Peer-to-peer Semantic Wikis

Hala Skaf-Molli, Charbel Rahhal, Pascal Molli

Peer-to-peer Semantic Wikis

Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli

INRIA Nancy-Grand Est
Nancy University, France
{skaf, charbel.rahal, molli}@loria.fr

Abstract. Wikis have demonstrated how it is possible to convert a community of strangers into a community of collaborators. Semantic wikis have opened an interesting way to mix web 2.0 advantages with the semantic web approach. P2P wikis have illustrated how wikis can be deployed on P2P wikis and take advantages of its intrinsic qualities: fault-tolerance, scalability and infrastructure cost sharing. In this paper, we present the first P2P semantic wiki that combines advantages of semantic wikis and P2P wikis. Building a P2P semantic wiki is challenging. It requires building an optimistic replication algorithm that is compatible with P2P constraints, ensures an acceptable level of consistency and generic enough to handle semantic wiki pages. The contribution of this paper is the definition of a clear model for building P2P semantic wikis. We define the data model, operations on this model, intentions of these operations, algorithms to ensure consistency and finally we implement the SWOOKI prototype based on these algorithms.

1 Introduction

Wikis are the most popular tools of Web 2.0, they provide an easy to share and contribute to global knowledge. The encyclopedia Wikipedia is a famous example of a wiki system. In spite of their fast success, wiki systems have some drawbacks. They suffer from search and navigation [1], it is not easy to find information in wikis [2]. They have also scalability, availability and performance problems [3, 4] and they do not support offline works and atomic changes [5]. To overcome these limitations, wiki systems have evolved in two different ways: semantic wikis and peer-to-peer wikis.

Semantic Wikis Semantic wikis are a new generation of collaborative editing tools.They allow users to add semantic annotations in the wiki pages. Users collaborate not only for writing the wiki pages but also for writing semantic annotations. Usually, this is done by annotating the links between wikis pages. Links in semantic wikis are typed. For instance, a link between the wiki pages "France" and "Paris" may be annotated by "capital". Semantic wikis provide a better structuring of wikis by providing a means to navigate and search based on annotations. These annotations express relationships between wikis pages, they are usually written in a formal syntax so they are processed automatically by machines and they are exploited by semantic queries. Semantic wikis try to preserve the benefits of classicals wikis such as, simplicity in creating and editing of wikis pages. Many semantic wikis are being developed such [1, Semantic MediaWiki], [6, IkeWiki] and [2, SweetWiki].

P2P wikis Wikis on a peer-to-peer network attempt to reconcile the benefits of mass collaboration and the intrinsic qualities of peer-to-peer network such as scalability, fault-tolerance, better performance and resistance to censorship [7]. There are currently many research proposals to build P2P wikis such [3, Wooki] and [8, git-wiki] [4, DistriWiki], [5, DtWiki], [9, Piki] and [10, Repliwiki]. The basic idea is to replicate wiki pages on the peers of a P2P network and the main problem is to ensure the consistency of copies.

In this paper, we propose to build the first peer-to-peer semantic wiki, called SWOOKI. SWOOKI combines advantages of P2P wikis and Semantic wikis. The main problem for building such a system is to maintain consistency of replicated semantic wiki pages. We propose an algorithm that ensures the CCI consistency (CCI stands for Causality, Convergence and Intention Preservation [11]) for the replicated semantic wiki pages. In the state of art, traditional wikis systems that ensure CCI consistency do not manage currently this data type.

Building a P2P semantic wiki based on the CCI model is challenging. The fundamental problem is to provide an optimistic replication algorithm that (1) is compatible with P2P constraints, (2) ensures the CCI model and (3) is generic enough to manage semantic wiki pages. In this paper, we define formally the semantic wiki page data type, we specify its operations and we define the intentions of these operations. We extend the WOOT [12] algorithm to take into account semantic annotations and finally we build SWOOKI the first P2P semantic wiki based on this algorithm.

The paper is organized as follow. Section 3 presents some related works. Section 4 details the general approach for building a P2P semantic wiki. It defines a data model and editing operations. Section 5 defines the causality and intentions of operations used to edit semantic data. Section 6 develops the integration algorithm. Section 7 gives an overview of the architecture. The last section concludes the paper and points to future works.

2 Use cases for P2P semantic wikis

We have identified three interesting use cases for P2P semantic wikis. We aim to develop a peer to peer semantic wiki that support these use cases.

Mass Collaboration . In this case, a P2P semantic wiki system is deployed as a Usenet network[13]. For instance, thousands of semantic wiki servers can be deployed within organizations or universities. Any user can connect to any semantic wiki server. This deployment allows :
- to handle a large number of users by dividing the load on the whole network. Semantic queries can be performed locally on each semantic wiki server.
- to tolerate many faults. A crash of one semantic wiki server does not stop the service.
- to share the cost of the infrastructure. Wikis are set up and maintained by different organizations. Therefore, it is not necessary to collect funds just to maintain the infrastructure. For instance, Wikipedia foundation has to collect 150000 $ every three months just to maintain the Wikipedia infrastructure.
- to resist to censorship. An organization controls only one semantic wiki server and not all data.

Off-line work and transactional changes Adding off-line capabilities to web applications is currently a major issue. For instance, the development of Google gears [10] and Firefox3 off-line capabilities demonstrate the need of the off-line work. Wikis are web applications and the need for off-line wiki editing is real. Current technologies for adding off-line capabilities to web applications focus on Ajax applications. However, the off-line mode of these web applications does not provide all features available in the on-line mode. This can be an obstacle for a wiki system. For instance, the off-line mode of the wiki allows navigation but it does not allow editing. A P2P semantic wiki tolerates naturally off-line work by means of an integrated merge algorithm. With such system, it is possible to travel with a complete wiki system on a laptop, make changes off-line and re-synchronize with the P2P network as soon as an Internet connection is available. The off-line mode enables also transactional changes. Users can work disconnected if they lack internet connection or if they decide to disconnect directly from the user interface. While disconnected, a user can change many semantic wiki pages in order to produce a consistent change. The P2P semantic wiki ensures that the execution of concurrent transactions on different peers lead to a consistent state of the semantic wiki pages.

