GRAPH-BASED MANAGEMENT OF COMMUNITIES OF WEB SERVICE REGISTRIES

Olfa Bouchaala[1,2], Mohamed Sellami[1], Walid Gaaloul[1], Samir Tata[1]
[1]*TELECOM SudParis, CNRS UMR Samovar, Evry, France*

Mohamed Jmaiel[2]
[2]*University of Sfax, ReDCAD Laboratory, National School of Engineers of Sfax, Sfax, Tunisia*

Keywords: Web service registry, Registry community, Communities management.

Abstract: Web services discovery represents a prominent step in the development process of B2B applications. In such environment, the number of Web services as well as registries that are made available for use can be as many as the large number of companies. Thus, the Web services discovery process will be time consuming. To deal with this issue, one obvious solution is to group Web service registries into communities. However, this solution can raise a managing issue since registries and communities are dynamic by nature. Our major contribution in this paper is an approach for managing registries and communities to reconcile conflicts result of the dynamic change aspect of Web service registries.

1 INTRODUCTION

Within a B2B context, we are interested in Web service (WS) discovery in a distributed registry environment, where companies use WSs to achieve transactions with their partners and offer online WSs. The involved companies have to make their WSs accessible on the net and available for consultation through WS registries. As a result, the number of WS registries can be very large. Therefore, WSs discovery will be a cumbersome task. To deal with this problem and to address the large number of WS registries and their poorly organized network, we propose to organize WS registries into communities. We define a WS registry community as a set of registries offering WSs providing similar functionalities. This organization is based on a semantic model, that we call **W**eb **S**ervice **R**egistry **D**escription (WSRD) (Sellami et al., 2010a). WSRD descriptions rely on the descriptions of the WSs belonging to a given registry and "semantically aggregate" the WSs functionalities.

In a distributed registry network, each registry is then described by a WSRD description. According to their descriptions, registries will be virtually structured into communities (Sellami et al., 2010b). This solution reduces the search space for a service requester in the discovery process. However, it may raise other issues mainly related to community management. Indeed, communities and their members (i.e. WS registries) are dynamic by nature. In fact, a new WS 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. Therefore, management mechanisms are necessary to monitor these changes and reconcile potential conflicts.

In this context, we propose a graph-based approach for managing communities of WS registries which consists in a set of algorithms and managing operations. The managing operations are pre and post-conditions checking triggers and potential effects for each step of community life-cycle. The algorithms are rather defined for managing the registry life-cycle steps. These algorithms and operations are tested and validated using graph simulation.

This paper is organized as follows: in Section 2, we start by a brief introduction of concepts of the graph theory, we present our registry description model, we provide our definition of registries communities and present our approach for building such communities. The graph based model, that we propose to facilitate the specification of managing operations, is presented in Section 3. In Section 4, we define managing algorithms and operations for reg-

Bouchaala O., Sellami M., Gaaloul W., Tata S. and Jmaiel M..
GRAPH-BASED MANAGEMENT OF COMMUNITIES OF WEB SERVICE REGISTRIES.
DOI: 10.5220/0003347400610070
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 61-70
ISBN: 978-989-8425-51-5

istry and community life-cycles. The implementation efforts are shown in Section 5. Finally, we conclude our paper and we foresee some future works.

2 BACKGROUND

Since we model our WS registry community network based on graph theory, we start by briefly introducing graph prerequisites and some of the special types of graphs playing prominent role in our work. Then, we present the WSRD semantic model used to describe a registry. Afterwards, we present our community definition and architecture. Finally, we present our approach for building communities.

2.1 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 through a collection of nodes and connections between them. A given problem is then mathematically 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. The number of vertices $|V|$ of the graph G is its order. When G is not the only graph under consideration, the vertex- and edge-sets are denoted respectively $V(G)$ and $E(G)$.

A graph can be either directed or undirected. In the first case, each edge is an ordered pair of vertices. In the second case, edges represent unordered pairs of vertices. Both directed and undirected graphs may be weighted by a weight function $w : E \longrightarrow \mathbb{R}$ assigning a weight on each edge. A weighted graph is then denoted $G = (V,E,w)$.

