Semantic Wikis Distributed on Structured Peer-to-Peer Networks

Charbel Rahhal*, and Houssam Yactin†

*Lebanese University, Faculty of Sciences
E-mail: charbelrahhal1@gmail.com

†HASLab-INESC TEC & Minho University
E-mail: hayactin@inesctec.pt

***Abstract*—P2P Semantic wikis were initially developed as collaborative editors with two main goals in mind: (1) ensure a massive collaborative editing in semantic wikis with low maintenance cost where thousands of users can participate in the wiki edition and (2) provide fault tolerance for semantic wikis. Their goals were attained through the adoption of an optimistic replication of their data (semantic wiki pages and annotations) on a set of interconnected semantic wiki servers. The edition of the semantic wiki pages and the execution of semantic queries are now distributed on different peers. Two Peer-to-Peer Semantic Wikis were developed, Swooki and DSMW. They are based on unstructured P2P network where each peer knows its neighbors without any knowledge about their contents. In spite of their advantages, Swooki will face scalability problem regarding the size of data that it is dealing with since it uses a total replication of the entire wiki pages and the triples store, it will be only suitable for small wikis, and DSMW semantic queries may return different and incomplete results since every peer has its own semantic wiki pages and annotations. In this paper, we propose to handle the problems of unstructured P2PSW by building Structured P2P semantic wiki (SP2PSW).The research work proposes all the possible approaches to build SP2PSW, evaluates each approach based on many standard evaluation metrics, derives the optimal one and develops the needed algorithms. At the end, a comparison with the existing unstructured P2PSW (Swooki and DSMW) shows the marked improvement of SP2PSW over Unstructured P2PSW.**

***Index Terms*—Semantic Web, Semantic Wikis, P2P Wikis, P2P Semantic Wikis, Unstructured P2P, Structured P2P**

I. Introduction

P2P Semantic Wikis (P2PSW) were initially developed as collaborative editors with two main goals in mind: (1) ensure a massive collaborative editing in semantic wikis with low maintenance cost where thousands of users can participate in the wiki edition and (2) provide fault tolerance for semantic wikis. The Semantic Web [1] technologies used in P2PSW, improves the navigation, the search, and the knowledge extraction in the wikis. The semantic annotations in the wiki pages can be processed automatically by machines and they are exploited by semantic queries using special languages such as SPARQL [2]. The queries can insert, update, delete and search RDF triples [3]. P2PSW goals were attained through the adoption of an optimistic replication of their data (semantic wiki pages and annotations) on a set of interconnected semantic wiki servers. The edition of the semantic wiki pages and the execution of semantic queries are now distributed on different peers. The failure of a semantic wiki peer will not affect the P2PSW work since the data will be available on other peers. Down or disconnected peers can resynchronize with the other peers after a recovery or a reconnect. P2PSW were built upon unstructured P2P networks. These wikis do not impose a particular structure on the overlay network by design, but rather are formed by semantic wiki peers that randomly form connections to each other. A peer can join and leave the network dynamically.

Nowadays, two P2PSW were built: Swooki and DSMW. Swooki [4] adopts a total replication of data. When a semantic wiki page is created on a peer, the page will be replicated on all the other peers of the network. In addition, changes on a peer are first integrated locally, then propagated through the network, and eventually received and integrated by the other peers. However, by employing a total replication i.e. every peer will host the entire wiki system, Swooki will face scalability problem regarding the size of data that it is dealing with. It will be only suitable for small wikis. The scalability limitation occurs for peers using devices with limited storage capacity and processing units as shown in Figure 1. In this figure, the peer (z) represents a smart phone from which the user is accessing the P2PSW. This peer has a limitation in storing data and so it is difficult to replicate the entire data in the P2PSW. On the other, since Swooki adopts a total replication and frequent changes can occur so there is a huge traffic of messages representing the changes are exchanged and used in the synchronization of the peers which reduces the efficiency.

Distributed Semantic MediaWiki (DSMW) [5] adopts a partial replication of data. In DSMW, the collaborative edition among the users is based on a publish/subscribe model. A user can choose with whom to collaborate and a Friend-Of-Friend network can be build. When a user on a peer creates a semantic wiki page, he can decide when to publish the page and its changes in special feeds. Users on other peers can subscribe to these remote feeds, retrieve their contents and integrate them locally. Different peers will have different content unless they integrate same changes. The search for all the Europe countries

