in Algorithm 5.

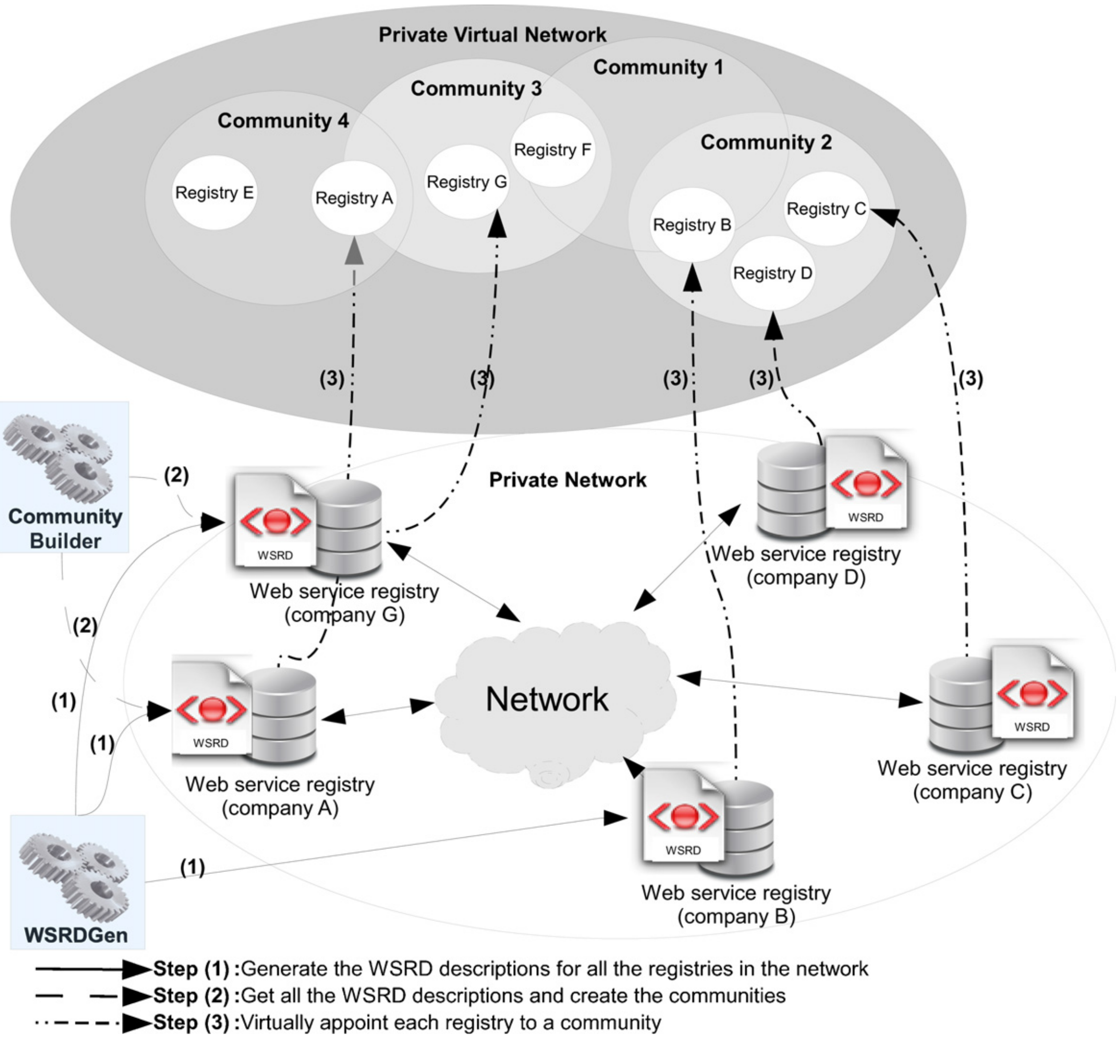

Private Virtual Network
Community 1
Community 3
Community 4
Community 2
Registry E
Registry A
Registry G
Registry F
Registry B
Registry C
Registry D
(3)
(3)
(3)
(3)
(3)
Community Builder
(2)
(2)
Private Network
WSRD
Web service registry (company G)
Web service registry (company D)
(1)
(1)
Web service registry (company A)
Network
Web service registry (company C)
WSRDGen
(1)
Web service registry (company B)
Step (1) :Generate the WSRD descriptions for all the registries in the network
Step (2) :Get all the WSRD descriptions and create the communities
Step (3) :Virtually appoint each registry to a community

Fig. 9. An overview on our test-bed.

Algorithm 5 *(Merging two communities).*

```
Require: $c_1, c_2$ : the two communities that triggered the merging
Ensure: $c_{merg}$ : the new community
1: precondition: $c_2 \in V(CG), \exists c_1 | w(c_1, c_2) < \xi \wedge V(c_1.G) \subset V(c_2.G)$   # if two communities offers similar functionalities and one community is included in another, they can be merged
2: Community $c_{merg}$;
3: List members;
4: $m_{max} = 0$;
5: $c_{merg}.f = \frac{c_1.f \times nb_1 + c_2.f \times nb_2}{nb_1 + nb_2}$   # compute the new mean functionality of the new community
6: members=$c_1.G.L \cup c_2.G.L \cup c_1.G.FL \cup c_2.G.FL$
7: for all Registries $r \in members$ do   # Compute the membership degrees of the registries to the new community and identify the leader
8:     $m \longleftarrow U_{merg}(r)$
9:     $r.MEM \longleftarrow r.MEM \cup (c_{merg}, m)$
10:    if $m > m_{max}$ then
11:        leader=$r$
12:    end if
13: end for
14: $c_{merg}.G.L = leader$
15: $c_{merg}.G.Fl = members \setminus \{leader\}$
16: effect: $V(CG) = (V(CG) - \{c_1, c_2\}) \cup \{c_{merg}\}$
17: return $c_{merg}$
```