2.1.1 Adjacency Matrix

The adjacency matrix A_G of a given graph $G = (V,E)$ of order n is an $n \times n$ matrix $A_G = (a_{i,j})$ where

$$a_{i,j} = \begin{cases} 1 & \text{if} \quad (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

In a weighted graph, the adjacency matrix can be used to store the weights of the edges (McConnell, 2008). Hence, the values of the adjacency matrix for G would be defined as follows :

$$a_{i,j} = \begin{cases} w_{i,j} & \text{if} \quad (i,j) \in E \\ 0 & \text{if} \quad i = j \\ \infty & \text{if} \quad (i,j) \notin E \end{cases}$$

2.1.2 Star Graph

Graphs are classified into different types according to the nodes organization as well as the relationship between them. In this work, we use a particular type of graphs, called star graphs. A star graph is a complete bipartite graph $K_{1,k}$ or $K_{k,1}$.

A graph G is called bipartite (or bigraph) (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. Such a graph is denoted $G = (X,Y,E)$. A bigraph 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. If $n = 1$ or $m = 1$, G becomes a star graph.

2.1.3 Operations on Graphs

To simplify the management operations and algorithms that we present in Section 4, we use some operations defined in graph theory. Since graphs are defined as pairs of vertex and edge-sets, we use the set-theoretical terminology to define operations on and between them. Among these operations, we remind the addition/deletion of a vertex or an edge to/from a graph and the complement of a graph.

The addition/deletion of a vertex v (resp. an edge e) to/from a graph $G = (V,E)$ yields to a union/substraction of the vertex set $V(G)$ and $\{v\}$ (resp. the edge set $E(G)$ and $\{e\}$). We remind that the deletion of a vertex v removes not only this vertex but also all edges with this vertex as extremity. The resulting graph is then denoted $G = (V \backslash \{v\}, E \backslash \{(u,v) \in E | u \in V\})$. 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 those edges that are not in G. $G \cup \overline{G}$ represents a complete graph.

2.2 WSRD: Web Service Registry Description

In our context, we are dealing with private registries, not public ones, belonging to a specific company. Thus, we assume that all the services advertised by a registry are homogenous in term of their business domain and semantics. In addition, since those services have the same provider, we suppose that they use the same semantic description language. To describe the functionalities of a WS registry, we proposed in previous works (Sellami et al., 2010a) to use a semantic model. This description, that we call **W**eb **S**ervice **R**egistry **D**escription (WSRD), results from the aggregation of the different WSs functional descriptions

advertised by a registry. The registry description computing process doesn't ask for any explicit knowledge from a registry provider. It is implicitly created using only as input the WSs descriptions of that registry and no further information are required.

In Figure 1, we consider an example of a registry advertising only two semantic WS descriptions written in SAWSDL (Lausen and Farrell, 2007). We suppose that the used ontology is composed of 7 concepts. Computing the registry's WSRD decription goes beyond three steps: (1) extracting the annotating concepts, (2) constructing the clouds of potential mean concepts and (3) reducing the clouds. In Figure 1, we show an "intermediate" WSRD graph description result of the first step consisting in storing the extracted concepts and the number of times they were identified in the corresponding service description elements. The obtained WSRD "intermediate" graph is used in the second step for computing the potential mean concepts of a WSRD description. A mean concept (C_{mean}) is the ontological concept annotating a WSRD element. A C_{mean} is computed on the basis of the extracted concepts and can be seen as the medium of these concepts. The third step consists in reducing the cloud of C_{mean} in order to select the median concept(s) which are the most similar to the ones identified in the first step. We use the Weak reduction technique (Sellami et al., 2010a) to deduce the resulting WSRD description graph.

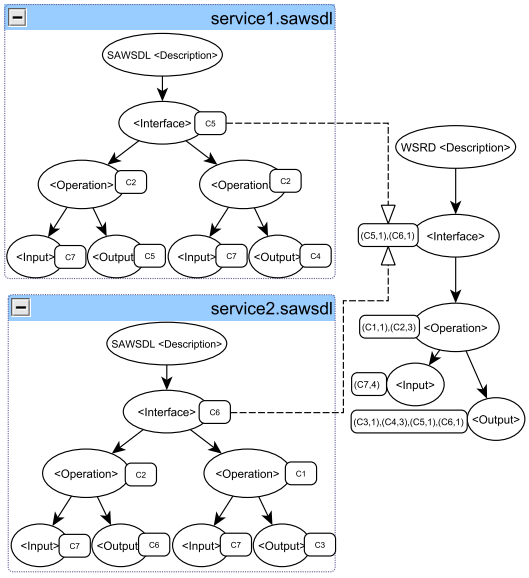


Figure 1: Concept extraction.