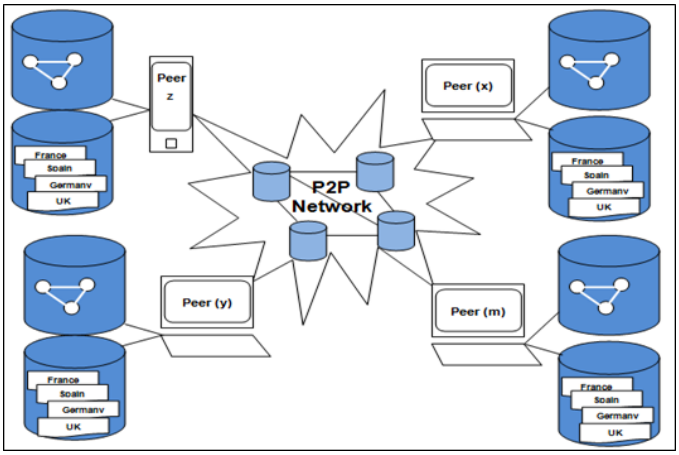

Fig. 1. Swooki with Total Replication problem

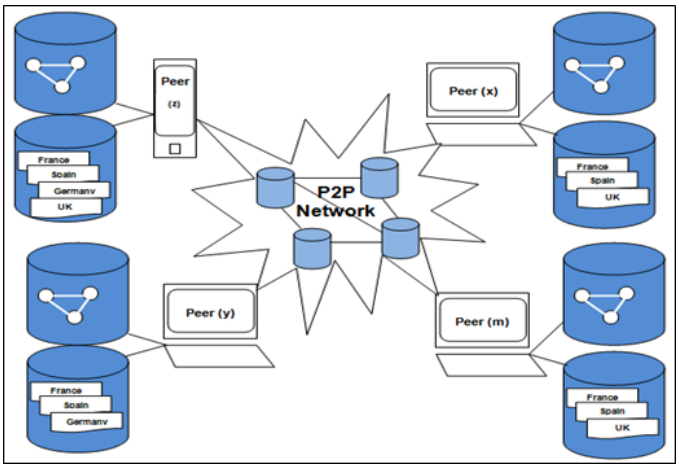

Fig. 2. DSMW structure with partial Replication

on two different peers through semantic queries will return two different lists of countries since every peer has its own semantic wiki pages and annotations. By employing a partial replication, different semantic pages for the same domain of interest could be created on different peers without a previous knowledge about their existence by the wiki users. Duplication of same data could occur and a waste of time and effort can take place.

For example in Figure 2, the execution of a query to search for all countries of Europe, the query will be executed locally on each peer, which is an advantage (Quick response time), but different results may occurr. The peer(y) doesnt store the new page UK inserted by peer(x) while executing the same query on peer(x) will not return the new page Germany inserted by peer(y). So incomplete queries results with data incoherency is the major problem of DSMW even that the data is stored on different peers of the network. To improve such approach, a search mechanism is needed that allows peer to search the P2P network and retrieve the needed data. However, such a mechanism will have a very low efficiency since DSMW is based on unstructured P2P network, where a peer does not know anything about stored data on other peers. All these problems and more, lead us to search for a solution using a Structured P2P Semantic Wiki (SP2PSW).

In this paper, we propose another way of building P2PSW by distributing Semantic Wikis on structured P2P networks. This architecture will provide a high storage capacity, data availability through a partial replication and a good performance for P2PSW. Building the first SP2PSW raised many questions:

1) What is the best way to store and replicate semantic wiki pages in the structured overlay network?
2) How to replicate the semantic stores that contain the semantic annotations of the pages? Should every peer has its own semantic store or a special architecture is required?
3) How to synchronize the changes in the replicated wiki pages and the annotations? How to ensure their consistency? The replication of the semantic stores in the structured network is crucial and will affect the result of semantic queries. For instance, the search for all the countries in Europe requires the access to all the peers that replicate pages about those countries which is not an easy task.

The P2PSW that we build answers most of these questions in an optimal way. The proposed solution will be evaluated using different metrics such as storage capacity, performance, and the accuracy of queries result. The paper is organized as follows: section 2 presents some major metrics used in the comparison of P2P networks. Section 3 details the proposal. Section 4 introduces the algorithms we derived in building SP2PSW. Section 5 presents an analysis and a validation of our approach. The last section concludes the paper and discusses future works.

II. Metrics for comparing P2P networks

P2P networks can be classified based on how the nodes in the overlay network are linked to each other into three categories: (1) unstructured when the overlay links are established arbitrarily, (2) structured that use Distributed Hash Table (DHT), set and get functions and in which every node is responsible for a specific part of the network content, and (3) hybrid between client/server systems and pure networks in which some peers called super-peers act as dedicated servers for some other peers.

In [6] they specify that from the perspective of data management, the main evaluation metrics of a P2P network are five presented below. The Table I shows the value of each evaluation metric for every P2P network. We use these metrics to evaluate our SP2PSW presented in the next section.

i) **Autonomy**: an autonomous peer should be able to join or leave the system at any time without restriction. It should also be able to control the data it stores and which other peers can store its data, e.g. some other trusted peers.
ii) **Query expressiveness**: the query language should allow the user to describe the desired data at the appropriate

TABLE I
COMPARISON BETWEEN P2P NETWORKS

Requirements	Unstructured	Structured	Super-Peer
Autonomy	High	Low	Moderate
Query expressiveness	High	Low	High
Efficiency	Low	High	High
Quality of Service	Low	High	High
Fault tolerance	High	High	Low

level of detail. The simplest form of query is key look-up which is only appropriate for finding files. Keyword search with ranking of results is appropriate for searching documents. But for more structured data, an SQL-like query language (SPARQL) is necessary to let the user to execute all the types of queries such as atomic query, disjunctive or range queries, conjunctive queries, etc.

iii) **Efficiency**: the efficient use of the P2P network resources (bandwidth, computing power, storage) should result in lower cost and thus higher throughput of queries, i.e. a higher number of queries can be processed by the P2P system in a given time.

iv) **Quality of service**: refers to the user-perceived efficiency of the P2P network, e.g. completeness of query results, data consistency, data availability, query response time, etc.

v) **Fault-tolerance**: efficiency and quality of services should be provided despite the occurrence of peers failures. Given the dynamic nature of peers which may leave or fail at any time, the only solution is to rely on data replication.