Ad-hoc Collaborative Editing this scenario is derived from the previous one. Imagine several off-line wiki users have a meeting. Unfortunately, there is no Internet connection available in the meeting room. Therefore, they decide to set up an ad-hoc network within the meeting room. A P2P semantic wiki is able to propagate changes within the ad-hoc network and allows collaborative editing just for these off-line users. Of course, when the meeting is finished and users return to their organizations, their semantic wiki systems will re-synchronize with the whole P2P network.

The use cases presented above illustrate the importance of optimistic replication algorithms. In order to build a P2P Semantic wiki, there is a need for an algorithm that handles linear text and semantic annotations embedded in this text and ensures the CCI consistency.

3 Related work

The fundamental problem for building a P2P semantic wiki is to provide an optimistic replication algorithm that (1) supports collaborative editing, (2) manages a semantic wiki data type, (3) ensures the CCI model and (4) is compatible with P2P Constraints.

In the following, we show the state of the art of data replication in semantic web and collaborative systems

3.1 Replication in Semantic Web

Many researches have been done in P2P semantic web [14–18]. These works focus on sharing, querying and synchronizing RDF resources rather than collaborative editing of RDF resources. Sharing is different from collaboration. In sharing, some peers publish data while others can only read these data and concurrent updates are not managed. In P2P semantic wikis, some peers publish data, others can read and write these data and a synchronization algorithm integrates concurrent updates while maintaining consistency of these data. Existing synchronization algorithms of RDF resources such RDFSync [15] and RDFPeers [16] are used in the sharing context. Hence, they are not adapted for P2P semantic wikis.In summary, P2P Semantic Web researches focus on knowledge sharing and querying. No algorithm has been designed for collaborative editing of RDF graphs. Consequently, the existing algorithms do not take into account concurrent updates on RDF graphs.

3.2 Replication in Collaborative systems

Data replication in collaborative P2P systems mainly relies on optimistic replication. Existing approaches can not be applied to the P2P semantic wikis context. For instance, the *Bayou* system [19] is suitable for deploying a collaborative application on a decentralized network. Unfortunately, in order to ensure the convergence of copies, *Bayou* has to arrange eventually operations in the same order. To achieve this, it relies on a primary site that will enforce a global continuous order on a growing prefix of history. Using such a primary site is not compatible with P2P network constraints. Other systems such as Usenet [13] apply the Thomas write rule [20] to ensure eventual consistency. They ensure that, when the systems are idle *i.e* all operations have been sent and received by all sites, all copies are identical. Unfortunately, if there is two concurrent write operations on the same data unit, then the rule of "the last writer wins" is applied and in this case, this means that the modifications of a user are lost. Collaborative editing cannot be easily achieved if the system can loose some changes just to ensure eventual consistency.

Many algorithms have been developed by the Operational Transformation community [11] such as SOCT2, GOTO, COT etc. They are designed to verify the CCI model. But only few of them support P2P constraints such as MOT2 [21]. However MOT2 algorithm suffers from a high communication complexity [22] and no transformation functions for a semantic wiki page are available.

Some existing P2P wikis such as *giki* or *git-wiki* use distributed version control systems (DVCS) to manage data. Wiki pages are stored as text files. DVCS manage them as code files. They ensure causal consistency. This implies that concurrent write operations can be seen in a different order on different machines. In this case, if two sites observe 2 write operations in different order then copies on both sites can diverge. DVCS systems are aware of this problem and delegate the problem to external merge algorithms for managing concurrent operations. However, as existing merge algorithms are not intrinsically deterministic, commutative and associative so convergence cannot be ensured in all cases. Wooki [3] is the only available P2P wiki that ensures the CCI consistency. Wooki relies on the WOOT [12] algorithm to ensure the CCI consistency for wiki pages. However, WOOT cannot be applied directly to a P2P semantic wiki. WOOT is designed to synchronize linear structures, it can not synchronize a mix of text and RDF graphs. We propose to extend the WOOT algorithm to handle collaborative writing on replicated RDF data model.

4 P2P Semantic Wiki Approach

A P2P semantic wiki is a P2P network of autonomous semantic wiki servers (called also peers or nodes) that can dynamically join and leave the network. Every peer hosts a copy of all semantic wiki pages and an RDF store for the semantic data. Every peer can autonomously offer all the services of a semantic wiki server. When a peer updates its local copy of data, it generates a corresponding operation. This operation is processed in four steps: it is executed immediately against the local replica of the peer, broadcasted through the P2P network to all other peers, received by the other peer and integrated to their local replica. If needed, the integration process merges this modification with concurrent ones, generated either locally or received from a remote server.

The system is correct if it ensures the CCI (Causality, Convergence and Intention Preservation) consistency model [11].

– *Causality preservation:* operations ordered by a precedence relation will be executed in same order on every peer.

– *Convergence:* When the system is idle, all copies are identical.

– *Intention and Intention preservation:* The intention of an operation is the effects observed on the state when the operation was generated. The effects of executing an operation at all sites are the same as the intention of the operation. The effect of executing an operation does not change the effects of independent (not causally dependent) operations.

4.1 Data Model

The data model is an extension of Wooki [3] data model to take in consideration semantic data. Every semantic wiki peer is assigned a global unique identifier named $NodeID$. These identifiers are totally ordered. As in any wiki system, the basic element is a semantic wiki page and every semantic wiki page is assigned a unique identifier $PageID$, which is the name of the page. The name is set at the creation of the page. If several servers create concurrently pages with the same name, their content will be directly merged by the synchronization algorithm. Notice that a URI can be used to unambiguously identify the concept described in the page. The URI must be global and location independent in order to ensure load balancing. For simplicity, in this paper, we use a string as page identifier.

Pages storage model

Definition 1. *A semantic wiki page Page is an ordered sequence of lines* $L_B L_1, L_2, \ldots L_n L_E$ *where* L_B *and* L_E *are special lines.* L_B *indicates the beginning of the page and* L_E *indicates the ending of the page.*

Definition 2. *A semantic wiki line L is a four-tuple* $<$ *LineID, content, degree, visibility* $>$ *where*

– $LineID$ *is the line identifier, it is a pair of (*$NodeID$*, logicalclock) where* $NodeID$ *is the identifier of the semantic wiki server and logicalclock is a logical clock of that server. Every semantic wiki server maintains a logical clock, this clock is incremented when an operation is generated. Lines identifiers are totally ordered so if* $LineID_1$ *and* $LineID_2$ *are two different lines with the values* ($NodeID_1$, $LineID_1$) *and* ($NodeID_2$, $LineID_2$) *then* $LineID_1 < LineID_2$ *if and only if (1)* $NodeID_1 < NodeID_2$ *or (2)* $NodeID_1 = NodeID_2$ *and* $LineID_1 < LineID_2$.