4.3.4. *Splitting a community*

A community is automatically divided if it becomes sparse. The community sparsity describes a non density in the center vicinity and a dispersion between members (Fig. 8). If this condition is satisfied, splitting a community can be seen as a graph partitioning problem. Let's take the example of a community c and its undirected weighted graph $c.G$. $c.G$ represents the similarity weights between the *leader* and the *followers* of c. $\overline{c.G}$, the complement of $c.G$ (see Section 3.1), is also a weighted graph representing similarity weights between *followers*. The weighted adjacency matrix of the complete graph $c.G \cup \overline{c.G}$ will contain the similarity weights between all the community members (see Section 3.1). An algorithm that meets

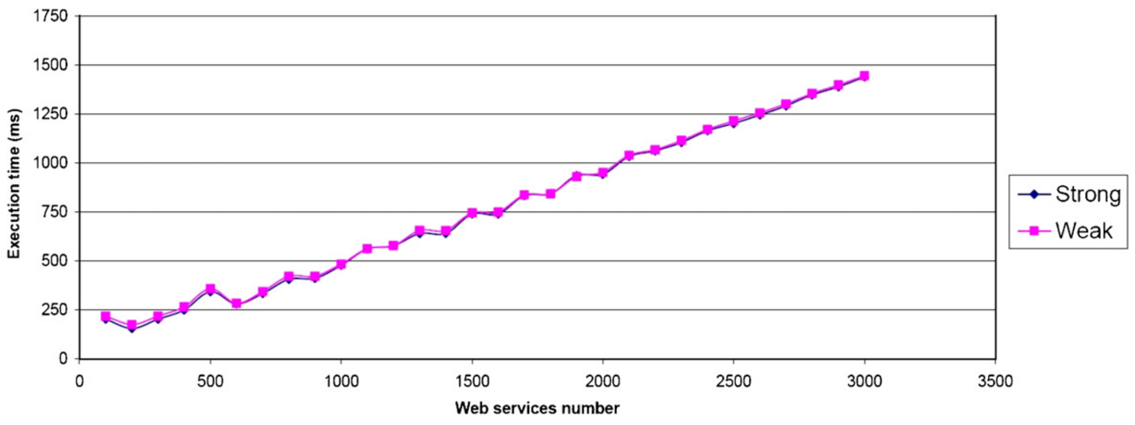


Fig. 10. Execution time for WSRD description computing.

our needs taking as input a weighted adjacency matrix of an undirected weighted graph is the *Mcut* algorithm (Ding et al., 2001; Nie et al., 2010). It proposes a graph partition method based on min-max clustering principle: the similarity between two sub graphs is minimized while the similarity within each sub graph is maximized. Fig. 8 shows how this algorithm is applied on a community *c* that satisfied the splitting precondition.

5. Experiments

In the following, we present the implementation and experimentation efforts made to validate the different steps of our approach. Our experimentation work was achieved in three stages: (5.1) preparing the test-bed, (5.2) experimenting the efficiency of our approach by selecting the right communities for Web services queries and (5.3) simulating communities of Web service registries using graphs and implementing a *Community Manager* to test the feasibility of our management approach. We also present in Section 5.4 a use case to show the usability of our approach.

5.1. Preparing the test-bed

The test-bed for our experiments, consists of a network of communities organized using our functionality-driven clustering approach (see Fig. 9).

Creating the test collection: We start by creating a test collection of semantic Web service descriptions written in SAWSDL. We adapted a semantic Web services description generator introduced in Chabeb et al. (2010) and adopted it to create a collection of SAWSDL service descriptions. A Web services description generator introduced in Oh et al. (2006) has characterized realistic Web services. We have followed this work to generate a corpus of services with a similar distribution of occurrences of signatures. These service descriptions are annotated using concepts from an ontology described in OWL. For our experimentation, we generated a collection of 1400 SAWSDL descriptions. The generated test collection also includes seven queries Q_i, $i = 1, \ldots, 7$ and the relevance sets (queries responses) associated to these queries. Each relevance set contains 100 SAWSDL descriptions. We split up the generated SAWSDL descriptions into 7 Web service registries R_j, $j = 1, \ldots, 7$. Each registry R_m contains the 100 descriptions of the relevance set of the query Q_m in addition to 100 randomly generated descriptions.

Computing the WSRD descriptions: We compute for each registry the associated WSRD descriptions using the WSRDGen tool[5] that we implemented (step (1) in Fig. 9). WSRDGen performs the three steps (see Section 2) of our WSRD computing approach and experiments show that our approach is usable in realistic situations. For example, processing the WSRD description of a registry advertising 1400 Web services is done in 0.65 s (see Fig. 10).

Building the communities: After that, we applied our clustering approach to build the communities of registries. We implemented a *Community Builder* using the Fuzzy Clustering and Data Analysis Toolbox.[6] We used the 7 WSRD descriptions (step (2) in Fig. 9) and transformed them into 5-dimensional vectors according to our vector space (Section 3.3.1). After that, we partitioned them into 4 communities (step (3) in Fig. 9).

We pushed further our experiments by testing our community building approach on a greater test collection. We used a set of 100 generated WSRD descriptions and transformed them into 6-dimensional vectors. We partitioned our data set into 5 communities and we present the clustering results in Fig. 11. Since we can only graphically visualize our results in 3- or 2-dimensional graphs, the *N*- (6 in our case) dimensions were devised in two 3-dimensions graphs: Fig 11a shows the clustering results using the first 3 dimensions of our data set and Fig 11b using the next 3 dimensions.[7]

While creating the vector representation of a WSRD description, we considered all the semantic concepts of the used DO to provide an accurate representation. This choice can theoretically lead to "relatively"[8] high dimensional vectors (number of dimensions equal to the size of the DO). However, these vectors are sparse and do not lead to scalability issues since the WSRD's information that we represent are not related to all the semantic concepts. To test the scalability of our clustering approach while dealing with high dimensional data, we applied our clustering approach on the 100 generated WSRD descriptions and considered the execution time. In this experiment, the size of the WSRD vectors was increased from 10 to 100. Fig. 12 shows that the clustering time is linear with respect to the vectors' size.

