Towards Real-Time Co-authoring of Linked-Data on the Web

Moulay Driss Mechaoui, Nadir Guetmi, Abdessamad Imine

Towards Real-Time Co-Authoring of Linked-Data on the Web

Moulay Driss Mechaoui, Nadir Guetmi, and Abdessamad Imine

[1] University of Sciences and Technology Oran 'Mohamed Boudiaf' USTO-MB
Mathematics and Computer Science Faculty
Oran, Algeria
[2] LIAS/ISAE-ENSMA, Poitiers University
Chasseneuil, France
[3] Université de Lorraine and INRIA-LORIA Grand Est
Nancy, France
{moulaydriss.mechaoui@univ-usto.dz,
nadir.guetmi@ensma.fr,
abdessamad.imine@loria.fr}

Abstract. Real-time co-authoring of Linked-Data (LD) on the Web is becoming a challenging problem in the Semantic Web area. LD consists of RDF (Resource Description Framework) graphs. We propose to apply state-of-the art collaborative editing techniques to manage shared RDF graphs and to control the concurrent modifications. In this paper, we present two concurrency control techniques. The first one is based on client-server architecture. The second one is more flexible as it enables the collaborative co-authoring to be deployed in mobile and P2P architecture and it supports dynamic groups where users can leave and join at any time.

Keywords: Linked-Data, Collaborative Editing Systems, Optimistic Replication.

1 Introduction

Recently, providing collaborative co-authoring tools in the Web Semantic is becoming more attractive as they enable semantic web data to be produced in online mode and to be available to a large public. Linked Data (LD) is recently used to replace collections of offline RDF data [3]. The goal of LD is to enable people to share structured data on the web as easily as they can share documents today. It uses RDF technology that (i) relies on HTTP URIs to denote things; (ii) provides useful information about a thing at that thing's URI; and (iii) includes in that information other URIs of LD. Tabulator [2] is a LD browser, designed to provide the ability to navigate the web of linked things. In [3], Berners-Lee et al. raise some interesting challenges when adding collaborative co-authoring mode in Tabulator. This mode consists in collaboratively editing the LD which is represented by a RDF graph.

In this paper, we sketch two solutions that may meet to some extent the read-write requirement in LD browser. We consider a RDF graph as a shared data which can be edited and updated by several users. To control the concurrent access to this shared data, we propose to apply state-of-the art collaborative editing techniques [9, 6]. The CRDT (Commutative Replicated Data Type) is a class of algorithms that is emerging for ensuring consistency of highly dynamic content on P2P networks. However, this approach incurs some overhead they do not consider directly a set as a list (or a sequence) [1]. Also, with the continuously growing amount of structured data available on the Semantic Web there is an increasing desire to replicate such data to mobile devices. This enables services and applications to operate independently of the network [18, 11]. Classical replication techniques cannot be properly applied to mobile systems because they do not adopt to changing user information needs, and they do not consider the technical, environmental, and infrastructural restrictions of mobile devices.
We think that Operational Transformation (OT) approach [4, 14] may be a good candidate as it supports unconstrained interaction. Indeed, it allows any user to modify any shared data consistently at any time without any restrictions on users's actions.

The rest of the paper is organized as follows. Section 2 presents the ingredients of OT approach. In Section 3, we suggest two concurrency control procedures for managing the collaborative edition of RDF graphs. Section 4 discusses performance evaluation, and concludes.

2 Transformational Approach

Principle. Operational Transformation (OT) is an optimistic replication technique which allows many users (or sites) to concurrently update the shared data and next to synchronize their divergent replicas in order to obtain the same data [17]. The updates of each site are executed on the local replica immediately without being blocked or delayed, and then are propagated to other sites to be executed again. Accordingly, every update is processed in four steps: (i) *generation* on one site; (ii) *broadcast* to other sites; (iii) *reception* on one site; (iv) *execution* on one site.