– *content is a string representing text and the semantic data embedded in the line.*

– *degree is an integer used by the synchronization algorithm, the degree of a line is fixed when the line is generated, it represents a kind of loose hierarchical relation between lines. Lines with a lower degree are more likely generated earlier than lines with a higher degree. By definition the degree of* L_E *and* L_B *is zero.*

– *visibility is a boolean representing if the line is visible or not. Lines are never really deleted they are just marked as invisible.*

For instance, suppose there are two lines in a semantic wiki page about "France" , "France" is the identifier of the page.

France is located **in** [locatedIn :: Europe]
The capital of France is [hasCapital:: Paris]

Now, suppose these two lines are generated on the server with $NodeID = 1$ in the above order and there are no invisible lines, so the semantic wiki page will be internally stored as:

L_B
((1,1), France is located **in** [locatedIn :: Europe], 1, true)
((1,2), The capital of France is [hasCapital:: Paris], 2, true)
L_E

Text and semantic data are stored in separate persistent storages. Text can be stored in files and semantic data can be stored in RDF repositories, as described in the next section.

Semantic data storage model RDF is the standard data model for encoding semantic data. In P2P semantic wikis, every peer has a local RDF repository that contains a set of RDF statements extracted from its wikis pages. A statement is defined as a triple (Subject, Predicate, Object) where the subject is the name of the page and the predicates (or properties) and the objects are related to that concept.

For instance, the local RDF repository of the above server contains : $R = \{$("France", "locatedIn", "Europe"), ("France", "hasCapital", "Paris") $\}$. As for the page identifier, a global URI can be assigned to predicates and objects of a concept, for simplicity, we use a string.

We define two operations on the RDF repositories:

– insertRDF(R,t): adds a statement t to the local RDF repository R.

– deleteRDF(R,t): deletes a statement t from the local RDF repository R.

These operations are not manipulated directly by the end user, they are called implicitly by the editing operations as shown later.

4.2 Editing operations

A user of a P2P semantic wiki does not edit directly the data model. Instead, she uses traditional wiki editing operations, when she opens a semantic wiki page, she sees a view of the model. In this view, only visible lines are displayed. As in a traditional semantic wiki, she makes modifications i.e. adds new lines or deletes existing ones and she saves the page(s).

To detect user operations, a diff algorithm is used to compute the difference between the initial requested page and the saved one. Then these operations are transformed into model editing operations. A *delete* of the line number n is transformed into a *delete* of the n^{th} visible line and an *insert* at the position n is transformed into *insert* between the $(n-1)^{th}$ and the n^{th} visible lines. These operations are integrated locally and then broadcasted to the other servers to be integrated.

There are two editing operations for editing the wiki text: *insert* and *delete*. An update is considered as a delete of old value followed by an insert of a new value. There are no special operations for editing semantic data. Since semantic data are embedded in the text, the RDF repositories are updated as a side effect of text replication and synchronization.

1. *Insert(PageID, line, l_P , l_N)* where $PageID$ is the identifier of the page of the inserted line. *line* is the line to be inserted. It is a tuple containing < LineID, content, degree, visibility >. l_P and l_N are respectively the identifiers of the lines that precedes and follows the inserted line.

During the insert operation, the semantic data embedded in the line are extracted, RDF statements are built with the page name as a subject and then they are added to the local RDF repository thanks to the function $insertRDF(R, t)$.

2. The *delete(PageID, LineID)* operation sets the visibility of the line identified by $LineID$ of the page $PageID$ to false. The line is not deleted physically, it is just marked as deleted. The identifiers of deleted lines must be kept as a tombstones. During the delete operation, the set of the

RDF statements contained in the deleted line are deleted from the local RDF repository thanks to the $deleteRDF(R, t)$.

5 Correction Model

This section defines causal relationships and intentions of the editing operations for our P2P semantic wiki data model.

5.1 Causality preservation

The causality property ensures that operations ordered by a precedence relation will be executed in the same order on every server. In WOOT, the precedence relation relies on the *semantic causal dependency*. This dependency is explicitly declared as preconditions of the operations. Therefore, operations are executed on a state where they are legal *i.e.* preconditions are verified. In the following, we define causality for editing operations that manipulate text and RDF data model.

Definition 3. ***insert Preconditions*** *Let Page be the page identified by PageID, let the operation op=*Insert($PageID$, *newline*, p , n), *newline =< LineID, c, d, v> generated at a server NodeID, R is its local RDF repository. The line newline can be inserted in the page Page if its previous and next lines are already present in the data model of the page Page.*

$$\exists i \; \exists j \; LineID(Page[i]) = p \wedge \; LineID(Page[j]) = n$$

Definition 4. ***Preconditions of delete operation*** *Let Page be the page identified by PageID, let op =* Delete($PageID$, *dl*) *generated at a server NodeID with local RDF repository R, the line identified by dl can be deleted (marked as invisible), if its dl exists in the page.*

$$\exists i \; LineID(Page[i]) = dl$$

When a server receives an operation, the operation is integrated immediately if its pre-conditions are evaluated to true else the operation is added to a waiting queue, it is integrated later when its pre-conditions become true.