III. SEMANTIC WIKIS DISTRIBUTED ON STRUCTURED P2P NETWORKS

This section describes our proposal and it is structured as follows. First, we propose main aspects to consider in building a structured P2PSW. Then, we propose different alternatives to build a structured P2PSW. An evaluation showing the advantages and disadvantages of every proposed solution concludes every proposal. For every proposed solution, we emphasis on the replication of the semantic wiki pages, the replication of the semantic stores, and the synchronization mechanism. Finally, we derive and adopt the best approach to build a structured P2PSW based on the important metrics presented in the previous section.

A. Main Proposed Aspects to build Structured P2PSW

A SP2PSW should enable end users to search for wiki pages, view and edit the semantic wiki pages and their annotations, and execute semantic search queries. Overall, we propose four main aspects to be taken into consideration while designing our SP2PSW system:

1) Wiki Pages storage and replication: How peers will store their wiki pages and where to replicate these pages?
2) RDF triplestores replication: How to replicate the triplestores and ensure consistency between these replicas?
3) Query processing: How can we take advantage of P2P search mechanisms to efficiently query and retrieve RDF data in large scale settings? Moreover, whenever a query, composed of a set of sub-queries, can be answered by several nodes, how to efficiently combine RDF data residing in different locations before sending the result back to the user?
4) Network dynamicity: What about the DHT dynamicity of join and leave of peers?

B. Proposed Alternatives for building a SP2PSW

In order to answer the previous questions, we propose different approaches for building SP2PSW. We discuss the strength points and weak points of each proposed approach taking into consideration the three following criteria: (1) the data storage capacity for wiki pages and their annotations, (2) the efficiency and (3) execution accuracy of different types of queries. These criteria were previously detailed in section 2. We studied many research works on RDF data storage and retrieval in structured P2P systems like [12], [10], [7] and [11], others concerned with the replication of data on P2P networks like [6], and P2P wikis like [9] before proposing our approaches.

In the different approaches we aim to build, the storage of the wiki pages on the peers of the DHT and their replication will follow the same method. After its creation, the wiki page URI that we call $WikiPageSubject$ is hashed and it will be stored on the peer responsible for that key with initial content using the dht.put(getHash(WikiPageSubject), initialContent) method. In fact, the initial content is a set of insert line operations where the first operation is create a wiki page. As in any DHT, every peer is responsible for a key space to which the wiki pages are mapped. Like this, every peer will be responsible for a part of the wiki content. So in the next proposed approaches, we will focus only on the RDF triple storage and the replication of the triplestores.

The replication of the semantic wiki pages in all the approaches will be the same. Different hash functions will be used to replicate a wiki page on many peers of the DHT. To handle concurrent updates of the same wiki page by multiple peers, an optimistic replication algorithm can be used to integrate these changes in that page. A change in a wiki page is expressed as two types of operations: an insert or a delete line. In addition, this replication algorithm will synchronize the replicas of a wiki page on the peers of the DHT replicating the same page and ensure the convergence of the page replicas. Any synchronization algorithm that ensures the CCI model can be adopted such as WOOT, WOOTO, or Logoot [8], etc.

1) Centralized RDF Triples Store: In this approach, different peers will replicate different pages and only one centralized triplestore is used for storing all the annotations of the entire wiki. This approach is illustrated in figure 3 (part 1).

In this approach, when a user searches for a semantic wiki page, there are two possible cases: either the page does not exist and the user will have to create it or the page exists and the user will be able to view its content. Since it is a structured P2PSW, the page is retrieved by its name which represents the page identifier. Actually, the page