2.3 Communities of WS Registries

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 WSs research field, Benatallah et al. (Benatallah et al., 2003) define a WS community as *"a collection of Web services with a common functionality although different non-functional properties"*. Zakaria et al. (Maamar et al., 2007) 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"*. In the same spirit, we define a WS registry community as a set of registries offering WSs providing similar functionalities. So, a distributed registry network will be virtually structured into communities and each registry belongs to at least one community with a certain extent. We assign for each registry a set of membership degrees indicating its membership to the different communities. In each community we associate to one registry the role of *leader* and to the other members the role of *followers* (Figure 2). 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. Obviously, a *leader* for a community c_1 could be a *follower* for another community c_2 and vice versa. The *leader-follower* relationship within a community indicates the level of similarity between the functionalities offered by both of them.

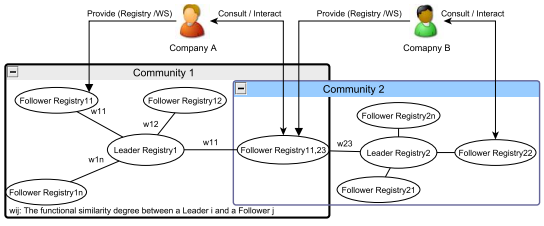


Figure 2: Architecture of WS registries communities.

2.4 Building WS Registry Communities

A WS registry community will bring together registries offering similar functionalities. Since a WS registry generally offers services proposing different functionalities, it is difficult to properly define in advance classes categorizing the functionalities of the different registries. To organize WS registries into communities, we used (Sellami et al., 2010b) 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 WS registries communities) will be identified from the given input data (i.e. the WSRD descriptions) and each data point (i.e. WS registry) will be assigned to one or many communities.

Since a registry can belong to different communities at the same time (Section 2.3), the use of an exclusive clustering is inadequate for building registry communities. Therefore, we proposed to use an overlapping clustering method to organize our distributed registries into communities. Using such clustering method, each data point (i.e. registry) represented by its WSRD description may belong to two or more communities with different *degrees of membership*.

Each WSRD description x will be represented as a vector $r_x = [w_{1x}, w_{2x}, \ldots, w_{tx}]$ where t is the total number of concepts in the ontology used to annotate the WSRD description. The weights of the different $w_{ix} = \alpha \times v_{interface} + \beta \times v_{operation} + \delta \times v_{input} + \lambda \times v_{output}$

- $v_{interface} = [e_1, e_2, \ldots, e_t]$ representing the C_{mean} of the resulting WSRD `<interface>`element.
- α, β, δ and λ are weights associated such as $\alpha+\beta+\delta+\lambda = 1$.

A distance measure is used to establish the degrees of membership of each WS registry, represented by the vector r_x, to the different clusters. In this work, we use the cosine similarity measure to establish the similarity between two given vectors r_1 and r_2 (formula (1)).

$$cosine(r_1, r_2) = \frac{r_1 \cdot r_2}{\|r_1\| \|r_2\|} \quad (1)$$

To deduce the distance from the cosine similarity function, we use formula (2).

$$distance(r_1, r_2) = 1 - cosine(r_1, r_2) \quad (2)$$

More details about our WS registries clustering approach can be found in (Sellami et al., 2010b).

3 MODELING COMMUNITIES OF WEB SERVICE REGISTRIES

Communities and WS registries operate within a dynamic environment where changes are mainly initiated by service and registry providers. The service provider can publish or delete a WS. Similarly, the registry provider can register its WS registry or dismantle it at any moment. To keep the consistency of our communities network against these events, management operations are needed. To facilitate the specification of these operations, we model the WS registry community network based on graph theory. In this section, we introduce our model representing a WS registry, a community and a community network.