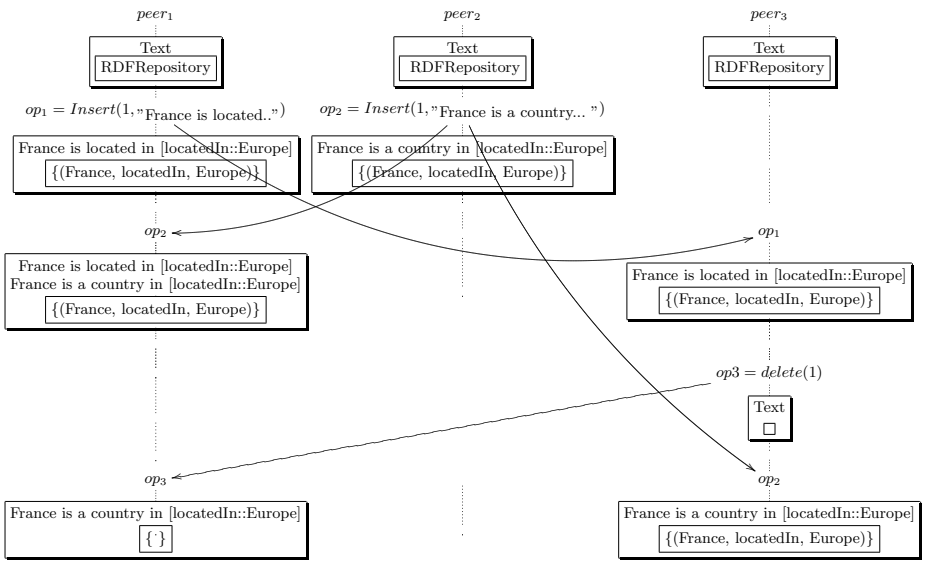

Fig. 1. Semantic inconsistency after integrating concurrent modifications

5.2 Intentions and Intentions preservation

The intention of an operation is the visible effect observed when a change is generated at one peer, **the intention preservation means that the intention of the operation will be observable on all peers, in spite of any sequence of concurrent operations.**

We can have a naive definition of intention for *insert* and *delete*:

– The intention of an insert operation *op= Insert(PageID, newline, p , n)* when generated at site $NodeID$, where *newline* =< nid, c, d, v> is defined as: (1) The content is inserted between the previous and the next lines and (2) the semantic data in the line content are added to the RDF repository of the server.

– The intention of a delete operation *op= delete(pid, l)* when generated at site S is defined as : (1) the line content of the operation is set to invisible and (2) the semantic data in the line content are deleted from the RDF repository of the server.

Unfortunately, it is not possible to preserve the previous intention definitions. We illustrate a scenario of violation of these intentions in figure 1. Assume that three P2P semantic wiki servers, $peer_1$, $peer_2$ and $peer_3$ share a semantic wiki page about "France". Every server has its copy of shared data and has its own persistence storage repository. At the beginning, the local text and the RDF repositories are empty. At $peer_1$, $user_1$ inserts the line "France is located [located In::Europe]" at the position 1 in her copy of the "France" page. Concurrently, at $peer_2$ $user_2$ inserts a new line "France is a country in [located In::Europe]" in her local copy of "France" page at the same position and finally at $peer_3$ $user_3$ deletes the line added by $user_1$.

When op_2 is integrated at $peer_1$, the semantic annotation is present two times in the text and just one time in the RDF repository. In fact, the RDF repository cannot store twice the same triple. When op_3 is finally integrated on $peer_1$, it deletes the corresponding line and the semantic entry in the RDF repository. In this state, the text and the RDF repository are inconsistent.

Concurrently, $peer_3$ has integrated the sequence [op_1;op_3;op_2]. This sequence leads to a state different than the state on $peer_1$. Copies are not identical, convergence is violated.

Intentions presented above cannot be preserved because the effect of executing op_3 changes the effect of op_2 which is independent, of op_3 i.e. op_3 deletes the statement inserted by op_2, but op_3 has not seen op_2 at generation time.

5.3 Model for Intention preservation

It is not possible to preserve intentions if the RDF store is defined as a set of statements. **However, if we transform the RDF store into multi-set of statements, it becomes possible to define intentions that can be preserved.**

Definition 5. ***RDF repository*** *is the storage container for RDF statements, each container is a* multi-set of RDF statements. *Each RDF repository is defined as a pair* (T, m) *where* T *is a set of RDF statements and* m *is the multiplicity function* $m : T \rightarrow \mathcal{N}$ *where* $\mathcal{N} = 1, 2.....$

For instance, the multi-set R = { ("France", "LocatedIn", "Europe"),("France", "LocatedIn", "Europe"),("France", "hasCapital", "Paris") } can presented by R = { ("France", "LocatedIn", "Europe")^2, ("France", "hasCapital", "Paris")^1 } where 2 is the number of occurrence of the first statement and 1 is this of the second one.

Definition 6. ***Intention of insert operation*** *Let* S *be a P2P semantic wiki server,* R *is its local RDF repository and Page is a semantic wiki page. The intention of an insert operation op=* Insert($PageID$, *newline*, p , n) *when generated at site* S*, where newline =< nid, c, d, v> and* T *is the set (or multi-set) of RDF statements in the inserted line, is defined as: (1) The content is inserted between the previous and the next lines and (2) the semantic data in the line content are*

added to R.

$$\exists i \wedge \exists\, i_P < i \; LineID(Page[i_P]) = p \tag{1}$$
$$\wedge \exists\, i \leq i_N \; LineID(Page[i_N]) = n \tag{2}$$
$$\wedge\, Page'[i] = newline \tag{3}$$
$$\wedge\, \forall j < i \; Page'[j] = Page[j] \tag{4}$$
$$\wedge\, \forall j \geq i \; Page'[j] = Page[j-1] \tag{5}$$
$$\wedge\, R' \leftarrow R \uplus T \tag{6}$$

Where $Page'$ and R' are the new values of the page and the RDF repository respectively after the application of the insert operation at the server S and $\uplus$ is the union operator of multi-sets. If a statement in T already exists in R so its multiplicity is incremented else it is added to R with multiplicity one.

Definition 7. ***Intention of delete operation*** *Let S be a P2P semantic wiki server, R is the local RDF repository and Page is a semantic wiki page. The intention of a delete operation op=* delete($PageID$, ld) *where T is the set (or multi-set) of RDF statements in the deleted line, is defined as: (1) the line ld is set to invisible and (2) the number of occurrence of the semantic data embedded in ld is decreased by one, if this occurrence is equal to zero which means these semantic data are no more referenced in the page then they are physically deleted from the R.*

$$\exists i \quad \wedge\, PageID(Page'[i]) = ld \tag{7}$$
$$\wedge\, visibility(Page'[i]) \leftarrow false \tag{8}$$
$$\wedge\, R' \leftarrow R - T \tag{9}$$

Where $Page'$ and R' are the new values of the page and the RDF repository respectively after the application of the delete operation at the server S and $-$ is the difference of multi-sets. If statement(s) in T exists already in R so its multiplicity is decremented and deleted from the repository if it is equal to zero.