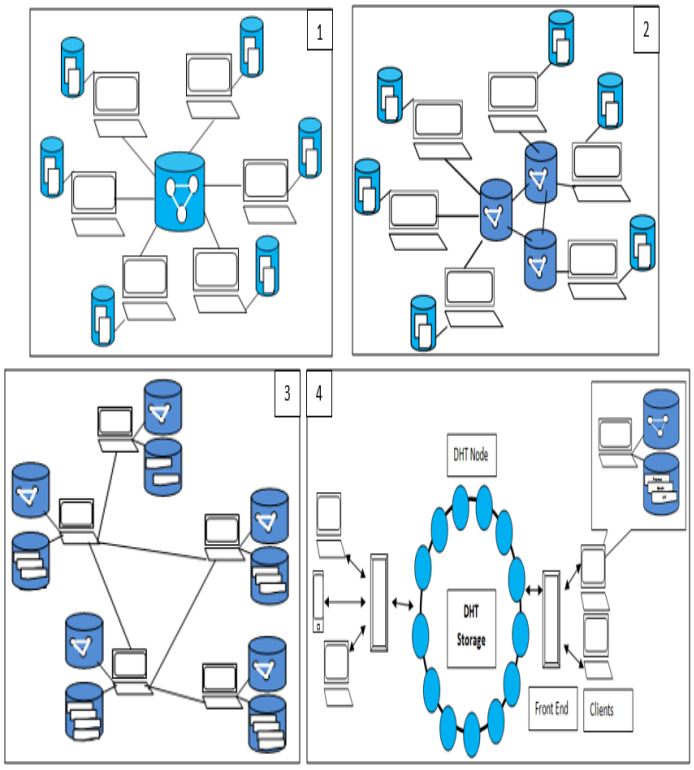

Fig. 3. Different approaches for building SP2PSW

identifier is hashed by some hash functions and the result is retrieved. When a user edits and saves a semantic wiki page, the changes generate the operations: delete and insert lines. These operations will be received and integrated by the peers replicating this page. On the other hand, the triplestore will be updated in case of the semantic wiki page contains annotations. The semantic annotations in the saved page will be extracted and transformed into RDF triples. The previous annotations of that page will be deleted and replaced by the new ones. If the user decides to run a semantic query using SPARQL in a special semantic wiki page, the search will be made on the central triplestore and the result of the query will be complete.

This approach offers a good solution for scalability since every peer stores only its own semantic wiki pages. It adopts a partial replication. The modifications on a peer will be immediate and affects only the peers replicating this page. This approach provides also a good solution for the semantic queries processing. Every query will be executed immediately on the central triplestore independently of the query form such as queries that contains many subqueries or need to generate a dynamic web query result. For instance, the execution of a query that "Find all countries of Europe", will be executed directly which will reduce traffic and increase the performance. In spite of its advantages, this approach has offers a major problem which is data availability and performance. Only one central triplestore will handle all the requests for semantic queries and in case of a failure the semantic annotations will not be available (single point of failure).

2) Replicated Centralized RDF Triples Store: This approach is similar to the previous one with only one difference that the centralized triplestore will be replicated on many peers as shown in figure 3 (part 2). Consequently, it will solve the single point of failure problem. In this solution, synchronizing the replicated triplestores is required which generates high amount of traffic in the network. On the other hand, more storage capacity and maintenance of the triplestores is needed.

3) Total Replication of RDF triples store: This approach as shown in figure 3 (part 3) consists of storing all the RDF triples of all the wiki pages of the entire wiki locally on each peer. Similar to the previous solution, when a user edits and saves a semantic wiki page, these operations will be received and integrated by the peers replicating this page. On the other hand, the triplestore will be updated locally in case of the semantic wiki page contains annotations and a synchronization mechanism will be fired to update all the triplestores for all the peers in the network. If the user decides to run a semantic query using SPARQL in a special semantic wiki page, the search will be made on its own local triplestore and the result of the query will be complete.

In this approach, the execution of semantic queries of any type will be executed locally on the peer triplestore which returns complete results with high performance in searching time. Since RDF triplestore is automatically replicated over all the peers in the network, so there is no single point of failure. This solution will greatly limit the scalability since all the triples in the wiki are replicated on every peer which requires highest storage capacity on every peer. When a change in a triplestore of a peer occurs, a DHT Broadcast mechanism is automatically applied; it broadcasts the changes to all the peers in the network. Since changes in the triplestores are frequent, the synchronization of the triplestores will occur with high rate and consequently a high traffic will flood the network. This solution is similar to Swooki in totally replicating the triplestores without obtaining benefits from the Structured P2P network architecture.

4) Partial Replication of RDF triplestore: In this approach, every peer stores some semantic wiki pages and only their annotations are stored in the local triplestore. When a user edits and saves a semantic wiki page, the delete and insert operations will be received and integrated by the peers replicating this page. On the other hand, the triplestore will be updated in case of the semantic wiki page contains annotations. The previous annotations of that page will be deleted and replaced by the new ones locally on the same peer. In case of a peer decides to leave the network, its wiki pages and the content of its triplestore will be replicated to its closest neighbors peers using two different synchronization mechanisms, one for the wiki pages and the second for the triplestores.

