Synchronizing semantic stores with Commutative Replicated Data Types

Luis Daniel Ibáñez
University of Nantes
luis.ibanez@univ-nantes.fr

Hala Skaf-Molli
University of Nantes
hala.skaf@univ-nantes.fr

Pascal Molli
University of Nantes
pascal.molli@univ-nantes.fr

Olivier Corby
INRIA Sophia Antipolis-Méditerranée
olivier.corby@inria.fr

ABSTRACT

Social semantic web technologies led to huge amounts of data and information being available. The production of knowledge from this information is challenging, and major efforts, like DBpedia, has been done to make it reality. Linked data provides interconnection between this information, extending the scope of the knowledge production.

The knowledge construction between decentralized sources in the web follows a co-evolution scheme, where knowledge's generated collaboratively and continuously. Sources are also autonomous, meaning that they can use and publish only the information they want.

The updating of sources with this criteria is intimately related with the problem of synchronization, and the consistency between all the replicas managed.

Recently, a new family of algorithms called Commutative Replicated Data Types have emerged for ensuring eventual consistency in highly dynamic environments. In this paper, we define SU-Set, a CRDT for RDF-Graph that supports SPARQL Update 1.1 operations.

1. INTRODUCTION

The uprising of social semantic web technologies [6] led to the publication of enormous amounts of data and information. The continuous production of knowledge from this information is challenging, for example, DBpedia [4] makes a huge effort in converting the information from wikipedia pages to semantic knowledge.

Linked Data [3] provides a set of best practices to interconnect the information published on the web, creating the so-called "Web of Data". With it, the knowledge production started to take in consideration the links between the information sources, augmenting considerably its scope.

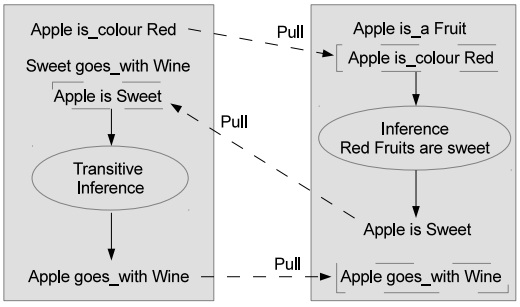

Figure 1: Co-evolution of knowledge in decentralized stores

This knowledge can be used to modify content, which then could be used to generate new knowledge, creating a co-evolution scheme, where knowledge is generated collaboratively and continuously. For example, imagine two semantic stores that are subscribed to each other updates, one has the information that apple is a fruit, and the other that apple is red plus an inference mechanism, whether it be by an automatic procedure or by human intervention, that allow it to derive that red fruits are sweet. After the second store "pulls" the knowledge that an apple is a fruit from the first one, it can use its inference mechanism to produce the knowledge that apple is sweet. Subsequently, the first store can pull this new fact and use it to infer another. Figure 1 illustrates this.

These stores, besides being decentralized, are autonomous: they do not pull everything from the other, nor gives each other writing privileges. Moreover, a store can have extreme behavior: it could pull everything from other stores, or it could stop pulling for an undetermined amount of time and then start again. Autonomy of participants pose interesting challenges on the implementation: if a group of stores agrees to pull everything between them, they should converge to

the same knowledge; if one of them stops pulling, whether it be completely or partially, there should exists the possibility of converging again if it executes all the missing updates asynchronously.

The updating of decentralized stores leads immediately to the question of synchronization, as raised by Berners-Lee [2]. A first solution is maintaining a copy of each involved dataset at each site, but this quickly falls out because of the freshness (what happens if after the copying the original site updates?), and the consistency (what happens if after the copying both sites update?). If we also take in consideration the size of the stores and the potentially high and generally unknown number of participants, only the optimistic replication family of algorithms can fit in.

In optimistic replication [14], each store sends the local operations asynchronously, and the others will eventually receive them and execute them, possibly in different orders. The system is consistent if it preserves the following properties [17]:

- Convergence: When all sites have executed all the operations, the state is the same in each site.
- Causality: If an Operation O was executed before another operation O' in one site, O will be executed before O' at all the other sites.
- Intention: For any operation O , the effects of executing O at all sites are the same as the intention of O , and the effect of executing O does not change the effects of independent operations.