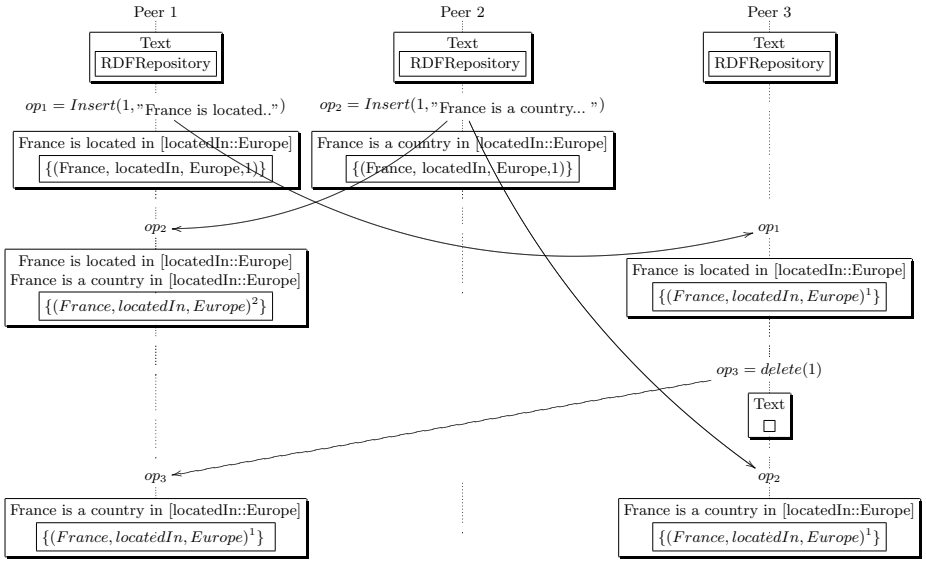

Fig. 2. Convergence after integrating concurrent modifications

Let us consider again the scenario of the figure 1. When op_2 is integrated on $peer_1$, the multiplicity of the statement ("$France$","$locatedIn$","$Europe$") is incremented to 2. When op_3 is

integrated on $peer_1$, the multiplicity of the corresponding statement is decreased and the consistency between text and RDF repository is ensured. We can observe that $Peer_1$ and $Peer_3$ now converge and that intentions are preserved.

6 Algorithms

As any wiki server, a P2P semantic server defines a *Save* operation which describes what happens when a semantic wiki page is saved. In addition, it defines *Receive* and *Integrate* operations. The first describes what happens upon receiving a remote operation and the second integrates the operation locally.

6.1 Save operation

During saving a wiki page, a $Diff$ algorithm computes the difference between the saved and the previous version of the page and generates a patch. A *patch* is the set of delete and insert operations on the page (Op = *Insert(PageID, line,* l_P *,* l_N*)* or Op = *Delete(PageID, LineID)*). These operations are integrated locally and then broadcasted to other sites in order to be executed as shown below.

```
1  Upon Save(page, oldPage) :−
2     let P ← Diff(page, oldPage)
3   for each op ∈ P do
4     Receive(op)
5   endfor
6  Broadcast(P)
```

At this level of description, we just make the hypothesis that $Broadcast(P)$ will eventually deliver the patch P to all sites. More details are given in section 7.

6.2 Delivery Operation

When an operation is received its preconditions are checked. If they are not satisfied, the operation is added to the waiting log of the server, else according to the type of the operations some steps are executed.

```
7  Upon Receive(op) :−
8   if isExecutable(op) then
9     if type(op) = insert then
10      IntegrateIns(op)
11    if type(op) = delete then
12      IntegrateDel(op)
13  else
14        waitingLog ← waitingLog ⋃ {op}
15  endif
```

The waiting log is visited after the integration and the operations that satisfy their preconditions are removed from the log and integrated.

```
16 isExecutable(op) :−
17 if type(op) = del then
18    return containsL(PageID,LineID) and isVisible(LineID)
19 else
20    return ContainsL(PageID,l_P)
21         and ContainsL(PageID, l_N)
22 endif
```

The function $ContainsL(PageID, id)$ tests the existence of the line in the page, it returns true if this is the case. The function $isVisible(LineID)$ tests the visibility of the line.

6.3 Integrate operation

The integration of an operation is processed in two steps: (1) text integration and (2) RDF statements integration.

```
IntegrateDel(LineID) :−
    IntegrateDelT(LineID)
    IntegrateDelRDF(LineID)
```

To integrate a *delete* operation, the visibility flag of the line is set to false whatever is its content.

```
IntegrateDelT(LineID) :−
    Page[LineID]. visibility    ← false
```

To integrate RDF statements, a counter is used to implement a multi-set RDF repository. A counter is attached to every RDF triple, the value of the counter corresponds to the number of occurrence of the triple in the repository.

```
IntegrateDelRDF(LineID) :−
let   S ← ExtractRDF(LineID)
if  S ≠ ∅ then
 for each  triple  ∈ S do
      triple .counter−−
     if    triple .counter = 0 then
       deleteRDF(R,triple)
     endif
endif
```

During the delete operation, the counter of the deleted statements is decreased, if the counter is zero the statements are physically deleted from the repository.

To integrate an insert operation $op = insert(\ PageID, line, l_P, l_N)$, the *line* has to be placed among all the lines between l_P and l_N, some of these lines can be previously deleted or inserted concurrently and the inserted semantic data are integrated.

```
IntegrateIns(PageID, line,  l_P , l_N) :−
    IntegratedInsT(PageID, line,  l_P , l_N)
    IntegrateInsRDF(line)
```

To integrate a line in a wiki page, we use the integration algorithm defined in [3].

```
IntegrateInsT(PageID, line,  l_P , l_N) :−
let  S'   ← subseq(Page[PageID]), l_P, l_N)
if  S = ∅ then
   insert(PageID, line, l_N)
else
    let  i   ← 0
    let   d_min ← min(degree(S'))
    let  F ← filter(S', degree = d_min)
    while (i < |F| − 1) and (F[i] <_id line) do
        i   ← i +1
     IntegrateInsT(PageID, line,  F[i−1],  F[i ])
endif
```

This algorithm selects the sub-sequence S' of lines between the previous and the next lines, in case of an empty result, the line is inserted before the next line. Else, the sub-sequence S' is filtered by keeping only lines with the minimum degree of S'. The remaining lines are sorted according to the line identifiers order relation $<_{id}$ [12], therefore, *line* will be integrated in its place according $<_{id}$ among remaining lines, the procedure is called recursively to place *line* among lines with higher degree in S'.