It is worth mentioning that in a real world scenario, the size of WSRD vectors (i.e. equal to the size of the used DO) can be hundreds, but should not be thousands normally. In addition, in our work, we use the clustering technique only once as "cold starter" to build the communities and we use management operations to guarantee

[5] http://www-inf.it-sudparis.eu/SIMBAD/tools/WSRDGen/.

[6] http://www.fmt.vein.hu/softcomp/fclusttoolbox/.

[7] Further details on our clustering results can be found at http://www-inf.it-sudparis.eu/SIMBAD/tools/FCM_results/.

[8] Very high dimensional vectors can be observed while clustering text documents where, if a word frequency vector is used, the number of dimensions is equal to the size of a dictionary.

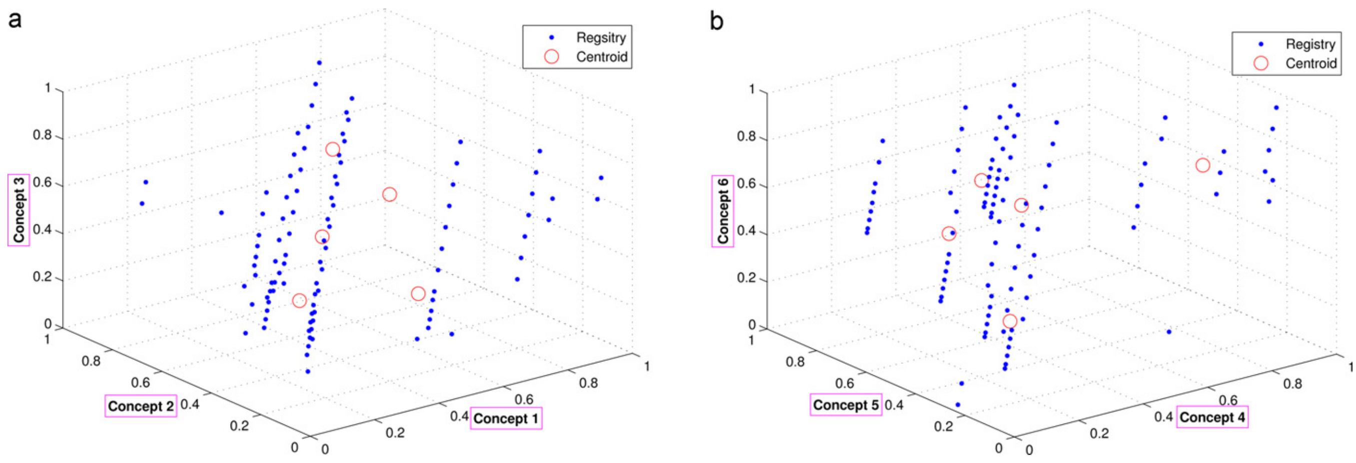


Fig. 11. Fuzzy clustering of 100 WSRD registry descriptions into 5 clusters.

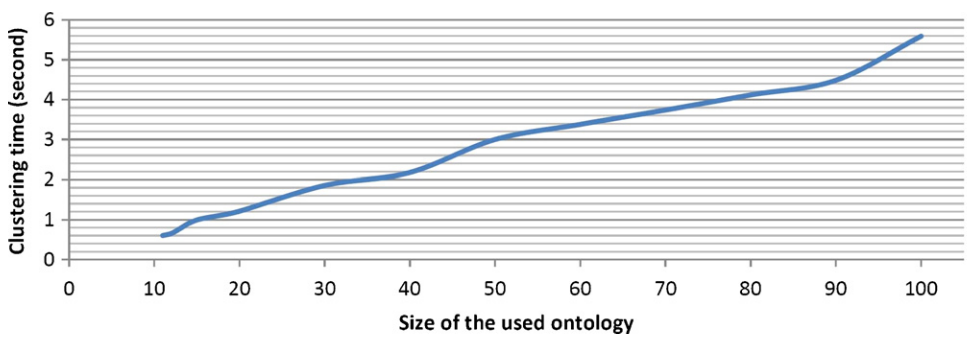


Fig. 12. Relation between size of WSRD vectors and clustering time.

the evolution of the organization. Consequently, the performance of our clustering approach is satisfactory in real situations.

5.2. Community selection

To verify the well organization of our communities through our clustering approach, we experimented the community selection process for a service requester's query. For an easy illustration, we took our first test-collection containing only 7 registries. We computed the similarities between the generated queries Q_i, $i = 1, \ldots, 7$ of our test collection and the different communities C_k, $k = 1, \ldots, 4$ centers. Since the community centers are represented by 5-dimensional vectors, we transformed our queries, written in SAWSDL, into 5-dimensional vectors using the same approach as for the WSRD descriptions (see Section 3.3.1). The similarity between a query Q_i and a community C_k center is computed using the cosine function (Section 3.3.2, formula (3)). The results are given in Table 2.

Based on our registry membership table (Table 3), computed as shown in Section 3.3.3, we can deduce that for each query Q_m the most similar community C_k contains the registry R_m proposing the services of the relevance set of Q_m. For example, the most similar cluster to the query Q_2 is C_4 (*Similarity*(Q_2, C_4) = 0.982). According to Table 3, R_2 (registry containing the relevance set for query Q_2) belongs to the cluster C_4 ($U_4(R_2)$ = 0.9287) which indicates the accuracy of our community building approach and its efficiency for query routing.

In real world B2B scenario, we can deal with hundreds of Web service registries. The used clustering technique in this work (i.e. fuzzy C-means) is a well-known technique used for many data analysis and classification problems. It provides efficient clustering results for small and average sized data sets. Indeed, the experimentation done in Fung (2001) on different real data sets (size = 486, 569 and 4192) prove the clustering efficiency of Fuzzy C-means. So we are able to state that the size of the data set will not have a negative impact on the community selection efficiency.