The execution of a semantic query in an unstructured P2PSW may lead to pass across all the peers of the network to return a specific result. For instance, the execution of the query "Find all the Countries of Europe" will pass through all the peers of the network which highly increases the traffic

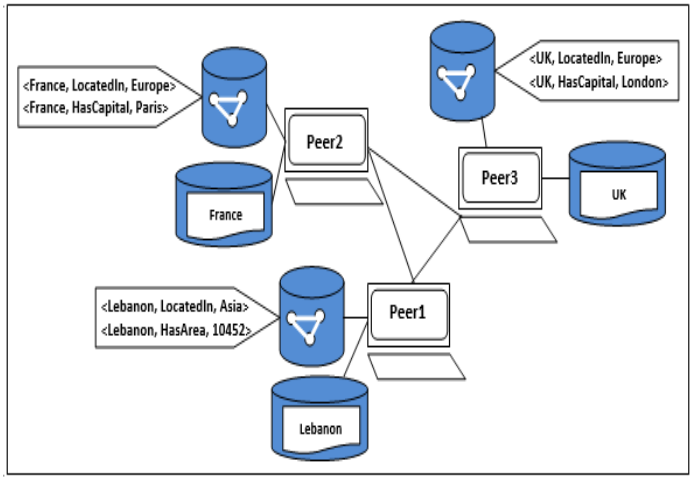

Fig. 4. Incomplete query result case

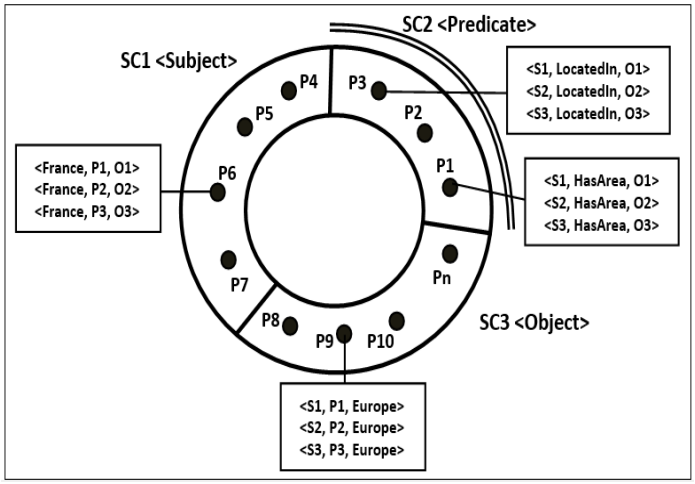

Fig. 5. SP2PSW Clustering design

messages over the network. The traffic comes from the sent message to all the peers and the returned results which decreases the performance, responsiveness and increases the search time. A possible solution to this problem can be made using an expiry time (TTL) to terminate the execution of a query if it exceeds the specified TTL. The result will be a less traffic however incomplete or missing results of queries can occur even if the triples are available in the Wiki system.

The partial replication of RDF triplestore approach is shown in figure 3 (part 4). If the user decides to run a semantic query in a special semantic wiki page, the search will be made first locally on the requested peer. For example, the search for "Find the area of Lebanon" can be done locally if the peer contains the page Lebanon. Otherwise, some hash functions on the triples are needed and will be used to generate a set of keys. The DHT method get (Key) $\rightarrow$ value will return only one node id for each used hash function. This will lead to find nodes that should contained the requested annotations. However, this peer may not have this annotation since each peer stores only the annotations of its own pages. For instance, if the peer1 executes a semantic query to find the area of Lebanon, it will be executed locally and the result will be found. While the execution of a query to find all countries located in Europe will use a HashFunction (LocatedIn) and return a value for a specific peer, then the search will be directed to this peer (peer2 or peer3), so incomplete query result will be obtained as shown in the figure 4.

5) Clustering of RDF triplestore: In this proposed solution, each peer contains some wiki pages and a triplestore. Peers having the same semantics are arranged into the same semantic cluster as shown in figure 5. Peers P4, P5, P6, and P7 compose the semantic cluster1 in which the RDF triples are hashed based on their <subject> and stored in the corresponding peer. P1, P2, and P3 compose the semantic cluster2 where hashing is made on <predicate> and the peers P8, P9, P10, and Pn compose the third semantic cluster using <object>. In this approach, when the annotations of a page are updated, only the peers of the appropriate semantic clusters concerned with these annotations will be updated. If the user runs a semantic query using SPARQL, the appropriate hash function will be applied based on the form of query whether it is atomic, range, or conjunctive. For example, when the query searches the pattern {"Lebanon", hasArea, ?value}, the hash function on the subject Lebanon will be used.

In case of conjunctive queries, many peers in the same semantic cluster can be used at the same time. For example to execute the query "Find all countries located in Europe and they have a common border with Asia", the patterns of the query look like this: < ?country , LocatedIn , "Europe"> ^<?country , HasBorder , "Asia">. The query will search for RDF triples containing two different predicates LocatedIn and HasBorder which may be found on one or two different peers using the same hash function on <predicate>.

Other queries may have a more complex form. For example the query "Find all countries in Europe and having any relation with Pizza" has the following two patterns: <?country, LocatedIn , "Europe"> ^<?country, ?anyPredicate, "Pizza">. The query will use two different hash functions, the first hash function is based on the predicate LocatedIn and the second is based on the object "Pizza". The result will come from two different peers belonging to two different semantic clusters. The final result of the query will be a combination of both results.