To integrate the semantic data, the RDF statements of the inserted line are extracted and added to the local RDF repository. If the statements exist already in the repository, their counter is incremented, otherwise, they are inserted into the RDF repository with a counter value equals to one as shown below.

```
52 IntegrateInsRDF(line) :−
53 let  S ← ExtractRDF(line)
54 if S ≠ ∅ then
55   for each triple ∈ S do
56     if  Contains(triple) then
57         triple .counter++
58     else
59        insertRDF(R,triple)
60     endif
61  endif
```

6.4 Synthesis

Causality as defined in section 5.1 is ensured by the *Receive* algorithm.

Convergence for text is already ensured by the WOOT algorithm [12]. Convergence for semantic data is trivially ensured by the multi-set extension of the RDF repository.

The intention preservation for a text is demonstrated in [12]. Here, we are concerned with the intention of semantic data as defined in 5.2. The intention of an *insert* operation is trivially preserved by the algorithm *IntegrateInsRDF*. Since a possible way to implement a *multi-set* is to associate a counter to every element. In the same way, the algorithm *IntegrateDelRDF* preserves the intention of the *delete* operation. The basic idea behind all these algorithms is to reach convergence and preserve intentions whatever is the order of reception of operations. This implies that these algorithms "force" commutativity of operations. If operations are commuting then all concurrent executions are equivalent to a serial one. In our system, users can start a transaction just by switching to the offline mode and end a transaction by switching to online mode. We chose this way to interact with users in order to keep the system simple. If a user produces a consistent change, as all operations of any transaction are commuting and ensure the same effects, then all concurrent execution of transactions generate a correct state.

7 Implementation and discussion

We have implemented the first peer-to-peer semantic wiki called SWOOKI based on algorithms presented in section 6. The SWOOKI prototype has been implemented in Java as servlets in a Tomcat Server and demonstrated in [23]. This prototype is available with a GPL license on sourceforge at `http://sourceforge.net/projects/wooki` and it is also available online at: `http://wooki.loria.fr/wooki1`.

7.1 SWOOKI Architecture

A SWOOKI server is composed of the following components (see figure 3):

User Interface. The SWOOKI UI component (shown in figure 4) is basically a regular wiki editor. It allows users to edit a view of a page by getting the page from the SWOOKI manager. Users can disconnect their peer to work in an off-line mode (feature1) and they can add new neighbors in their list to work with (feature2). In addition, the UI allows users to see the history of a page, to search for pages having some annotation (feature3), to execute semantic queries (feature4), and to export the semantic annotations of the wiki pages in an RDF format (feature5).

SWOOKI Manager. The SWOOKI manager is responsible for the generation and the integration of the editing patches. It implements the SWOOKI algorithm. Its main method is

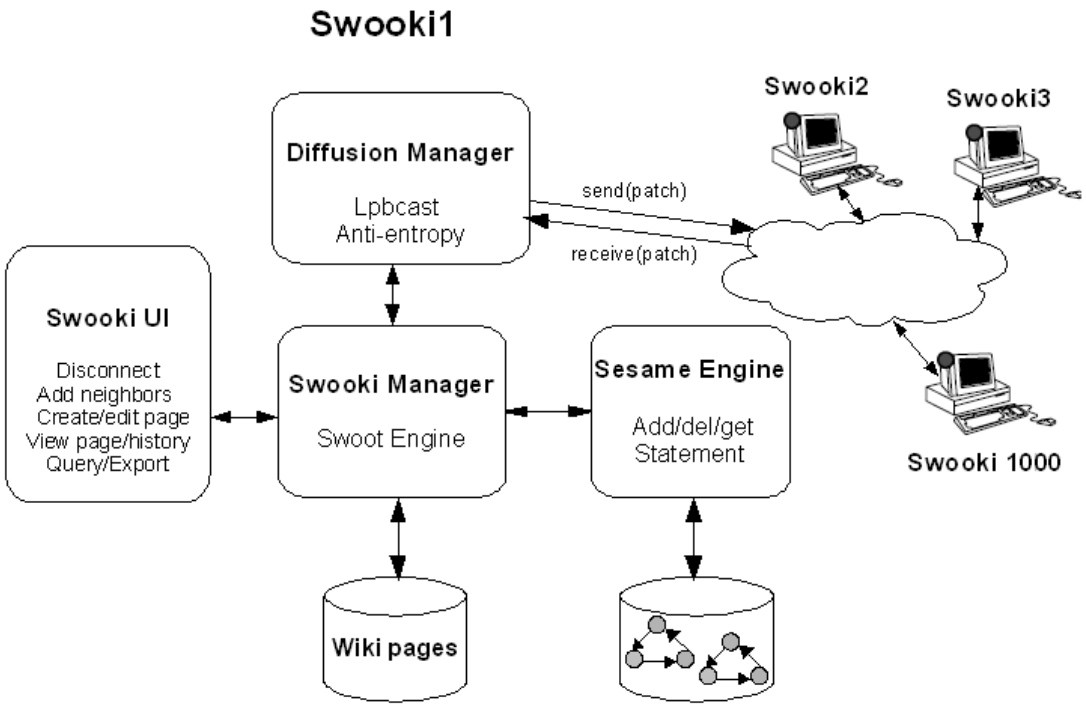

Fig. 3. SWOOKI Architecture

`Integrate(Patch)` that calls the `Receive()` algorithm for all operations contained in the patch. Requesting and modifying a page or resolving a semantic query in the RDF repository pass through this manager.

Sesame Engine. We use Sesame 2.0 [24] as RDF repository. Sesame is controlled by the SWOOKI manager for storing and retrieving RDF triples . We used a facility of the Sesame interface to represent RDF triples as multi-set. This component allows also generating dynamic content for wiki pages using queries embedded in the wiki pages. It provides also a feature to export RDF graphs.