In addition, we considered the effect of the distribution of the Web services inside a registry on the community selection. We randomly re-distributed our description collections among our seven registries and experimented again the community selection process. We tested several distributions and used seven different queries for each experiment and considered the recall (the rate of registries in the selected community offering adequate services to the query) and the precision (the rate of discovered registries in the community offering adequate services to the query among all

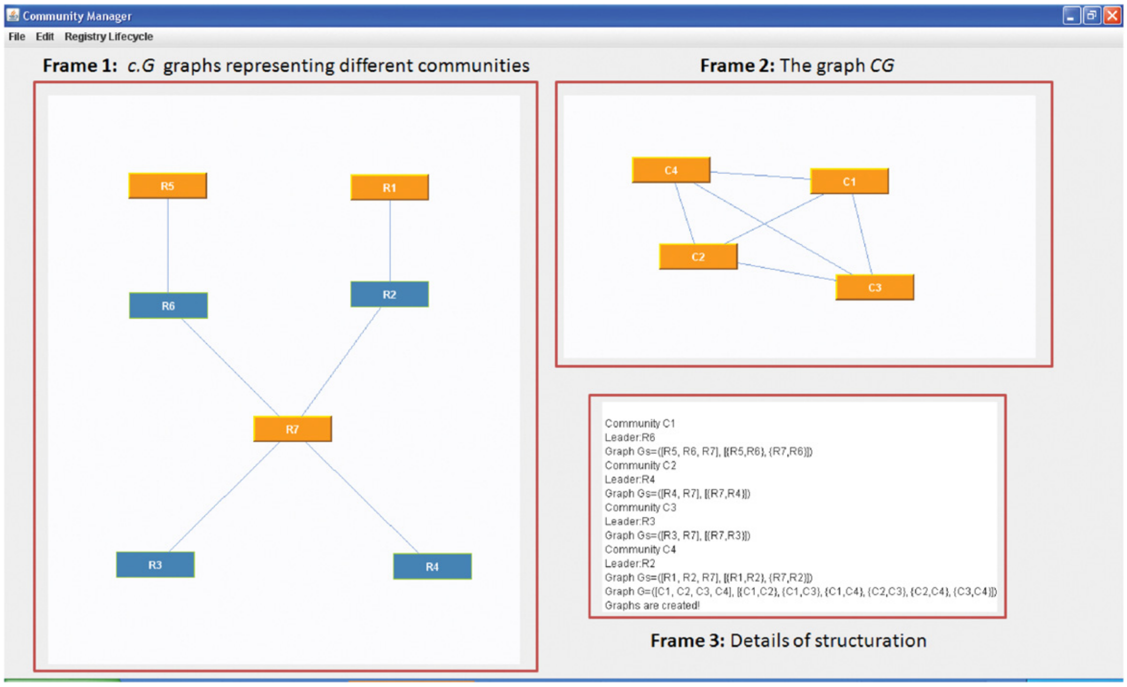
Table 2
Similarity degrees between Q_i and C_j.

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
C_1	0.960	0.968	0.808	0.921	**0.943**	**0.953**	0.784
C_2	0.967	0.962	0.887	**0.932**	0.939	0.935	**0.982**
C_3	0.920	0.899	**0.944**	0.802	0.790	0.776	0.856
C_4	**0.975**	**0.982**	0.822	0.905	0.889	0.905	0.852

Bolded values are used to identify the most similar community C_j to the query Q_i.

Table 3
Membership degrees of the registries to different clusters.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
C_1	0.0216	0.0271	0.0017	0.0329	0.9388	0.9491	0.1485
C_2	0.0183	0.0177	0.0026	0.9537	0.0450	0.0298	0.4002
C_3	0.0233	0.0165	0.9922	0.0060	0.0050	0.0057	0.3090
C_4	0.9369	0.9387	0.0034	0.0074	0.0112	0.0154	0.1423


Community Manager
File Edit Registry Lifecycle
Frame 1: c.G graphs representing different communities
R5
R1
R6
R2
R7
R3
R4
Frame 2: The graph CG
C4
C1
C2
C3
Frame 3: Details of structuration

```
Community C1
Leader:R6
Graph Gs=([R5, R6, R7], [{R5,R6}, {R7,R6}])
Community C2
Leader:R4
Graph Gs=([R4, R7], [{R7,R4}])
Community C3
Leader:R3
Graph Gs=([R3, R7], [{R7,R3}])
Community C4
Leader:R2
Graph Gs=([R1, R2, R7], [{R1,R2}, {R7,R2}])
Graph G=([C1, C2, C3, C4], [{C1,C2}, {C1,C3}, {C1,C4}, {C2,C3}, {C2,C4}, {C3,C4}])
Graphs are created!
```

Fig. 13. Community manager.

registries in the selected community). These experiments showed an average recall of 85% and an average precision of 80%.

5.3. Community management

In this part of our experiments, we simulate the different management operations over our built network of communities. We implemented a *Community Manager & Simulator* with java using the *Jgrapht* [9] library. In order to build the graphs modeling the communities and the registries, the inputs for our simulator were:

- 7 vectors $f = [w_1, w_2, \ldots, w_5]$ representing the functionalities offered by the advertised Web services of a registry (i.e. the registries WSRD descriptions vectors) as computed in Section 3.3.
- 4 vectors $\{c \cdot f | c \in C\}$ representing the mean functionality of the communities.
- 7 registry membership degrees to the 4 communities $\{r.MEM | r \in R\}$, with R the set of registries in the network.

These inputs represent the initialization data for our *Community Manager & Simulator* to draw the graphs modeling the communities and the registries. In Frame 1 of Fig. 13, we represent the graph composed by the set of graphs $c.G$ $(c \in C)$ corresponding to the 4 communities. The *leader* of each community is represented by a blue rectangle. In Frame 2, we represent the global graph *CG* of our network of communities. We fixed the threshold `th` beyond of which a registry belongs to a community at 0.1.