3.1 Modeling a Web Service Registry

In this work, we refer to each WSRD description of a WS 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. Accordingly, a WS registry is defined as follows:

Definition 1. A registry is defined as a tuple $r = (id, f, MEM)$ where:

- id is the registry identifier.
- f is a vector representing functionalities offered by the advertised WSs within r.
- MEM represents the registry membership degrees to the different communities in the network. It is defined as a binary relation on $C \times [0,1]$. We remind that a binary relation is a set of ordered couples. $MEM = \{(c,d) | c \in C, d \in [0,1]\}$ 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:
$dom(MEM) = \{c | (c,d) \in MEM \text{ for some } d \in [0,1]\}$
$ran(MEM) = \{d | (c,d) \in MEM \text{ for some } c \in C\}$

3.2 Modeling a Web Service Registry Community

A community in our distributed registry environment is mainly characterized by its mean functionality f which represents the average of community registries functionalities. Registries can enter and leave a community almost at any time. Besides, we fix a threshold `th` beyond of which a registry could belong to a given community. As reported in section 2.3, 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 $Fl = \{fl_i | i : 1..n\}$ where n is the number of the community *followers*. Thus, the community nodes are modelled 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 (Section 2.4, formula (1)). Hence, we define a community as follows:

Definition 2. A community is a tuple $c = (id, f, G)$ with:

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

3.3 Modeling the Community Network

So far, our distributed registry environment which is a set of communities is modelled 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 their vectors f representing their mean functionalities. The distance can be computed using formula (2) of Section 2.4. The distance measure will be the weight of the edge relating these two communities. Our distributed registry network is then defined as follows:

Definition 3. The registry network is represented by an undirected weighted graph $CG = (C, E, w)$

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

4 MANAGING COMMUNITIES

In the following, we define the necessary management operations to handle WS registries and WS registry communities during their life cycles.

4.1 The Registry Life-cycle

A registry life-cycle starts when a registry provider decides to **register its WS registry** in the network (Figure 3 (1)). This registry **joins the network** (2) and then **joins the adequate communities** according to its offered functionalities (3). Since a service provider can **publish** (4) or **delete** (5) a **WS** within this registry, its WSRD description can change and an **update of the registry functionalities** (6) is needed. In such scenario, a suitability check of the registry membership should be done: If the registry membership degree is lower than a certain threshold `th` set by the designer, it **leaves the community** (7) and joins another one. Finally, the registry can **leave the whole network** (8) if its provider decides to **dismantle** it (9). In the following, we detail the steps (2), (3) and (6) of the registry life-cycle.

4.1.1 Joining the Network

When a new WS registry r joins our distributed registry environment, its WSRD description should be computed. After that, the registry can be guided to the adequate communities according to its set of membership degrees MEM. MEM is computed by the *CommunitySelection* algorithm (Algorithm 1). This algorithm takes as input the current registry's WSRD description and it is essentially based on the computation of the membership degree as the inverse of the distance between the community mean functionality $c.f$ and the registry functionalities $r.f$ (line 2). This distance measure is computed according to formula (2) (Section 2.4). 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 threshold `th` defined above (line 4), the registry will be guided to the adequate communities (line 5). If all membership degrees are lower than `th`, a new community will be created (Section 4.2.1).

Algorithm 1: *CommunitySelection*.

Require: r :registry
Ensure: $r.MEM$
1: **for each** community $c \in C$ **do**
2: $m \longleftarrow 1/distance(c.f, r.f)$
3: $r.MEM \longleftarrow r.MEM \cup \{(c,m)\}$
4: **if** $m >$ `th` **then**
5: $V(c.G) \longleftarrow V(c.G) \cup \{r\}$
6: **end if**
7: **end for**

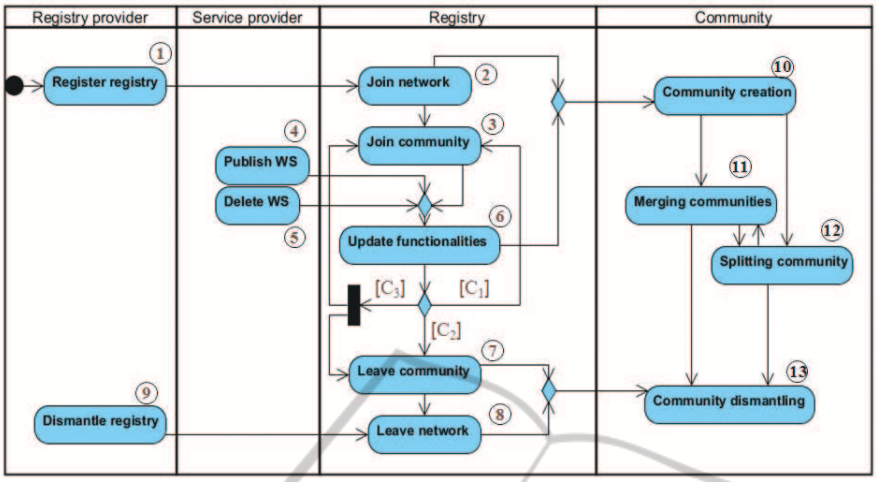


Figure 3: Communities management process.