Diffusion Manager. In order to ensure the CCI model, we made the hypothesis that all operations eventually reach all sites of the unstructured P2P network. The diffusion manager is in charge to maintain the membership of the unstructured network and to implement a reliable broadcast. Membership and reliable broadcast of operations are ensured by an implementation of the Lpbcast algorithm [25]. This algorithm ensures that all connected sites receive messages and that there is no partition in the P2P network. Of course, disconnected sites cannot be reached. So we added an anti-entropy mechanism based on [26]. The anti-entropy algorithm selects randomly a neighbor in the local table of neighbors and sends a digest of its own received messages. The receiver returns missing messages to the caller. Using the anti-entropy implies that each server keeps received messages in a log, as this log can grow infinitely, the log is purged as detailed in [3].

7.2 Discussion

Every SWOOKI server provides all the services of a semantic wiki server. We analyze our system with respect to the following criteria, more detailed analyzing results about the implemented algorithms can be found in [25] and [22].

Availability, fault tolerance and load balancing Data are available on every peer so they are accessible even when some peers are unavailable. The global naming of the wiki pages

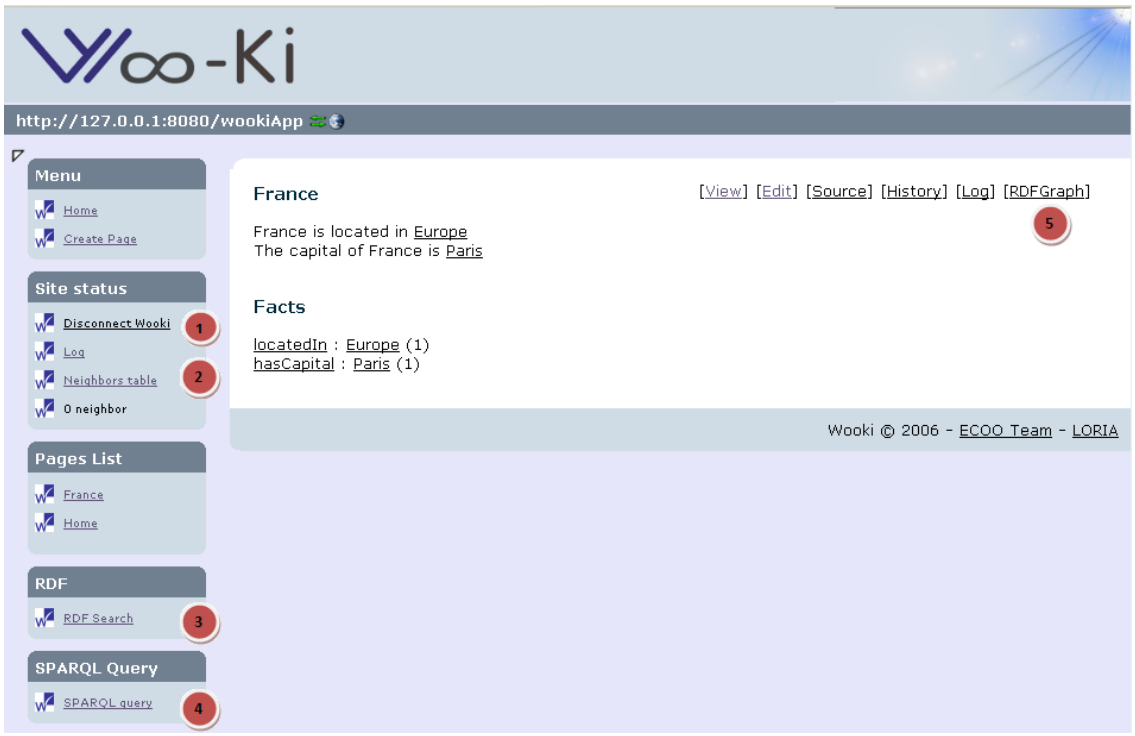

Fig. 4. User Interafce of SWOOKI

(concepts) and their associated properties and objects and the respect of the CCI model ensure to have the same data at any node. So if a server is unavailable or slow, it is possible to access to another server.

Performance We analyze the performance with respect to messages necessary to execute query, propagate modification and synchronize data.

– *Query execution:* Every semantic wiki server can execute every semantic query locally without generating network traffic for resolving it.

– *Messages delivery:* As our algorithm generates no traffic for ensuring CCI consistency, the traffic cost for our system is the traffic cost of Lpbcast [25] and the traffic cost of the anti-entropy classical algorithm. As logs of messages can be purged safely, the traffic cost, even for anti-entropy, is bounded. The volume of changes generated by the Wikipedia in French is less than 100Mb per month between September 06 and May 08 [1]. In 2008, the total size of Wikipedia in French was less than 2,5Gb.

– *Data synchronization:* The complexity of the integration of n operations is $O(n * l^2)$ [22], where l is the number of lines that have been inserted in the wiki page. In fact, as deleted lines are just marked as deleted and there is no garbage collecting algorithm compatible with P2P constraints, the size of the wiki page is growing infinitely. However, traditional wikis such as Wikipedia keeps all changes in log history and never delete them. Consequently, in the context of a wiki, our solution seems acceptable.

Scalability SWOOKI scales with respect to the number of peers. The number of peers is not a parameter of the complexity in time and space of our algorithm.

However, It does not support solution for scalability with the size of data, the storage capacity is limited by the storage capacity of each node. For achieving this scalability, a solution based on partial replication is better. But in this case, offline editing and transactional changes are much more difficult to obtain.

[1] `http://stats.wikimedia.org/EN/TablesDatabaseEdits.htm`

Offline-work and transactional changes Users can work disconnected if they have no Internet connection or if they decide to disconnect directly from the user interface. While disconnected, a user can change many semantic wiki pages in order to produce a consistent change. By this way she generates a transaction. All changes performed in disconnected mode are kept in the diffusion manager component. As our optimistic replication algorithm forces all operations to commute (according to the CCI consistency) then, the concurrent execution of several transactions is always equivalent to a serial one. Thus, a consistent state is produced in all cases.

Cost sharing The deployment of SWOOKI network is very similar to the deployment of the Usenet P2P network. A trusted peer of any organization can join the network, take a snapshot of replicated data and start answering wiki requests. The proposed architecture can be easily deployed on the Internet across different organizations. In the contrast to the Wikipedia infrastructure that requires a central site with costly hardware and high bandwidth, the cost of the underlying infrastructure of our system can be shared by many different organizations.