As reported in Section 4, the main triggers of dynamic changes are service and registry providers. In the following, we apply these changes to test managing algorithms and operations:

- **Adding a registry**: In order to test *CommunitySelection* and *LeaderReselection* algorithms, we add a new registry in the network represented by its vector *f*=[0.234 0.314 0.048 0.181 0.534]. The *Community Manager* assigns an identifier to this new registry (r_8) and then compute its membership degrees: $r_8.MEM = \{(c_1, 0.1943), (c_2, 0.282), (c_3, 0.2759), (c_4, 0.2479)\}$.
  MEM is then compared with `th` = 0.1. The new registry belongs to all the existing communities. At this level, the *LeaderReselection* algorithm assign the role of *follower* for r_8 in each community it belongs to. If we change `th` to 0.3, we notice that all membership degrees of r_8 to the existing communities are lower than `th`. In this case, the *community Manager* checks the community creation precondition and establishes automatically a new community by adding a new vertex to *CG* graph. Accordingly, the *CommunityCreation* algorithm is executed establishing a new community where r_8 assumes the role of leader.
- **Dismantling a registry**: The community c_3 is composed of only two registries: r_3 the *leader* and r_7 its *follower* (Fig. 13). By dismantling r_3, the *LeaderReselection* algorithm is applied assigning r_7 to be the *leader*. By dismantling r_7, the community c_3 is automatically dismantled since the dismantling precondition is satisfied (i.e. c_3 becomes empty).
- **Updating registry functionalities**: The registry functionalities are updated when a service provider publish, unpublish or modify a Web service description advertised within this registry. Using the updated graph of Frame 1 result of the addition of r_8 to the network and assuming that `th` = 0.1, we first update r_3's functionalities. r_3 leaves the community c_3 because its membership to this community is now lower than `th`. Actually, c_3 is composed of r_7 and r_8. We notice that c_3 is included in c_1, c_2 and c_4.
  The **merging precondition** is partially satisfied. However, we must check that the weights of edges whose ends are c_3 and one of the communities c_1, c_2 and c_4 are lower than $\xi = 0.2$ (see Definition 3.7). c_2 satisfies this condition. Thereby, c_2 and c_3 are merged into a new community c_5.

These scenarios show the feasibility and validity of our algorithms as well as the management operations used to handle registry and community life cycles.

[9] http://www.jgrapht.org/.

5.4. Use case: data providing services clustering and management

Our communities construction and management approach has been adopted in the French ANR funded research project PAIRSE.[10] The PAIRSE project deals with issues (heterogeneities, query processing, etc.) related to data sharing in P2P environments by using data providing (DP) services. A DP service is particular type of Web services that only allows data access.

DP services clustering and categorization is important for locating an appropriate service, when a user needs to (i) discover a service that can fulfill her requirements, or (ii) to replace a service involving a given interaction when this service disappears or is unavailable. For this purpose, we adapted our communities construction and management approach to organize DP services described by RDF views (Zhou et al., 2011). Concretely, DP services were represented in terms of vectors while considering the composite relation between input, output, and semantic relations between them. Thereafter, DP service vectors are clustered using a refined fuzzy C-means algorithm. In addition, we have adopted some community/registry management operations for managing service clusters and the cluster network when handling the following situations: new service emergence, and existing service disappearance or unavailability.

6. Related work

In this paper, we presented our approach for organizing Web service registries as registry communities. We also provided the needed management mechanisms to monitor communities' potential changes as well as the associated management operations. In this section we divide our literature review into the following parts: services organization (Section 6.1), registries organization (Section 6.2) and communities management (Section 6.3).

6.1. Services organization

Some Web services discovery approaches have used Web services WSDL descriptions to group Web services in order to enhance the discovery process. In Liu et al. (2009), homogenous Web service communities are built where, a community contains a set of services providing either similar operations or potentially composable operations, with respect to a given request. Concretely, concepts are clustered from terms appeared in inputs and outputs. The similarity of service operations is computed as the combined similarity of (i) the text description and (ii) the parameters of inputs and outputs. Service Aggregation Graph (SAG) is constructed where: (i) each vertex represents an operation and (ii) each edge reflects an output and input link between operations. To summarize, this approach aims to apply SAG to represent potential links between operations of Web services. So service discovery can be conducted by searching SAG and returning either a single operation or the shortest path of operations. Whereas in our work we group Web service registries into communities, where a Web service registry may represent several Web service operations. We aim to group similar registries into a same community, rather than to explore the links between Web services.

Another Web services organization approach is proposed in Dong et al. (2004). In this work, Web services parameter names defined in service operations are grouped into semantically meaningful concepts. Then, these concepts are used to measure the similarity of service operations by applying TF/IDF (i.e., term frequency/inverse document frequency) on the bag of words. This approach considers mostly parameter names, and does not deal with the composition problem. Generally, this approach aims to discover services based on their parameter names.

To summarize, compared to these techniques, we share the same methodology through using the Web services functional descriptions as an organization criterion. However, the main differences are:

- We are not organizing single services but rather atomic sets of Web services (i.e. Web service registries). These registries can advertise many functionally different Web services. So, in order to ensure this organization, we proposed a semantic model (WSRD) reflecting the mean functionality of a set of Web services and used it as a criterion unlike the work in Liu et al. (2009) and Dong et al. (2004) where the organization is based on WSDL descriptions.
- We consider semantically annotated descriptions and uses the semantic concepts annotating the Web service descriptions to compute the similarity between registries. In Liu et al. (2009) and Dong et al. (2004) the similarity measures are based on syntactic terms from WSDL descriptions specified by different Web services providers and can thereby be ambiguous.
- We consider the combined relation between service elements, and provide methods for handling the cases of new service/registry emergence or existing service/registry disappearance.