There are two ways to guarantee the convergence: a consensus algorithm, which solves conflicts between updates [18, 9], and Commutative Replicated Data Types, in which the operations are defined in a way that, they commute in case of concurrency, avoiding the need of reconciliation. As the burden of a consensus algorithm is high, we choose the CRDT approach. Causality can be achieved if the underlying network delivers in causal order, further analysis is needed if one does not work under that assumption. Intention needs to be analyzed depending on the operations.

The question we try to answer is: can we define a CRDT to manage a semantic store, that is, compliant with the SPARQL 1.1 Update specification?. A Semantic store is composed of a sequence of RDF-Graphs, which is a set of RDF-Triples. Therefore, we need first a CRDT for set. The SPARQL Update operations for RDF-Graphs can be summarized as the insertion or deletion of "ground triples", e.g. triples that are defined by the user in the body of the query, and "pattern- oriented" operations, where the triples to be affected are calculated from patterns.

In this paper we present SU-Set, a CRDT to handle RDF-Graphs and the SPARQL 1.1 Update operations.

2. RELATED WORK

In [2], Berners-Lee and Connolly proposed Delta, an approach in which the RDF-Graphs differences are calculated on their serialized forms, and the synchronization is achieved by applying this patches, much like a Version Control System. However, there is no explanation about how it can be consistent.

RDFGrowth [20] puts in place a sharing platform, where some peers can publish data, but other peers can only read. A reconciliation algorithm takes the responsibility to integrate concurrent updates. However, sharing is not the same as collaboration.

Edutella [12] is a P2P network for searching and sharing semantic web metadata (therefore in RDF). It divides peers in simple peers, which provides the data source along with its schema, and super peers, which mediate and integrate the data, and perform query routing whenever is necessary. There is no mechanism to synchronize the metadata.

RDFSync [19] is an implementation of the diff-sending idea described by Berners-Lee. RDF-Graphs are decomposed in subgraphs that are self-contained, called MSGs. The diff between MSGs is less expensive to calculate and send than the diff of the whole graph. An HTTP protocol is provided to ease the process in real-life conditions. RDFSync is more suitable for mirroring, and there is no considerations of what happens in case of concurrent modifications.

RDFPeers [5] is a scalable distributed RDF repository, using partial replication to tolerate faults. As RDFSync, there is no specification of what to do when updates are concurrent.

The Thomas Write rule [8] is a classical result from distributed systems that allows a fixed number of processes to maintain replicas of a database over an unreliable communication channel. However, it requires a garbage collection protocol that needs to know the number of total sites, which don't complies with the P2P networks context.

A Commutative Replicated Data Type [23] is a data type whose operations commute when they are concurrent [1], avoiding any worries about the arrive order and posterior reconciliation. A CRDT is composed of a payload: a set of atoms or objects that carry the representation of the type, and operations, which can be queries, if they don't mutate the object, and updates, if they do. Update operations have two parts: *atSource*, where the source replica prepares arguments for the *downstream*, which is then executed asynchronously in all the remote replicas. Both phases of update operations can have preconditions, marked with the keyword *pre*.

CRDTs for character sequences has been successfully used for building a collaborative infrastructure based on distributed semantic Wikis [16].

If we try to directly write the traditional set specification for non-distributed environments as a CRDT, with the operations of addition and removal of elements, we will find that it will not work, because the addition of an element does not commute with the removal of the same element, e.g.

$$(\{x\} \setminus \{x\}) \cup \{x\} = \{x\}$$

[1]Note the difference between this and the operations being commutative themselves

Specification 1: C-Set

payload set S = $\{(element, count), ...\}$
  initial $\emptyset$
query *lookup* (element e) : boolean b
  let $b = (\exists k \mid : (e,k) \in S \wedge k > 0)$
update *add* (element e)
  atSource(e)
    if $(\exists k \mid : (e,k) \in S \wedge k \leq 0)$
      let $j = |k| + 1$
    else
      let $j = 1$
  downstream(e, j)
    let $k' : (e,k') \in S$
    $S := S \setminus \{(e,k')\} \cup \{(e,k'+j)\}$
update *remove* (element e)
  atSource(e)
    pre *lookup*(e)
  downstream(e)
    $S := S \setminus \{(e,k')\} \cup \{(e,k'-1)\}$