4.1.2 Joining a Community

When a registry joins a community, it may have a membership degree greater than the community *leader*. In this case, the *LeaderReselection* algorithm (Algorithm 2) should be applied. It checks either or not the new registry will take the role of *leader* in one of the communities it belongs to (line 1, 5). This is done through a simple comparison between the new registry's membership degree and the community *leader*'s one (line 5). If the *leader*'s membership degree is still the greatest, then we only link the current registry with the community *leader* (line 6, 7). Otherwise, we remove all *followers-leader* links (line 9,10,11), add the *leader* to the *followers* set (line 12), the current registry takes the *leader*'s place (line 13) and the community *followers* will be linked to the new community *leader* (line 14,15,16).

Algorithm 2: *LeaderReselection*.

Require: r: registry
1: **for all** Communities $c \in dom(r.MEM)$ **do**
2: Let $\{l\} \longleftarrow c.G.L$
3: Let $d_r \in [0,1]$ such that $(c,d_r) \in r.MEM$
4: Let $d_l \in [0,1]$ such that $(c,d_l) \in l.MEM$
5: **if** $d_r >$ th and $d_l > d_r$ **then**
6: $c.G.Fl \longleftarrow c.G.Fl \cup \{r\}$
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: $c.G.Fl \longleftarrow c.G.Fl \cup \{l\}$
13: $c.G.L \longleftarrow \{r\}$
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**

4.1.3 Updating Registry Functionalities

We recall that the WSs advertised within a registry frequently change (new WSs arrive, others leave). Therefore, the registry functionalities have to be regularly updated. When a change occurs, the registry can stay in the same community, leave or move from a community to another. After a functionalities update, a registry acceptance or denial in a community happens according to the *CommunityAcceptance* algorithm algorithm (Algorithm 3). By applying this algorithm on a set of updated registries, 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. (Figure 3.[C1])
- $E' \subset E$, i.e. some registries leave the set of community members. (Figure 3.[C2])
- $E \nsubseteq E' and E' \nsubseteq E$, i.e. some new registries join the set of community members and some others leave. (Figure 3.[C3])

4.2 The Community Life-cycle

The main steps describing a community life-cycle revolve around community creation, dismantling, merging and splitting. When the membership degrees of a registry became lower than the threshold th for all existing communities, a **new community will be cre-**

Algorithm 3: *CommunityAcceptance*.

Require: r:registry, c:community
Ensure: (accept/deny)
1: Let $d \in [0,1]$ such that $(c,d) \in r.MEM$
2: $d \longleftarrow 1/distance(c.f, r.f)$
3: **if** $d >$ th **then**
4: *return accept*
5: **else**
6: *return deny*
7: **end if**

ated (Figure 3(10)). Also, a **community will be dismantled** (13) if it becomes empty. 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** (11) to another one or **split** (12). In the following, we present triggers and effects for each step.

4.2.1 Community Creation

A new community $c_{new} = (id, f, G)$ is established automatically, if the membership degrees of a registry to all the existing communities are lower than the threshold th. This situation necessarily implies that c_{new} provides a new functionality in the network. This can happen when a new registry joins the network (Section 4.1.2) or after an update of the registry's functionalities (see Section 4.1.3). So the ***Pre-condition*** for a community creation is modeled as: $\forall d \in ran(r.MEM), d <$ th

The registry r that triggered the community creation, will get the role of *leader* for the new community c_{new}. The community mean functionality $c_{new}.f$ will be the same as the functionality $r.f$ proposed by the registry. Afterwards, the *FollowersSelection* algorithm (Algorithm 4) will be executed to recruit *followers* for the new community. In this aim, the membership degrees of existing registries to the new community are computed. These different actions form the ***Post-condition*** for a community creation and are modeled as follows: $c_{new} \in V(CG) \wedge c_{new}.G.L = \{r\} \wedge c_{new}.f = r.f \wedge c_{new}.G.Fl = FolowersSelection(c_{new})$

4.2.2 Community Dismantling

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

After deleting a community, we must check the

Algorithm 4: *FollowersSelection*.