The benefits of this approach are many. The RDF triples are distributed over a limited set of peers, which means that there is no need for high storage capacity. Consequently, it ensures a high scalability. The execution time of a semantic query is low, because only the peers concerned with the used hashed function will be accessed directly which will reduce the traffic messages over the entire network and increase the performance. In addition, the result of the query will be complete and the benefits of structured P2P network will be totally reached. This approach has some limitations: (1) in case of a failure of one or some peers, all their stored RDF triples

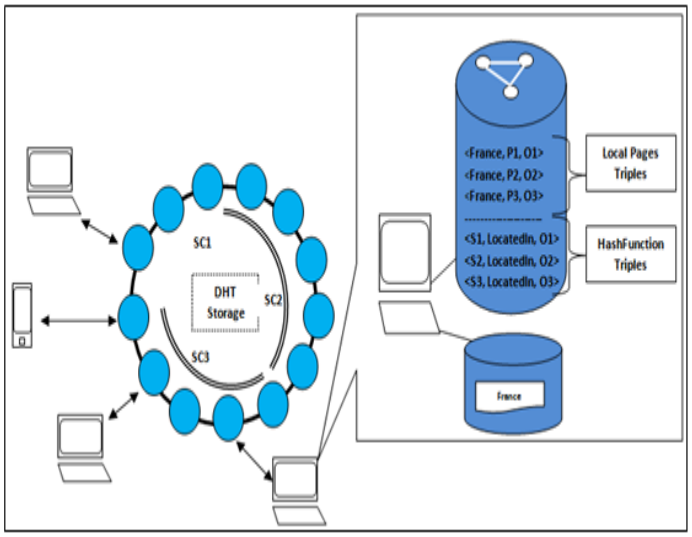

Fig. 6. The Optimal approach to build SP2PSW

will be unavailable, and (2) there will be an inconsistency between the wiki pages on a peer and the triples stored on this peer.

6) The Optimal Proposed Approach of SP2PSW: In conclusion, we proposed all the previous possible solutions for building the first structured P2P Semantic Wiki, and identified their advantages and limitations. Based on this, we propose now the optimal solution of SP2PSW. It will be a combination between the last two proposed approaches as shown in figure 6. In this solution, every peer will store a set of wiki pages. As we said earlier at the beginning of the section, the URI or name of a page is hashed and based on the result the page will be stored on the appropriate peer. Every peer has a triplestore that contains the annotations of these pages. In addition, to storing the triples locally, these triples will be hashed usually using three different hash functions based on <subject>, <predicate>, and <object> and stored in different semantic clusters. Moreover, to ensure availability of the triples, many hash functions are used to replicate these triples on additional peers.

So as a final result, this approach is the best solution since it offers all the following features:

- **High scalability** since wiki pages and the RDF triples of the entire wiki are partially replicated over all the peers in the network, so a peer needs only to store a subset of wiki pages with the triples of these pages and a subset of RDF triples used by the put method of the hash function.
- **High Quality of Service** in execution of any type of semantic queries, since all the triples are always available, so the results of the queries are always complete.
- **High Fault tolerance** since there is no single point of failure, all the wiki pages and the triples are replicated on many different peers by using many different hash functions for each semantic cluster.
- **High efficiency** since the usage of DHT storage will directly route the semantic query to the corresponding peers, which will reduce the traffic over the network.

We developed the necessary algorithm for this optimal approach to build the first SP2PSW shown in the next section.

IV. Algorithm for building Structured P2PSW

This section presents the algorithm we developed to build the first SP2PSW. The algorithm is written in pseudo-code and can be easily implemented in any programming languages based on the SP2PSW that we aim to build. For each operation made on the Wiki, we detail the resulted actions and their effects. These operations are (1) view/search a page, (2) edit a page, (3) save a page, and (4) run a semantic query. So any proposed SP2PSW should offer these services in optimal way.

A. On Wiki Page View

When a user decides to view a wiki page there are two ways either using a search tab or by clicking on a link on another wiki page. First, the request for a page is hashed then sent to the corresponding peer, there are two possible results:

1) Page found
   a) Read (no modification)
   b) Edit
2) Page does not exist
   a) Create a new semantic wiki page
   b) Do nothing.

function OnView(WikiPageSubject)
  $Page \leftarrow dht.get(getHash(WikiPageSubject))$
  if Page == NULL **then**
    return "$WikiPageNotFound$!
    $WouldYouLikeToCreateit$!"
  else
    $wikiContent \leftarrow extractContent(Page)$
    $htmlContent \leftarrow ShowHTML(wikiContent)$
    return $htmlContent$
  end if
end function

B. On Wiki Page Edit

When a user wants to edit a page, he clicks on the edit tab and the plain text content of the page is extracted and displayed.

function OnEdit(WikiPageSubject)
  $Page \leftarrow dht.get(getHash(WikiPageSubject))$
  $wikiContentText \leftarrow extractContent(Page)$
  return $WikiContentText$
end function

C. On Wiki Page Save

When a user edits a page and saves, the corresponding operations are generated using a diff algorithm. Then these operations are sent to the peers replicating the page and integrated. On the other hand, the annotations of the page are extracted, transformed into triples and inserted in the appropriate peers in addition to the peers of the page.

function OnSave(WikiPageSubject, newWikiPageContent)
  $oldPage \leftarrow dht.get(getHash(WikiPageSubject))$
  $oldWikiContent \leftarrow extractContent(oldPage)$
  $Ops[] \leftarrow diff(oldWikiContent, newWikiPageContent)$ ▷ The generated operations will be sent to the destination peer