8 Conclusion, Open Issues and Perspectives

Peer-to-peer semantic wikis combines both advantages of semantic wikis and P2P wikis. The fundamental problem is to develop an optimistic replication algorithm that ensures an adequate level of consistency, supports P2P constraints and manages semantic wiki page data type. In this paper, we proposed such an algorithm. By combining P2P wikis and semantic wikis, we are able to deliver a new work mode for people working on ontologies: transactional changes. This work mode is useful to help people to produce consistent changes in semantic wikis. Often, managing a semantic wiki requires to change a set of semantic wiki pages. These changes can take a long time and if intermediate state results are visible, it can be confusing for other users and it can corrupt the result of semantic requests. We believe that this working mode is crucial if we want to use semantic wikis for collaborative ontologies building. However, this approach has many open issues and perspectives:

– Security issues are an important aspect. Replication makes security management more difficult. Often in wikis, security is represented as page attributes. If wiki pages are replicated, it means that security policies are replicated. In this case, it is possible to produce concurrent changes on security policy itself. If we re-centralize security management, we loose the benefits of replication. This problem can be solved by applying the approach proposed in this paper. To manage security policy: define the security policy data type, its operations, the intentions of these operations and update the replication algorithm with these new operations.

– An alternative way for managing security issues is to deploy a P2P semantic wiki on the web of trust. Instead on relying on Lpbcast to build an unstructured P2P network, users can organize themselves in the topology of a network based on trusted relationships. Consequently, the resulting system is a P2P semantic wiki based on a social network that promotes privacy.

– We explored also the combination of an unconstrained semantic wiki with a P2P wiki system based on total replication. We motivated this choice by pointing out to the need of transactional changes. However, even if it is more difficult, it is possible to achieve the same objective with a P2P wiki based on partial replication and consequently take advantage of partial replication benefits such as reduced traffic, infinite storage and cheap join procedure.

References

1. Vôlkel, M., Krtôzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic wikipedia. In: WWW '06: Proceedings of the 15th international conference on World Wide Web, New York, NY, USA, ACM Press (2006) 585–594
2. Buffa, M., Gandon, F.L., Ereteo, G., Sander, P., Faron, C.: Sweetwiki: A semantic wiki. Journal of Web Semantic **6**(1) (2008) 84–97
3. Weiss, S., Urso, P., Molli, P.: Wooki: a p2p wiki-based collaborative writing tool. In: Web Information Systems Engineering, Nancy, France, Springer (2007)

4. Morris, J.: DistriWiki: a distributed peer-to-peer wiki network. Proceedings of the 2007 international symposium on Wikis (2007) 69–74
5. Du, B., Brewer, E.A.: Dtwiki: a disconnection and intermittency tolerant wiki. In: WWW '08: Proceeding of the 17th international conference on World Wide Web, New York, NY, USA, ACM (2008) 945–952
6. Schaffert, S.: Ikewiki: A semantic wiki for collaborative knowledge management. In: WETICE, IEEE Computer Society (2006) 388–396
7. Androutsellis-Theotokis, S., Spinellis, D.: A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys **36**(4) (2004) 335–371
8. : git based wiki: http://atonie.org/2008/02/git-wiki (2008)
9. Patrick Mukherjee, C.L., Schurr, A.: Piki - a peer-to-peer based wiki engine. In: P2P08: Eighth International Conference on Peer-to-Peer Computing, IEEE (2008) 185–186
10. Kang, B., Wilensky, R., Kubiatowicz, J.: The hash history approach for reconciling mutual inconsistency. Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on (2003) 670–677
11. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems. ACM Transactions on Computer-Human Interaction **5**(1) (1998) 63–108
12. Oster, G., Urso, P., Molli, P., Imine, A.: Data Consistency for P2P Collaborative Editing. In: Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006, Banff, Alberta, Canada, ACM Press (2006)
13. Spencer, H., Lawrence, D.: Managing Usenet. O'Reilly Sebastopol (1988)
14. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: Edutella: a p2p networking infrastructure based on rdf. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, ACM (2002) 604–615
15. Morbidoni, C., Tummarello, G., Erling, O., Bachmann-Gmür, R.: Rdfsync: efficient remote synchronization of rdf models. In: Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea, Berlin, Heidelberg, Springer Verlag (2007)
16. Cai, M., Frank, M.: Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, ACM (2004) 650–657
17. Chirita, P.A., Idreos, S., Koubarakis, M., Nejdl, W.: Publish/subscribe for rdf-based p2p networks. In: The Semantic Web: Research and Applications, First European Semantic Web Symposium, ESWS 2004, LNCS 3053 (2004) 182–197
18. Staab, S., Stuckenschmidt, H., eds.: Semantic Web And Peer-to-peer. Springer (2005)
19. Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M., Demers, A.J.: Flexible update propagation for weakly consistent replication. In: Proceedings of the sixteenth ACM symposium on Operating systems principles - SOSP'97, ACM Press (1997) 288–301
20. Johnson, P., Thomas, R.: RFC677: The maintenance of duplicate databases (1976)
21. Cart, M., Ferrie, J.: Asynchronous reconciliation based on operational transformation for P2P collaborative environments. In: CollaborateCom: International Conference on Collaborative Computing: Networking, Applications and Worksharing, White Plains, New York, USA, IEEE Computer Society (2008) 127–138
22. Ignat, C.L., Oster, G., Molli, P., Cart, M., Ferrié, J., Kermarrec, A.M., Sutra, P., Shapiro, M., Benmouffok, L., Busca, J.M., Guerraoui, R.: A Comparison of Optimistic Approaches to Collaborative Editing of Wiki Pages. In: Proceedings of CollaborateCom 2007, White Plains, New York, USA, IEEE Computer Society (2007) 10
23. Rahhal, C., Skaf-Molli, H., Molli, P.: Swooki: A peer-to-peer semantic wiki. In: The 3rd Workshop: 'The Wiki Way of Semantics'-SemWiki2008, co-located with the 5th Annual European Semantic Web Conference (ESWC), Tenerife, Spain. (2008)
24. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A generic architecture for storing and querying rdf and rdf schema. In: ISWC 2002: First International Semantic Web Conference. (2002)
25. Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kouznetsov, P., Kermarrec, A.M.: Lightweight Probabilistic Broadcast. ACM Transactions on Computer Systems **21**(4) (2003) 341–374
26. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic Algorithms for Replicated Database Maintenance. In: Proceedings of the ACM Symposium on Principles of Distributed Computing - PODC'87, Vancouver, British Columbia, Canada, ACM Press (1987) 1–12