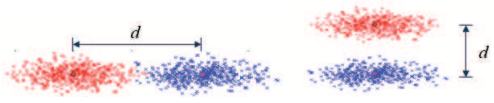
Require: c_{new} : community
Ensure: $c_{new}.Fl$: Follower Set
1: **for all** Communities $c \in C$ **do**
2: **for all** Registries $r \in V(c.G)$ **do**
3: Let $\{l\} \longleftarrow c_{new}.G.L$
4: $m \longleftarrow 1/distance(c_{new}.f, r.f)$
5: $r.MEM \longleftarrow r.MEM \cup (c_{new}, m)$
6: **if** $m >$ th **then**
7: $c_{new}.G.Fl \longleftarrow c_{new}.G.Fl \cup \{r\}$
8: $E(c_{new}.G) \longleftarrow E(c_{new}.G) \cup \{(l,r)\}$
9: **end if**
10: **end for**
11: **end for**
12: return $c_{new}.Fl$

Post-condition stating that c is not the extremity of any edge in the community graph CG: $c \notin V(CG) \wedge \forall c_1 \in V(CG), (c, c_1) \notin E(CG)$

4.2.3 Merging Communities

The natural idea that first comes to mind when deciding which communities to merge is closeness. Based on the graph CG and assuming that $(c_1, c_2) \in E(CG)$ (Section 3.3), this issue can be specified as follows: $w(c_1, c_2) < \xi$ such that $\xi \in [0,1]$ a threshold beyond of which two communities can be merged.

However, the closeness is computed using a geometrical distance without taking into account the registries dispersion. Thus, an exception can take place when communties centers are close to each other but not dense in the middle way between centers. i.e. few registries in the intersection (Figure 4(a)) or communities are completely separated (Figure 4(b)). As a consequence the closeness condition is necessary to check the similarity between communities functionalities but not sufficient.



(a) (b)

Figure 4: Distance between clusters centers (Zhang and Shin, 2005).

Thus, we define the communities merging pre-condition by adding another condition to the closeness one. This second condition checks if a community is included in another one. Our resulting ***Pre-condition*** will be: $c_2 \in V(CG), \exists c_1 | w(c_1, c_2) < \xi \wedge V(c_1.G) \subset V(c_2.G)$. When this pre-condition 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 $c_1.f$ and $c_2.f$:

$$c_{merg}.f = \frac{c_1.f \times nb_1 + c_2.f \times nb_2}{nb_1 + nb_2} \quad (3)$$

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.

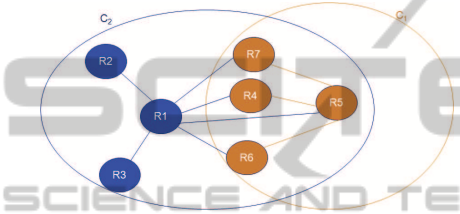


Figure 5: Merging pre-condition.

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. This ***Post-condition*** is modeled as follows: $V(CG) = (V(CG) - \{c_1, c_2\}) \cup \{c_{merg}\}$

4.2.4 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 (Figure 6). If this pre-condition is satisfied, this issue can be observed as a graph partitionning problem. Indeed, we consider a community c represented with its undirected weighted star graph $c.G$ which represents the similarity relationship between the *leader* and its *followers*. $\overline{c.G}$ is the complement of $c.G$ (Section 2.1.3) and is also a weighted graph representing similarity relationships between *followers*. The weighted adjacency matrix of the complete graph $c.G \cup \overline{c.G}$ contains all similarity weights between each pair of community members (Section 2.1.1). An algorithm which suits well to our needs taking as input a weighted adjacency matrix of an undirected weighted graph is the *Mcut* algorithm (Ding et al., 2001) which proposes a graph partition method based on min-max clustering principle: the similarity between two subgraphs is minimized while the similarity within each subgraph is maximized. Figure 6 shows how this algorithm is applied on a community c that satisfied the splitting pre-condition.