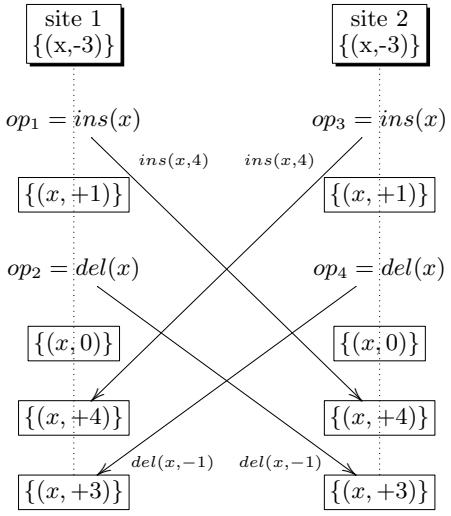

Figure 2: C-Set Intention Counterexample

while

$$(\{x\} \cup \{x\}) \setminus \{x\} = \{\}$$

Therefore, one needs to add extra information to preserve commutativity, while conserving the semantic of the original set.

C-Set [1], was the first attempt made with RDF in mind. The idea is to associate to each set element a counter that tracks how many times it has been inserted or deleted. Whenever an insert is executed, it compensates the effect of all previous deleted operations. As the whole process reduces to a sequence of additions over Z, which commutes, C-Set operations commute, and therefore, it converges. Specification 1 shows the C-Set CRDT.

A very interesting property of C-Set is that it does not require causal delivery from the underlying network to achieve convergence, although it would need it if one wants to perform a garbage collection to remove elements with counter equal to zero.

Unfortunately, although the convergence is assured, there is a case where the intention is compromised: imagine two sites in a state where a certain element e has a negative counter, meaning that it is not visible to the users. Now, each site performs an insert(e) followed by a delete(e), an "undo" operation. Due to the $|k|$ factor needed to cancel the negative counter and compensate the previous deletions, the final convergence is to a state where e is still there, contradicting the intended effect of both sites. The full execution is described in figure 2. The labels of the arrows represent the arguments being pushed downstream.

C-Set teaches us that convergence only is not enough to have an usable type, unless one can tolerate the intention violation within the semantic of the application.

In [15], Shapiro et. al. presents the Observed-Removed Set (OR-Set), in which each insertion is tagged with an unique id. These unique ids can be generated by the means of an

Specification 2: OR-Set

payload set S
  initial $\emptyset$
query *lookup* (element e) : boolean b
  let $b = (\exists u \mid : (e,u) \in S)$
update *add* (element e)
  atSource(e)
    let $\alpha = unique()$ // returns unique value
  downstream(e, α)
    $S := S \cup \{(e,\alpha)\}$
update *remove* (element e)
  atSource(e)
    pre *lookup*(e)
    let $R = \{(e,u) \mid (\exists u : (e,u) \in S)\}$
  downstream(R)
    // Causal Reception precondition
    pre $\forall (e,u) \in R : add(e,u)$ has been delivered
    $S := S \setminus R$

UUID [13] implementation, or by a mechanism for ordering events (in this case, the insertions) in a distributed system such as [11].

An element is considered member of the set if there is at least one pair of the form (element,id). When removing an element, a semantic store can only delete, and only send downstream to delete, the pairs (element,id) that it saw at that time. Specification 2 [15] details the OR-Set

As the pairs inserted in the set are unique, it is straightforward to show that OR-Set operations commute, and therefore, eventually converge. Although its creators did not considered intention in their consistency model, OR-Set also preserves it. The effect of each operation is to add or delete an unique pair, meaning there is no clash between independent operations. In case of concurrency of addition and deletion of the same element, the insertion has precedence. Figure 3 shows an execution of OR-Set. Causality is assumed to be provided by the underlying network, as expressed in