$dht.put(getHash(WikiPageSubject), Ops[])$ ▷ The received operations are integrated using any optimistic replication algo
SYNCHRONIZEPAGES()
▷ Update the triple store if necessary
$Annotations[] = extractAnnotations(newWikiPageContent)$
if Not (empty(Annotations)) **then**
$oldAnnotations[] = extractAnnotations(extractContent(oldPage))$
$oldTriples[] = convertTo(oldAnnotations)$
$Triples[] = convertTo(Annotations)$
▷ Previous triples of Page will be replaced by the new Ones
$updateTripleStore(getHash(WikiPageSubject), Triples[])$
▷ Delete old triples from the Semantic Clusters
for each triple in oldTriples[] **do**
$Delete(dht.get(gethashSubject(triple), triple))$
$Delete(dht.get(gethashPredicate(triple), triple))$
$Delete(dht.get(gethashObject(triple), triple))$
end for
▷ Insert new triples into the Semantic Clusters using a triple store synchronization mechanism
for each triple in Triples[] **do**
$Insert(dht.get(gethashSubject(triple), triple))$
$Insert(dht.get(gethashPredicate(triple), triple))$
$Insert(dht.get(gethashObject(triple), triple))$
end for
end if
end function

The triplestore of a peer and those of the clusters are updated only if they were changes in the annotations of the wiki page i.e. Triples[] $\neq$ oldTriples. In order to reduce the traffic that could be generated from the update of the triples in the clusters. Instead of deleting all the previous triples in the old page and adding all the triples found in the new saved page, we can compute a reduced set of triples to insert IS = Triples[] $-$ oldTriples[] and a reduced set of triples to delete DS = oldTriples[] $-$ Triples[]. The triplestores of clusters will be updated using these two sets IS and DS.

D. On Semantic Query Execute

When a user executes a semantic query two possible cases can take place:

1) First case, if the user is working offline, the query is executed locally and searches only in the peer triplestore.
2) Second case, if the user is working online:
   a) If the semantic query has the following pattern (?s; ?p; ?o) then call DHT Broadcast to retrieve all RDF triples.
   b) Else if the query has a different atomic pattern than the previous one some hash function(s) are used and generate the key(s) corresponding to the known <subject>, <predicate> and <object> of the query pattern. Next, the DHT GET(Key) method is used and returns the peer Id for each used hash function, then the query will be routed to the corresponding peer. In case of more than one peer having the result, we give a priority to hashSubject function over hashPredicate over hashObject to reduce the traffic.
   c) Else if the query has the conjunctive form, for each sub query, the generated key will route the search to the corresponding peer, get the triples stored on this destination peer, combine all the retrieved triples. Finally show the query result on the screen.

function ONQUERY(SPARQLstmnt)
if SPARQLstmnt haspattern("(?S,?P,?O)") **then**
BRAODCASTDHT(AllRDF)
else
$NeededRDF[] = \{\}$
for Each AtomicSubQueryPattern **do**
$DestinationPeer \leftarrow dht.get(getHash(AtomicSubQueryPattern))$
$NeedeRDF[] \leftarrow NeededRDF[]U$
$LoadTriples(AtomicSubQueryPattern, DestinationPeer)$
end for
end if
return $showQueryResult(SPARQLstmnt, NeededRDF[])$
end function

TABLE II
COMPARISON TABLE BETWEEN SP2PW AND P2PSW

	Evaluation Metrics				
	Peer's RDF Storage	Scalability	Query QOS	Fault Tolerance	Control Messages Traffic
SWOOKI	Low	Low	V.High	V.High	V.Low
DSMW	High	High	Low	High	Low
SP2PSW	High	V.High	V.High	V.High	High

V. RESULTS ANALYSIS AND VALIDATION

We derived an optimal approach to build a semantic wiki distributed on structured P2P network (SP2PSW), after a deep study and evaluation of the previous approaches we proposed. A comparison between our approach and the existent unstructured P2PSW (Swooki and DSMW) shows that our approach offers much enhancements on different levels. The comparison is made based on the evaluation metrics is presented in the Table II and in a chart (see figure 7). The measure Low in the table denotes bad, very low denotes worst, high and very high denote better and best. In Swooki, there is a total replication of both wiki pages and the triplestores. This requires a huge storage for data and generates a lot of traffic to synchronize the wiki pages and the triplestores on all the peers of the network. Handling the semantic queries and the failure of some peers is very high. In SWMW, there is a partial replication of the pages and the triplestores. A user on a peer can decide to whom replicate his pages. When the peers replicating the same page are down. In this case, these pages will be unavailable. Running queries is always local on a peer and incomplete results may be obtained. since only few peers replicate, the exchange of messages among these peers is moderate. In our approach, there is a partial replication of data. A peer stores only a set of wiki pages and their annotations, plus the annotations hashed to it. Newly created data will be distributed on the DHT. The results of the queries are always complete; the precise peers storing the annotations are quickly located. The failure of a peer will not affect the functioning of the SP2PSW.