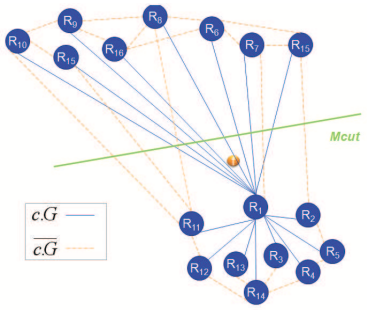


Figure 6: Splitting a community using *Mcut* algorithm.

5 IMPLEMENTATION

To test the feasibility of our approach, we simulate WS registry communities using graphs. Indeed, we implemented a *Community Manager* (Figure 7) based on the *Jgraphtjava* library (Jgrapht, 2003).

To validate the proposed algorithms and managing operations, we consider the following scenario: We generate 700 SAWSDL descriptions using the semantic description generator presented in (Chabeb et al., 2010). These descriptions are organized into 7 registries such that each one contains 100 descriptions. We compute the WSRD description of each registry using the *WSRDGen* implemented in (Sellami et al., 2010a). Each description is modeled with a vector $f = [w_1, w_2, ...w_5]$ computed as shown in section 2.4. These vectors represent the input of the clustering method (Sellami et al., 2010b) which outputs the mean functionality vectors $\{c.f|c \in C\}$ and the membership degrees of each registry to different communities $\{r.MEM|r \in R\}$, with R the set of registries in the network.

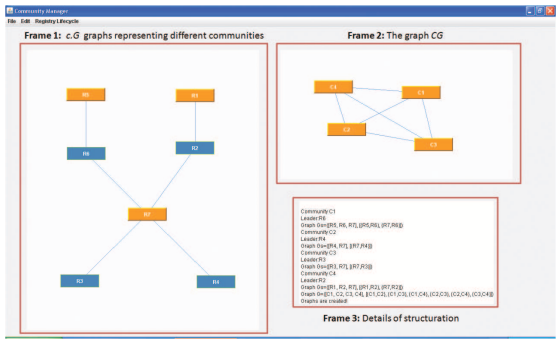


Figure 7: Community Manager.

The vectors $r.f$, $c.f$ and $r.MEM$, such that $r \in R$ and $c \in C$, are saved in an XML file representing initialization data for our *Community Manager* in order to build the graphs modeling the communities and the registries (Figure 7). In Frame 1, we present the graph composed by the set of graphs $c.G$ representing communities. The *leader* of each community is represented by a blue rectangle. In Frame 2, we represent the graph CG representing our community network.

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

Adding a Registry. In order to test ***Community-Selection*** 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 compared with $\mathtt{th} = 0.1$. The new registry belongs to all the existing communities. At this level, the *LeaderReselection* algorithm assigns 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* cheks the **community creation** pre-condition and establishes automatically a new community by adding a new vertex to the CG graph. Accordingly, the ***FollowersSelection*** algorithm is executed selecting *followers* for r_8 which assumes the role of leader in this new community.

Dismantling a Registry. The community c_3 is composed by only two registries: r_3 the *leader* and r_7 its *follower* (Figure 7). 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 automaticaly dismantled** since the dismantling pre-condition is satisfied.

Updating Registry Functionalities. The ***CommunityAcceptance*** algorithm is tested through an update of registry functionalities f. The update is done if a service provider publish , unpublish or modify a WS 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 $\mathtt{th} = 0.1$, 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 pre-condition** 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 $\xi = 0.2$ (Section 3.3). 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 managing operations used to handle registry and community life cycles. In fact, they execute well and automatically call each other after every change.

6 RELATED WORK

To enhance the WS discovery process in a private registries network, we use a functionality-driven approach to organize them into communities. Such a registry network organization needs to be regularly managed to ensure the consistency of the communities. In this paper, we are interested in the management phase of registry communities. As far as we know, this work is the first attempt to manage communities of WSs registries. Indeed, several WS discovery approaches in distributed registry environments (Sellami et al., 2010c; Sivashanmugam et al., 2004; Xu and Chen, 2007) structure their networks as clusters but did not provide management mechanisms for their clusters. In this section, we overview some related efforts in the field of managing communities that helped us tailor our approach.

In (Paik et al., 2005), Paik et al. present the WS-catalogNet framework allowing to group e-catalogs having similar domain into communities, build relationships between them and manage them constantly. 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 pre-conditions and effects for each operation based on the graph-based model they have defined (Paik et al., 2002). However, authors employ a classification technique to organize communities, while we use a dynamic clustering one. Furthermore, we use the functionality criterion to structure communities rather than business domain.