6.2. Registries organization

Several Web services discovery systems rely on a distributed registry environment to overcome the problems related to single centralized registry based discovery (bottlenecks, single point of failure, etc.). Our literature review yielded a good number of research projects that used distributed registries such as (Verma et al., 2005; Sellami et al., 2010c; Ayorak and Bener, 2007; Pilioura and Tsalgatidou, 2009; Xu et al., 2007), but as far as we know, none of them have promoted the idea of using a functionality based semantic description model to organize Web service registries.

MWSDI (Verma et al., 2005) propose a distributed Web service registry infrastructure for Web services discovery. In this work, the registry network is structured into federations using a specialized ontology called the *Registries Ontology*. This ontology allows grouping different Web service registries based on their business domain as the authors map each registry to one or several nodes in the *Registries Ontology*. Associating with each registry a specific domain or a group of domains from the *Registries Ontology* enhances Web services discovery. A Web service requester query will not be spread across all the registries since the query can be directly routed to the more adequate registry according to the requester's domains of interest. In Pilioura and Tsalgatidou (2009), the authors also propose a distributed registry infrastructure called PYRAMID-S, where Web service registries are categorized by concepts token from a specialized ontology. This categorization allows selecting the adequate registry for a service requester query. In previous work (Sellami et al., 2010c), we introduced the concept of requester characterization: a data structure containing a service requester's areas of interests, invocation history and non-functional requirements. A global registry characterization, which is the fusion of past requester characterizations who successfully discovered a service in that registry, is associated with a Web service registry. Using recommendation techniques, one or several registries can be advised to the service requester according to his characterization. A global registry characterization describes a Web service registry based on the consumers who used services it advertise but do not give an idea to the real functionalities it offers.

Authors in Ayorak and Bener (2007) present a super peer Web service discovery architecture. This architecture is based on a

[10] https://picoforge.int-evry.fr/cgi-bin/twiki/view/Pairse/Web/.

semantically clustered P2P network of *registry peers* (repository of Web service descriptions). Each *registry peers* cluster is indexed by a super peer called the *index peer* that stores the index information of the *registry peers* in a tree-based search data structure. In addition, the *index peers* are connected to each other with the CAN (Ratnasamy et al., 2001) routing protocol. In this architecture, a received search query will be routed by the *index peers* to the adequate *registry peer*. Using the CAN routing protocol avoids useless messages flooding the P2P network.

In Xu et al. (2007), the authors use the match history to route a requester's query to an adequate registry. A service routing table is introduced into each registry node to store the history of matches. A services requester's query will be routed to a registry according to past requests and their matches, stored in a registry's routing table.

To sum up, our literature review yielded a good number of research projects that used distributed registries, but as far as we know, none of them have promoted the idea of using a functionality-based semantic description model to organize Web service registries. Compared to the above literature review, our contributions can be summarized in three main points:

1. Our registry description and organization is based only on implicit knowledge using existing advertised service descriptions. Thus, our approach is self-contained within the Web service discovery process, independent from any explicit or human centric or error prone knowledge. Whereas other classification approaches (Pilioura and Tsalgatidou, 2009; Verma et al., 2005) ask for additional explicit knowledge such as user's classification or service reputation and rating which can be hard to be captured.
2. Our approach proposes a functionally-based organization of Web service registries into communities. Such an organization enhances the Web services discovery process since the registries will be grouped according to the functionalities proposed by their advertised services, and thus a service requester's query can be guided to the adequate registry cluster for his needs. Such a query routing mechanism is radically different from those used in the previous approaches. Indeed, their query routing is based on registry users characterizations (Sellami et al., 2010c), the registry's business domains (Verma et al., 2005; Pilioura and Tsalgatidou, 2009), Web services business domains (Ayorak and Bener, 2007) or previous discovery results (Xu et al., 2007).
3. Due to the inherent autonomy, continuous and unforeseeable evolution of Web services description, Web service registries operate within a dynamic environment. By fuzzy clustering Web service registries according to their service descriptions, our approach has intrinsically the means for a dynamic, flexible and automatic management of Web service evolutions. Indeed, WSRD can be updated whenever the service provider can publish, delete or modify a Web service description in order to be in line with the services they advertise. Whereas existing distributed registry environments (Pilioura and Tsalgatidou, 2009; Sellami et al., 2010c; Verma et al., 2005) are not suitable for the management mechanisms. Indeed most of them are characterized by a rigid and a priori registry classification and organization which hamper dynamic management mechanisms.

6.3. Communities management

Several Web service discovery approaches in distributed registry environments (see Section 6.2) organize their networks as groups but did not provide the management mechanisms for these groups. In this section, we overview some related efforts in the field of managing e-catalogs communities (Paik et al., 2005) and Web services communities (Medjahed and Bouguettaya, 2005; Maamar et al., 2009) that helped us tailor our approach.

Paik et al. (2005) present the WS-catalogNet framework allowing to group e-catalogs into communities, build relationships between them and manage them constantly. An e-catalog is defined as a set of products organized based on a categorization. An e-catalog community is a set of e-catalogs having similar domain. The system offer monitoring functionalities and managing operations to restructure a community network according to the user interaction. Therefore, authors model the community network and then specify preconditions and effects for each operation based on the model they have defined (Paik et al., 2002). However, the specified management operations, in particular merging and splitting communities, are not applicable in our context. Indeed, the members of a community (catalogs, Web services) have exclusive memberships while in our work a registry can belong to one or more communities.

Medjahed and Bouguettaya (2005) propose an approach to organize Web services into communities depending on their domain of interest. A community is an instance of an ontology meta-data called community ontology and is described by a set of generic operations. In this context, community providers can add, delete or modify some generic operations. Service providers, in turn, can delete a WS from a community or make its operations temporarily unavailable. Thus, authors propose a P2P approach to manage these changes.

Maamar et al. (2009) discuss the dynamic nature of Web service community and focus on potential conflicts. They propose an approach to engineer Web services communities in order to reconcile these potential conflicts. This approach is based on two protocols. The first one is called Community development protocol (WSCDProtocol) and is interested in managing communities in term of attracting and retaining Web services, creating and dismantling communities.