A crucial issue when designing shared data with a replicated architecture and arbitrary messages communication between sites is the *consistency maintenance* (or *convergence*) of all replicas. To illustrate this problem, consider the following example:

Example 1. Consider the following group text editor scenario (see Figure 1.(a)): there are two users (on two sites) working on a shared document represented by a sequence of characters. These characters are addressed from 0 to the end of the document. Initially, both copies hold the string "*efecte*". User 1 executes

operation $op_1 = Ins(1, f)$ to insert the character f at position 1. Concurrently, user 2 performs $op_2 = Del(5)$ to delete the character e at position 5. When op_1 is received and executed on site 2, it produces the expected string "*effect*". But, when op_2 is received on site 1, it does not take into account that op_1 has been executed before it and it produces the string "*effece*". The result at site 1 is different from the result of site 2 and it apparently violates the intention of op_2 since the last character e, which was intended to be deleted, is still present in the final string. Consequently, we obtain a *divergence* between sites 1 and 2. It should be pointed out that even if a serialization protocol [4] was used to require that all sites execute op_1 and op_2 in the same order (*i.e.* a global order on concurrent operations) to obtain an identical result *effece*, this identical result is still inconsistent with the original intention of op_2.

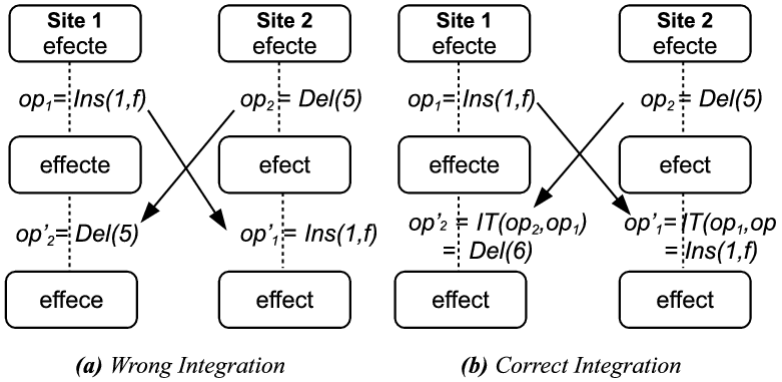


Fig. 1. Serialization of concurrent updates

To maintain convergence, the OT approach has been proposed by [4]. When User X gets an operation op that was previously executed by User Y on his replica of the shared object User X does not necessarily integrate op by executing it "as is" on his replica. He will rather execute a variant of op, denoted by op' (called a *transformation* of op) that *intuitively intends to achieve the same effect as* op. This approach is based on a transformation function (or algorithm) IT that apply to couples of concurrent operations defined on the same state.

Example 2. In Figure 1.(b), we illustrate the effect of IT on the previous example. When op_2 is received on site 1, op_2 needs to be transformed according to op_1 as follows: $op'_2 = IT((Del(5), Ins(1, f)) = Del(6)$. The deletion position of op_2 is incremented because op_1 has inserted a character at position 1, which is before the character deleted by op_2. Next, op'_2 is executed on site 1. In the same way, when op_1 is received on site 2, it is transformed as follows: $IT(Ins(1, f), Del(5)) = Ins(1, f)$; op_1 remains the same because f is inserted before the deletion position of op_2.

Intuitively we can write the transformation IT as follows:

```
IT(Ins(p1,c1),Ins(p2,c2)) =
          if (p1 < p2) return Ins(p1,c1)
          else return Ins(p1+1,c1)
          endif;
```

OT Model. Using the OT approach, each site is equipped by two main components [4, 10]: the *integration component* and the *transformation component*. The integration component determines how an operation is transformed against a given operation sequence (*e.g.*, the log buffer). It is also responsible for receiving, broadcasting and executing operations. It is rather *independent* of the type of the shared data. The transformation component is a set of IT algorithms which is responsible for merging two concurrent operations defined on the same state. Every IT algorithm is *specific* to the semantics of a given shared data.

The most known OT-based theoretical framework is established by Ressel et al. [10]. They define two consistency criteria:

- **Causality.** If one operation op_1 causally precedes another operation op_2, then op_1 must be executed before op_2 at all sites.
- **Convergence.** When all sites have performed the same set of operations, the copies of the shared data must be identical.