Medjahed and Bouguettaya (Medjahed and Bouguettaya, 2005) propose an approach to organize WSs into communities depending on their domain of interest. A community is an instance of an ontology metadata 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. However, their operations are described informally compared to our management operations.

In (Maamar et al., 2009), Maamar et al. discuss the dynamic nature of WS community and focus on potential conflicts. They propose in (Subramanian et al., 2007) an approach to engineer WSs communities in order to reconcile these potential conflicts. This approach is based on the Community development protocol which is interested in managing communities in term of attracting and retaining WSs, creating and dismantling communities. Similarly to our approach, communities are organized according to WSs functionalities. However, this approach did not propose a model and their operation descriptions are rather informal.

7 CONCLUSIONS

In this paper, we proposed an approach for managing communities of WSs registries. We first defined a model to facilitate the managing step. Then we identified the main steps of registry and community life-cycles. Afterwards, we specified managing operations and algorithms based on the model that we have proposed. Finally, we implemented a *Community Manager* to test and validte these algorithms and operations using graph simulation. Experiments show that our algorithms and managing operations execute well. The splitting operation has not been tested since its pre-condition is not yet specified. Indeed, we used different methods to detect the sparsity criterion in a given community such as standard deviation of a statical series but each one represents an exception. As part of our short term perspectives, we plan to specify a pertinent pre-condition for splitting operation. Furthermore, we foresee to implement these algorithms on top of the platform P2P *JXTA* (JXTA, 2004) in order to test the precision/time ratio of our approach.

REFERENCES

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

Bondy, J. A. and Murty, U. S. R. (2007). *Graph Theory*. Springer London.

Chabeb, Y., Tata, S., and Ozanne, A. (2010). Yasa-m: A semantic web service matchmaker. In *AINA 2010, April 20-23, Perth, Australia*.

Ding, C. H. Q., He, X., Zha, H., Gu, M., and Simon, H. D. (2001). A min-max cut algorithm for graph partitioning and data clustering. In *ICDM '01*, pages 107–114, Washington, DC, USA. IEEE Computer Society.

Jgrapht (2003). Jgrapht. http://www.jgrapht.org/. http://www.jgrapht.org/.

JXTA (2004). Jxta. https://jxta.dev.java.net/.

Lausen, H. and Farrell, J. (2007). Semantic annotations for WSDL and XML schema. W3C recommendation, W3C. http://www.w3.org/TR/2007/REC-sawsdl-20070828/.

Maamar, Z., Lahkim, M., Benslimane, D., Thiran, P., and Sattanathan, S. (2007). Web services communities - concepts & operations -. In *WEBIST'2007*.

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

McConnell, J. J. (2008). *Analysis of algorithms: an active learning approach*. Jones and Bartlett publishers.

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

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

Sellami, M., Bouchaala, O., Gaaloul, W., and Tata, S. (2010a). WSRD: A web services registry description. In *NOTERE'10, May 31- June 2, Tozeur, Tunisia*.

Sellami, M., Gaaloul, W., and Tata, S. (2010b). Functionality-driven clustering of web service registries. In *SCC 2010, Miami, Florida, USA*.

Sellami, M., Gaaloul, W., Tata, S., and Jmaiel, M. (2010c). Using recommendation to limit search space in web services discovery. In *AINA*, pages 974–981. IEEE Computer Society.

Sivashanmugam, K., Verma, K., and Sheth, A. P. (2004). Discovery of web services in a federated registry environment. In *ICWS'04, San Diego, California, USA*, pages 270–278. IEEE Computer Society.

Subramanian, S., Thiran, P., Maamar, Z., and Benslimane, D. (2007). Engineering communities of web services. In Kotsis, G., Taniar, D., Pardede, E., and Ibrahim, I. K., editors, *iiWAS*, volume 229 of *books@ocg.at*, pages 57–66. Austrian Computer Society.

Xu, B. and Chen, D. (2007). Semantic web services discovery in p2p environment. In *ICPPW '07*, page 60. IEEE Computer Society.

Zhang, H. and Shin, D.-G. (2005). Objective function based fuzzy clustering. Technical report, University of Connecticut.