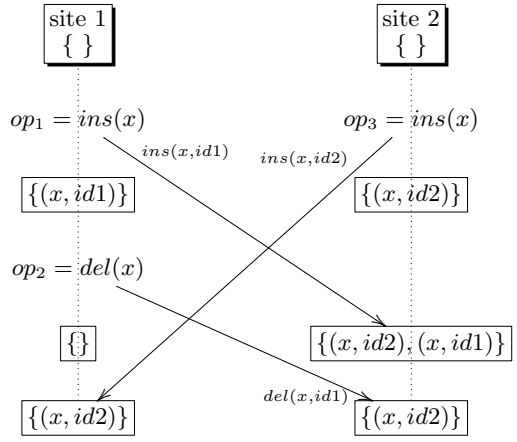

Figure 3: OR-Set execution

the precondition of the downstream phase of the remove operation:

$$\forall (e, u) \in R : add(e, u) \textit{ has been delivered}$$

Meaning that before a removal can be executed when it arrives to a remote semantic store, all additions of the element being removed observed at the originating store, and therefore, scheduled for deletion, must have been arrived and executed in the remote one.

We try to extend the OR-Set operations, which consider only single elements, to the insert and delete of SPARQL Update, which consider sets and patterns.

3. SU-SET SPECIFICATION

We recall the following definitions from the SPARQL specification [21]:

- URI is the set of URI References.
- $Literal$ is the set of literal unicode strings.
- $Blank$ is the set of blank nodes.
- an RDF triple is a 3-tuple (s, p, o) where $(s \in URI \vee s \in Blank) \wedge p \in URI \wedge (o \in URI \vee o \in Blank \vee o \in Literal)$. s is the subject of the triple, p is the predicate and o is the object.
- An RDF Graph G is a set of RDF triples.
- A Graph Store is a mutable RDF Dataset. An RDF Dataset is a set of RDF-Graphs that have an associated URI (called the Graph "name"). In an RDF Dataset always exists one graph without name called the "Default Graph".

For this work, we restrict to RDF-Graphs. Once we have validated the RDF-Graph approach, we may proceed with the RDF-stores. We also set aside the blank nodes, as their semantics and correct use are still a debate topic for the RDF and SPARQL community [10]. In this sense, we follow the recommendation by Heath and Bizer in the context of linked data [7]: "All resources in a dataset should be named with an URI reference", citing the added difficulty to merge data when the blank nodes are scoped to the document in they which appear.

The operations to update an RDF graph are the following [22]:

- $insert(T)$ Takes a set of triples and performs its union with the RDF-Graph.
- $delete(T)$ Takes a set of triples and performs its difference with the RDF-Graph.
- $delete{-}insert(whrPat, delPat, insPat)$ Executes a 'Select $*$ FROM CurrentGraph WHERE $whrpat$', then, performs the difference between the current graph and the triples in that 'Select' that match the pattern $delPat$, followed by the union with the triples in the 'Select' that match $insPat$. $insPat$ or $delPat$ can be null, meaning no execution of the union or difference step respectively.
- $load(IRI)$ loads all the triples from a remote RDF document at IRI. It can be expressed as an insertion.
- $clear()$ Removes all triples from the graph. It can be regarded as a $delete - insert(*, *, null)$, where $*$ is the pattern that matches any triple.

It is not possible to apply OR-Set directly to SPARQL Update it considers only insertion and deletion of single elements. We could modify the operations to, after calculating the relevant set of triples to affect, send them one by one, but that could flood the network with traffic, considering the potential size of an RDF-Graph.

To address this, we can modify the insert and delete operations to send sets instead of individual elements, effectively adding to the OR-Set the union and difference operations. This insert and delete of triples follows the same idea that OR-Set, the only difference is that we substitute the concept of a tag associated to each element, for a tag associated to the operation that inserted it. For the delete-insert, a little more work is needed because the deletion and posterior insertion need to remain atomic. Specification 3 shows the final result.

As an individual insert doesn't add the same element more than once, the pairs (element,tag) being added to the payload are unique (like a compound key), which allows us to preserve the eventual convergence. Also note that, if we restrict the sets of elements operated to singletons and ignore the pattern-oriented delete-insert, SU-Set reduces to the original OR-Set.