The aforementioned research works employ a classification technique to organize communities, while we use a dynamic clustering technique. Furthermore, only Maamar et al. use the functionality criterion to structure communities. The other works use rather the business domain. While studying these research works, we noticed that the community management is generally established after a change initiated by end-users or service or community providers. To address conflicts that may result due to these changes, all these works propose managing operations. To facilitate the specification of these operations, Paik et al. model their community network based on graph theory. The other works didn't propose a model and their descriptions are rather informal.

7. Conclusion

We presented an approach for building and managing communities of Web service registries. These communities are implicitly and automatically created using the registries WSRD descriptions. Compared to existing distributed registries organization approach, our approach uses a functionality-driven clustering and organizes registries according to the functionalities of the service they advertise. This functionality-driven organization of a registry network enhances a Web service discovery. In fact, a service requester's search space can be reduced to the community of registries advertising Web services offering the needed functionalities. We defined the required communities and community member's management operations to maintain the consistency of the communities. We also experimented the efficiency of our approach in selecting the right communities for a Web service query, simulated a network of registry communities to test our management approach and provided a use case to show the feasibility of our communities' construction and management approach.

As a second use case for our approach, we are working now on applying communities' organization in a Cloud context. A first paper in which we used our communities building approach to set up a marketplace for Cloud services was published (Sellami et al., 2012). As part of our perspectives, we plan to test the efficiency of our approach using a recommendation-based discovery system, introduced in a previous work (Sellami et al., 2010c), on top of a Web service registry community network. In addition, we aim to deal with the potential semantic lost induced by the reduction step of our WSRD computing approach. In fact, a lost of semantic could result while flattening the Web service descriptions. This issue can be handled by taking into account the different semantic relations between the WSRD elements at the reduction step.

References

Ayorak, E., Bener, A.B., 2007. Super peer web service discovery architecture. In: International Conference on Data Engineering, Istanbul, Turkey, 2007.

Benatallah, B., Sheng, Q.Z., Dumas, M., 2003. The self-serv environment for web services composition. IEEE Internet Computing 7, 40–48.

Bentahar, J., Maamar, Z., Benslimane, D., Thiran, P., 2007. An argumentation framework for communities of web services. IEEE Intelligent Systems 22 (6), 75–83.

Bezdek, J.C., 1981. Pattern Recognition with Fuzzy Objective Function Algorithms. Kluwer Academic Publishers.

Bondy, J.A., Murty, U.S.R., 2007. Graph Theory. Springer, London.

Bouchaala, O., Sellami, M., Gaaloul, W., Tata, S., Jmaiel, M.,2012. Modeling and managing communities of web service registries. In: Web Information Systems and Technologies. Vol. 101 of Lecture Notes in Business Information Processing. Springer, Berlin/Heidelberg, pp. 88–102.

Chabeb, Y., Tata, S., Ozanne, A., 2010. YASA-M: a semantic web service matchmaker. In: AINA 2010, Perth, Australia, April 20-23.

Ding, C.H.Q., He, X., Zha, H., Gu, M., Simon, H.D., 2001. A min–max cut algorithm for graph partitioning and data clustering. In: IEEE International Conference on Data Mining, Washington, DC, USA.

Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J., 2004. Similarity search for web services. In: Proceedings of the Thirtieth international conference on Very Large Data Bases – vol. 30. VLDB'04, pp. 372–383.

Dunn, J., 1973. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. Journal of Cybernetics 3, 32–57.

Fung, G., 2001, June. A Comprehensive Overview of Basic Clustering Algorithms.

Jiang, J.J., Conrath, D.W., 1997. Semantic similarity based on corpus statistics and lexical taxonomy. The Computing Research Repository, CoRR cmp-lg/9709008.

Jursic, M., Lavrac, N., 2008. Fuzzy clustering of documents. In: Conference on Data Mining and Data Warehouses, SiKDD 2008, Ljubljana, Slovenia.

Lausen, H., Farrell, J., 2007, August. Semantic Annotations for WSDL and XML Schema. W3C Recommendation, W3C. http://www.w3.org/TR/2007/REC-sawsdl-20070828/

Lin, D., 1998. An information-theoretic definition of similarity. In: The Fifteenth International Conference on Machine Learning (ICML 1998), Madison, WI, USA, July 24–27, 1998, pp. 296–304.

Liu, X., Huang, G., Mei, H., 2009. Discovering homogeneous web service community in the user-centric web environment. IEEE Transactions on Services Computing 2 (2), 167–181.

Maamar, Z., Sattanathan, S., Thiran, P., Benslimane, D., Bentahar, J., 2009, December. An approach to engineer communities of web services – concepts, architecture, operation, and deployment. International Journal of E-Business Research (IJEBR) 9 (4).

Martin, D., et al., 2004. Owl-s: Semantic Markup for Web Services. Tech. Rep. http://www.w3.org/Submission/OWL-S/

McConnell, J.J., 2008. Analysis of Algorithms: An Active Learning Approach. Jones and Bartlett Publishers.

Medjahed, B., Bouguettaya, A., 2005. A dynamic foundational architecture for semantic web services. Distributed and Parallel Databases 17 (2), 179–206.

Nie, F., Ding, C., Luo, D., Huang, H., 2010. Improved min–max cut graph clustering with nonnegative relaxation. In: European Conference on Machine Learning and Knowledge Discovery in Databases: Part II.

Oh, S.-C., Kil, H., Lee, D., Kumara, S.R.T., 2006. WSBen: a web services discovery and composition benchmark. In: ICWS, Chicago, IL, USA.

Paik, H.-Y., Benatallah, B., Hamadi, R., 2002. Dynamic restructuring of e-catalog communities based on user interaction patterns. World Wide Web 5 (4), 325–366.

Paik, H.-Y., Benatallah, B., Toumani, F., 2005. Toward self-organizing service communities. IEEE Transactions on Systems, Man, and Cybernetics, Part A 35 (3), 408–419.