It has been proved that any integration component can achieve convergence in the presence of arbitrary transformation paths if its IT algorithm satisfies two properties $TP1$ and $TP2$ [10]. For all op, op_1 and op_2 pairwise concurrent operations with $op'_1 = IT(op_1, op_2)$ and $op'_2 = IT(op_2, op_1)$:

- **TP1**: $[op_1 \,; op'_2] \equiv [op_2 \,; op'_1]$.
- **TP2**: $IT(IT(op, op_1), op'_2) = IT(IT(op, op_2), op'_1)$.

Property $TP1$ defines a *state identity* and ensures that if op_1 and op_2 are concurrent, the effect of executing op_1 before op_2 is the same as executing op_2 before op_1. This property is necessary but not sufficient when the number of sites is greater than two. Property $TP2$ defines an *update identity* and ensures that transforming op along equivalent and different operation sequences will give the same operation.

Properties $TP1$ and $TP2$ are sufficient to ensure the convergence for *any number* of concurrent operations which can be executed in *arbitrary order* [10]. Accordingly, by these properties, it is not necessary to enforce a global total order between concurrent operations because data divergence can always be repaired by operational transformation. However, finding an IT algorithm that satisfies $TP1$ and $TP2$ is considered as a hard task, because this proof is often unmanageably complicated [13]. To overcome this difficulty, we proposed in [6] a formal methodology for designing and analyzing IT algorithms by using a theorem prover.

Several OT-based integration components have been proposed in the groupware research area. These components may be categorized in two categories. The first one does not require $TP2$ property: it relies on client-server architecture for enforcing a unique transformation order. We can cite in this category algorithms like SOCT4 [16] and TIBOT [7]. As for the second category, it requires $TP2$ property. This constraint enables the concurrent operations to be synchronized in a decentralized way. Algorithms such as adOPTed [10] SOCT2,4 [15, 16] and GOTO [14] belong to this category.

3 Our Proposals

To manage all concurrent access for editing collaboratively a shared RDF graph, we need a concurrency control procedure. In this section, we first argue how to map a RDF graph into a sequence data structure. According to centralized and decentralized architectures, we suggest two concurrency control procedures.

3.1 RDF Graph as a sequence

When publishing LD on web, information about resources is represented using the RDF. Any expression in RDF is a collection of *triples*, each consisting of a *subject*, a *predicate* (also called property) and an *object*. The subject of a triple is the URI describing resource. The object can either be a simple literal value (*e.g.*, a string, a number) or the URI of another resource. The predicate indicates what kind of relation exists between subject and object. The predicate is a URI too. A set of such triples is called an RDF graph. This can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link.

Usually a set is implemented by means of a list. It means we can use operations, such as insert and delete, to edit a shared list. Thus, we can reuse the state-of-the-art of collaborative editing systems.

For instance, the following three english statements (this example is taken from [8]):

- <http://www.example.org/index.html>has a creator whose value is John Smith
- <http://www.example.org/index.html>has a creation-date whose value is August 16, 1999
- <http://www.example.org/index.html>has a language whose value is English

could be represented by the RDF graph shown in Figure 2.

An RDF graph can be serialized into a sequence of triples and considered as a text where each line corresponds to a simple triple of subject, predicate and

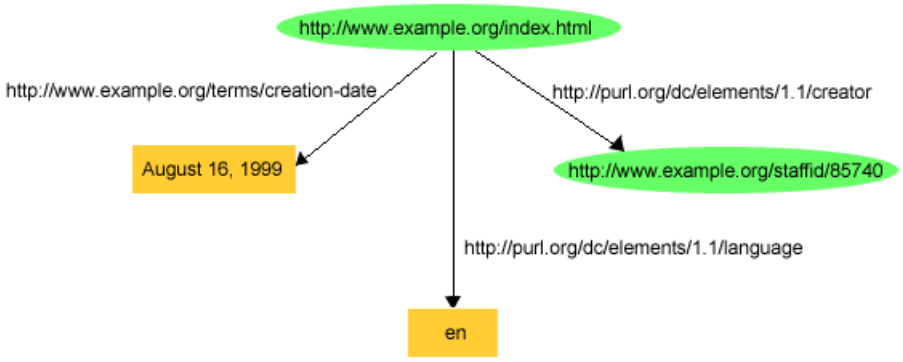


Fig. 2. An RDF Graph.

object. For example, the third statement shown in Figure 2 would be written as a text line:

```
<http://www.example.org/index.html>
                <http://purl.org/dc/elements/1.1/language> "en" .
```

By considering an RDF graph as a sequence, each triple is addressed simply by a position within the sequence. Therefore, we assume that the sequence of triples can be modified by the following primitive operations:

- $Ins(p, t)$ which adds triple t at position p;
- $Del(p, t)$ which deletes triple t at position p.