VI. CONCLUSION

This section concludes the work by pointing to perspectives and future works. The research work conducted focused on

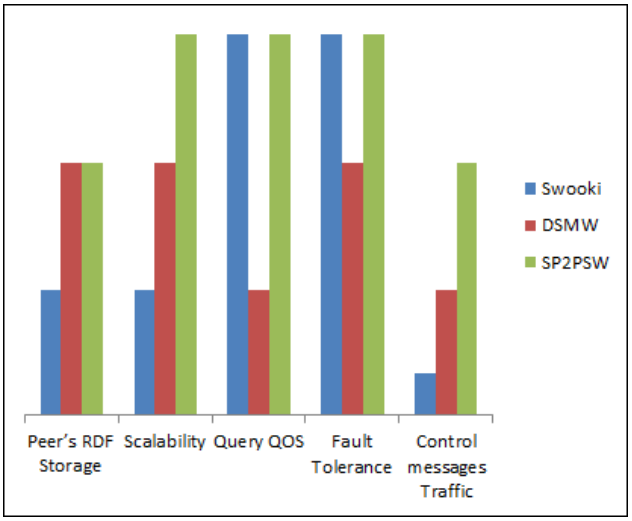

Fig. 7. Comparison Chart between SP2PSW and (Swooki and DSMW)

development of the first Semantic Wiki based on Structured Peer-to-Peer network. Initially, we studied a wide range of concepts such as the Semantic Web, Semantic Wikis, P2P networks (unstructured, structured and hybrid), and P2P Semantic Wikis. The development of our approach consisted on many steps: (1) propose all the possible approaches for building SP2PSW, evaluate each one and select the optimal one of them, (2) develop the essential algorithms needed for the basic operations of the SP2PSW i.e. browse and edit wiki pages, edit annotations and search RDF triples, (3) compare between our proposed SP2PSW and the existing unstructured P2PSW (Swooki and DSMW). In Conclusion, the development of the first SP2PSW solved many existing problems found in the currently used unstructured P2P semantic wikis (Swooki and DSMW) such as low scalability with respect to data storage, needs to high storage capacity and incompleteness of queries results. As future work, it would be interesting to implement and build SP2PSW, testing it in a real structured peer to peer large scale network and make real world evaluations and measurements based on real data. As future work also, we will study the dynamicity of the network and how joining or leaving the network affects the replication of data.

REFERENCES

[1] Tim Berners-Lee, James Hendler, and Ora Lassila, "The Semantic Web", Scientific American Magazine, pp. 34-43, May 2001.

[2] Steve Harris, and Andy Seaborne, "SPARQL 1.1 Query Language", https://www.w3.org/TR/sparql11-query/ W3C recommendatio, March 21, 2013.

[3] Patrick J. Hayes, and Peter F. Patel-Schneider,"RDF 1.1 Semantics", https://www.w3.org/TR/rdf11-mt, W3C Recommendation 25 February, 2014.

[4] Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli, "Peer-to-Peer Semantic Wikis", In DEXA09: 20th International Conference on Database and Expert Systems Applications, pp. 196-213, September 2009.

[5] Charbel Rahhal, Hala Skaf-Molli, Pascal Molli, and Stéphane Weiss, "Multi-synchronous Collaborative Semantic Wikis", In WISE'09: 10th International Conference on Web Information System Engineering, pp. 115-129, October 2009.

[6] Vidal Martins, Esther Pacitti, and Patrick Valduriez,"Survey of data replication in P2P systems", research report, INRIA, pp. 1-45, 2006.

[7] Min Cai, and Martin R. Frank, "RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network", WWW'04: 13th International Conference on World Wide Web, ACM, New York, USA, pp. 650-657, May, 2004.

[8] Stéphane Weiss, Pascal Urso, and Pascal Molli, "Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks", 29th IEEE International Conference on Distributed Computing Systems (ICDCS'09), Monteral, Québec, Canada, pp. 404-412, 22-26 June 2009.

[9] Gérald Oster, Pascal Molli, Sergiu Dumitriu, and Rubén Mondéjar, "UniWiki: A Collaborative P2P System for Distributed Wiki Applications", 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE'09, Groningen, Netherlands, pp. 87-92, 29 June - 1 July 2009.

[10] Imen Filali, Francesco Bongiovanni, Fabrice Huet, and Françoise Baude, "A Survey of Structured P2P Systems for RDF Data Storage and Retrieval", Trans. Large-Scale Data- and Knowledge-Centered Systems journal, volume 3, pp. 20-55, 2011.

[11] Laurent Pellegrino, Fabrice Huet, Françoise Baude, and Amjad Alshabani, "A Distributed Publish/Subscribe System for RDF Data", Data Management in Cloud, Grid and P2P Systems, Prague, Czech Republic, pp. 39-50, August, 2013.

[12] David C. Faye, Oliver curé, and Guillaume Blin, "A survey of RDF storage approaches", revue Africaine de la recherche en Informatique et Mathématique Appliquées (ARIMA), Vol. 15 pp. 11-35, February, 2012.