Pilioura, T., Tsalgatidou, A., 2009. Unified publication and discovery of semantic web services. ACM Transactions on the Web (TWEB) 3 (3).

Qian, G., Sural, S., Gu, Y., Pramanik, S., 2004. Similarity between Euclidean and cosine angle distance for nearest neighbor queries. In: ACM Symposium on Applied Computing.

Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., Shenker, S., 2001. A scalable content-addressable network. In: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, San Diego, CA, USA.

Resnik, P., 1995. Using information content to evaluate semantic similarity in a taxonomy. In: The 1995 International Joint Conference on AI, IJCAI-95, Montreal/Quebec, Canada, August 20–25, 1995, pp. 448–453.

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D., 2005. Web service modeling ontology. Applied Ontology 1 (1), 77–106.

Salton, G., Buckley, C., 1987. Term Weighting Approaches in Automatic Text Retrieval. Tech. Rep.

Salton, G., Wong, A., Yang, C.S., 1975. A vector space model for automatic indexing. Communications of the ACM 18 (11).

Saraçoğlu, R., Tütüncü, K., Allahverdi, N., 2007. A fuzzy clustering approach for finding similar documents using a novel similarity measure. Expert Systems with Applications 33 (3).

Sellami, M., Bouchaala, O., Gaaloul, W., Tata, S., 2010a. WSRD: a web services registry description. In: International Conference on New Technologies of Distributed Systems, Tozeur, Tunisia, 2010.

Sellami, M., Gaaloul, W., Defude, B., Tata, S., 2012. Towards a unified marketplace for functionality-based cloud service discovery. In: CLOSER 2012 – International Conference on Cloud Computing and Services Science, Porto, Portugal, April 18–21, 2012, pp. 252–257.

Sellami, M., Gaaloul, W., Tata, S., 2010b. Functionality-driven clustering of web service registries. In: International Conference on Services Computing, Miami, FL, USA.

Sellami, M., Gaaloul, W., Tata, S., 2011. An implicit approach for building communities of web service registries. In: iiWAS'2011 – The 13th International Conference on Information Integration and Web-Based Applications and Services, Ho Chi Minh City, Vietnam, December 5–7, 2011.

Sellami, M., Gaaloul, W., Tata, S., Jmaiel, M., 2010c. Using recommendation to limit search space in web services discovery. In: 24th IEEE International Conference on Advanced Information Networking and Applications, Perth, Australia.

Sioutas, S., Sakkopoulos, E., Makris, C., Vassiliadis, B., Tsakalidis, A., Triantafillou, P., 2009. Dynamic web service discovery architecture based on a novel peer based overlay network. Journal of Systems and Software 82 (5), 809–824.

Sivashanmugam, K., Verma, K., Sheth, A.P., 2004. Discovery of web services in a federated registry environment. In: Proceedings of the IEEE International Conference on Web Services, San Diego, CA, USA.

The Mckinsey Quarterly, 2007. How Businesses are Using Web 2.0: A Mckinsey Global Survey.

Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J., 2005. Meteor-s WSDI: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. Information Technology and Management 6 (1), 17–39.

Washington, D.C.I.A., 2012. The World Factbook 2012. https://www.cia.gov/library/publications/the-world-factbook/index.html

Xu, B., Chen, D.,2007. Semantic web services discovery in P2P environment. In: International Conference on Parallel Processing Workshops. IEEE Computer Society.

Xu, Y., Tang, S., Xu, Y., Xiao, R., Fang, L., 2007. Discovering web services based on match history. In: Advances in Intelligent Web Mastering. Proceedings of the 5th Atlantic Web Intelligence Conference, Fontainbleau, France.

Zadeh, L., 1965. Fuzzy sets. Information Control 8, 338–353.

Zaremba, M., Vitvar, T., Bhiri, S., Hauswirth, M., 2011. Preference-based discovery of dynamically generated service offers. In: International Conference on Services Computing, SCC 2011, Washington, DC, USA, July 4–9, pp. 338–345.

Zhou, Z., Sellami, M., Gaaloul, W., Defude, B., 2011. Clustering and managing data providing services using machine learning technique. In: International Conference on Semantics Knowledge and Grid (SKG 2011), Beijing, China, October 24–26, pp. 225–232.

Mohamed Sellami received the PhD degree in computer science from Telecom SudParis in 2011. He is a Postdoctoral researcher at the SIMBAD group part of the computer sciences department of Telecom SudParis in France and member of the CNRS SAMOVAR research team. His current research areas include service oriented computing and their applications in virtual enterprises. More information can be found at: http://www-inf.it-sudparis.eu/ sellami/.

Olfa Bouchaala obtained her MS degree (2010) and engineer's degree (2009) in computer science from the National Engineering School of Sfax (ENIS), Tunisia. She is curruntly a PhD student at Telecom SudParis, Evry, France and National School of Engineering Sfax, Tunisia. Her current research focuses on business process management. More information can be found at: http://www-inf.int-evry.fr/ bouchaal/.

Walid Gaaloul received the MS (2002) and PhD (2006) degrees in computer science from the University of Nancy, France. He is an associate assistant professor at TELECOM SudParis, France. He was also a postdoctoral researcher at the Digital Enterprise Research Institute (DERI) and an adjunct lecturer at the National University of Ireland, Galway (NUIG). Before joining DERI, he was a junior researcher in the Lorraine Laboratory of IT Research and its Applications (LORIA-INRIA) and a teaching assistant in the University of Nancy. His research interests are in semantic business process management, process intelligence, process reliability, service-oriented computing, and semantics for B2B integration. He has published more than 30 research papers and has coauthored a book and several book chapters. He serves as a program committee member and a reviewer at many international conferences and journals and has been participating in several national and European research projects.

Samir Tata is a full professor at the Institut TELECOM, Evry, France. His current research area includes service-oriented computing and business process management with an emphasis on the description and support of process interaction in virtual enterprizes. More information can be found at: http://www-inf.itsudparis.eu/ tata.