Now, we shall prove that the delete-insert operation commutes with concurrent operations of itself, delete and insert. Consider two concurrent delete-inserts over two different replicas in the same payload S. If we denote D and D' as the sets of pairs (triple, id) to delete of each operation, and I, I' the set of pairs (triple,id) to insert, we need to show that:

$$(((S \setminus D) \cup I) \setminus D') \cup I' = (((S \setminus D') \cup I') \setminus D) \cup I$$

Specification 3: SU-Set

payload set S
  initial $\emptyset$
query *lookup* (element e) : boolean b
  let $b = (\exists u : (t,u) \in S)$
update *insert* (set<element> T)
  atSource(T)
    let $R = \emptyset$
    foreach t in T:
      let $\alpha = unique()$
      $R := R \cup \{(t, \alpha)\}$
  downstream(R)
    $S := S \cup R$
update *delete* (set<element> T)
  atSource(T)
    let $R = \emptyset$
    foreach t in T:
      let $Q = \{(t,u) \,|\, (\exists u : |\, (t,u) \in S)\}$
      $R := R \cup Q$
  downstream(R)
    pre All add(t,u) have been delivered
    $S := S \setminus R$
update $delete - insert(whrPat, delPat, insPat)$
  // *match(m, pattern): triples that match*
  // *pattern within mapping m.*
  atSource($whrPat, delPat, insPat$)
    let $S' = \{t \,|\, (\exists u \,|\, : (t,u) \in S)\}$
    // *M is a Multiset of mappings*
    let $M =$ eval(Select $*$
                from S' where whrPat)
    $D' = \emptyset$
    foreach m in M:
      let $D' = D' \cup match(m, delPat)$
    let $D = \{(t,u) : t \in D' \wedge (t,u) \in S\}$
    foreach m in M:
      let $I' = I' \cup match(m, insPat)$
    let $\alpha = unique()$
    let $I = \{(i, \alpha) : i \in I'\}$
  downstream(D, I)
    // *Causal Reception*
    pre All add(f,u) $\in D$ have been delivered
    $S := (S \setminus D) \cup I$

All pairs are effectively unique and $D \cap I$, $D \cap I'$, $D' \cap I$ and $D' \cap I'$ equal to $\emptyset$, this, together with the fact that $A \cap B = \emptyset \Rightarrow A \cup B^{\complement} = B^{\complement}$, allows us to prove with simple set algebra that both sides of the formula reduce to:

$$((S \cup I \cup I') \setminus D) \setminus D'$$

We show how to convert the left side of the equality. The right side is analogous:

$(((S \setminus D) \cup I) \setminus D') \cup I'$
$= <A \setminus B = A \cap B^{\complement}>$
$(((S \cap D^{\complement}) \cup I) \cap D'^{\complement}) \cup I'$
$= <$Distributivity of $\cup$ over $\cap>$
$((S \cup I) \cap (I \cup D^{\complement}) \cap D'^{\complement}) \cup I'$
$= <I \cap D = \emptyset \Rightarrow I \cup D^{\complement} = D^{\complement}>$
$((S \cup I) \cap D^{\complement} \cap D'^{\complement}) \cup I'$
$= <$Distributivity of $\cup$ over $\cap>$
$(S \cup I \cup I') \cap (D^{\complement} \cup I') \cap (D'^{\complement} \cup I')$
$= <I' \cap D = \emptyset \wedge I' \cap D' = \emptyset>$
$(S \cup I \cup I') \cap D^{\complement} \cap D'^{\complement}$
$= <A \setminus B = A \cap B^{\complement}>$
$((S \cup I \cup I') \setminus D) \setminus D'$

The proofs of the commutativity of delete-insert with concurrent deletes or inserts are very similar.

4. DISCUSSION

SU-Set is a CRDT for RDF-Graphs that supports SPARQL Update 1.1 that satisfies the Strong Eventual Consistency. Unlike previous approaches like C-Set, it does not require tombstones, eliminating the burden of garbage collection. However, it does need causal delivery from the underlying network, which is challenging in highly dynamic contexts like ours.

SU-Set handles sets instead of individual elements, diminishing the traffic over the network compared to OR-Set operations like shown below:

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>

WITH <http://example/addresses>
DELETE { ?person foaf:givenName 'Bill' }
INSERT { ?person foaf:givenName 'William' }
WHERE { ?person foaf:givenName 'Bill' }
```

can induce a large set of triples to delete and a large set of triples to insert, affecting the time of re-execution. Further compression strategies may be needed to overcome this.

5. CONCLUSIONS AND FUTURE WORK

In this paper we present SU-Set, a CRDT for RDF-Graphs that supports the SPARQL 1.1 Update operations and guarantees strong eventual consistency. SU-Set is designed to serve as base for an RDF-Store CRDT that could be implemented in an RDF engine. SU-Set does not need tombstones, eliminating the burden of garbage collection, but it relies on causal delivery of the underlying network, which can pose problems in highly dynamic environments.

Future work includes:

- Implement SU-Set within an RDF engine.
- Perform validation on real data, comparing the network traffic and the time and space complexity against an implementation with pure OR-Set. We expect to have important savings in traffic and time, at an affordable cost in space.
- If we can validate the approach, use it to construct a CRDT for RDF-Stores.
- Investigate under which conditions blank nodes can be allowed in our scheme.
- Explore ways to formalize the intention, in order to be able to prove it algebraically. This should be added as an extra property to proof when showing correctness of a CRDT.

6. ACKNOWLEDGEMENTS

This work is supported by the French National Research agency (ANR) through the KolFlow project (code: ANR-10-CONTINT-025), part of the CONTINT research program.

7. REFERENCES

[1] K. Aslan, H. Skaf-Molli, P. Molli, and S. Weiss. C-set : a commutative replicated data type for semantic stores. In *RED: Fourth International Workshop on Resource Discovery, At the 8th Extended Semantic Web Conference (ESWC)*, pages 123–130, 2011.

[2] T. Berners-Lee and D. Connolly. Delta: an ontology for the distribution of differences between rdf graphs. `http://www.w3.org/DesignIssues/Diff`, 2004.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal of Semantic Web Information Systems*, 5(3):1–22, 2009.

[4] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[5] M. Cai and M. Frank. Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In *13th international conference on World Wide Web*, pages 650–657, 2004.

[6] T. Gruber. Collective knowledge systems: Where the social web meets the semantic web. *Journal of Web Semantics*, 6(1):4–13, 2008.

[7] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan and Claypool, 1st edition, 2011.

[8] P. Johnson and R. Thomas. Rfc677: The maintenance of duplicate databases, 1976.

[9] A. Mallea, M. Arenas, A. Hogan, and A. Polleres. On blank nodes. In *International Semantic Web Conference (1)*, pages 421–437, 2011.

[10] F. Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, 1(23):215–226, 1989.

[11] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. Edutella: a p2p networking infrastructure based on rdf. In *11th international conference on World Wide Web*, pages 604–615, 2002.

[12] M. M. P. Leach and R. Salz. Rfc4122: A universally unique identifier (uuid) urn namespace, 2005.

[13] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Survey*, 37(1):42–81, 2005.

[14] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of convergent and commutative replicated data types. Technical report, INRIA, January 2011.

[15] H. Skaf-Molli, G. Canals, and P. Molli. Dsmw: Distributed semantic mediawiki. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010*, pages 426–430, 2010.

[16] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, 1998.

[17] G. Tummarello, C. Morbidoni, R. Bachmann-Gmür, and O. Erling. Rdfsync: Efficient remote synchronization of rdf models. In *6th International and 2nd Asian Semantic Web Conference (ISWC + ASWC)*, pages 537–551, 2007.

[18] G. Tummarello, C. Morbidoni, J. Petersson, P. Puliti, and F. Piazza. Rdfgrowth, a p2p annotation exchange algorithm for scalable semantic web applications. In *Proceedings of the MobiQuitous'04 Workshop on Peer-to-Peer Knowledge Management (P2PKM 2004)*, 2004.

[19] W3C. *Resource Description Framework (RDF): Concepts and Abstract Syntax*, Feb. 2004. `http://www.w3.org/TR/rdf-concepts/`.

[20] W3C. *SPARQL 1.1 Update*, Jan. 2012. `http://www.w3.org/TR/2012/WD-sparql11-update-20120105/`.

[21] S. Weiss, P. Urso, and P. Molli. Logoot-undo: Distributed collaborative editing system on p2p networks. *IEEE Transactions on Parallel Distributed Systems*, 21(8):1162–1174, 2010.