Updating a triple (*e.g.*, by modifying the predicate URI) can be expressed by a sequence of delete (by removing the old triple) and insert (by adding the new one) operations.

3.2 Ingredients of collaboration

Each user's site has a local copy of RDF graph and a unique identity. We assume that the RDF graph is serialized in the same way on every site.

Every site generates operations sequentially and stores these operations in a stack also called a *log*. When a site receives a remote operation op, the integration component executes the following steps:

1. from the local log it determines the sequence seq of operations that are concurrent to op;
2. it calls the transformation component in order to get operation op' that is the transformation of op according to seq;
3. it executes op' on the current state;

4. it adds op' to the local log.

3.3 Centralized Solution

In this section, we propose a real-time co-authoring based on client-server architecture. Indeed, users can edit collaboratively a shared RDF graph by reconciliating their divergent copies via a particular site called *server*. We think that SOCT4 [16] is most appropriate to this kind of architecture.

In SOCT4, the operations are ordered globally by using a timestamp given by the server. When an operation is generated on site s, it is immediately executed (to satisfy real-time constraint), but it is not propagated until it gets a timestamp from the server and all the operations which precede it according to the timestamp order have been received and executed on s. Moreover, this operation is transformed against all concurrent operations (operations received after its generation and preceding it in the global order) before to be propagated. To ensure convergence, SOCT4 requires only the property $TP1$ to be satisfied by the IT algorithm.

$site\,1$	$site\,2$
$op_1 = Ins(0, t_1)$	$op_3 = Ins(0, t_3)$
$op_2 = Ins(1, t_2)$	
$s_1 = synchronize$	
	$s_2 = synchronize$
$s_3 = synchronize$	

Fig. 3. Scenario of collaboration

Example 3. Consider two users editing a shared RDF graph as described in Figure 3. Initially, each site has an empty copy. The index of each operation represents the timestamp given by the server. Two local insertion operations op_1 and op_2 have been executed by user 1 (at site 1). Concurrently, user 2 has executed another insertion operation op_3. The added triples t_1, t_2 and t_3 are respectively as follows (where `UR1` is `http://www.example.org/index.html` and `UR2` is `http://www.example.org/staffid/85740`):

```
<UR1> <http://www.example.org/terms/creation-date> "August 16, 1999" .

<UR1> <http://purl.org/dc/elements/1.1/language> "en" .

<UR1>  <http://purl.org/dc/elements/1.1/creator> <UR2> .
```

1. At point s_1, site 1 decides to synchronize with other sites. As there is no concurrent operation available, op_1, op_2 are sent to site 2 (via the server) in their original forms.

2. At point s_2, site 2 cannot send op_3 as long as it did not receive the precedent operations (according to the timestamp order). Thus the synchronization calls IT algorithm to produce the following transformations:

$op'_1 = IT(op_1, op_3) = Ins(0, t_1)$
$op'_3 = IT(op_3, op_1) = Ins(1, t_3)$
$op'_2 = IT(op_2, op'_3) = Ins(1, t_2)$
$op''_3 = T(op'_3, op_2) = Ins(2, t_3)$

op'_1, op'_2 are executed on site 2, and op''_3 is broadcast to other sites.

3. At point s_3, site 1 decides again to synchronize. The remote operation op''_3 is executed directly (without transformation) after op_1 and op_2.

4. Note that, after point s_3, sites 1 and 2 have the same log, namely op_1, op_2 and op''_3. However, site 1 has performed the following sequence:

op_1
op_2
$op''_3 = IT(IT(op_3, op_1), op_2)$

while site 2 has executed the following sequence:

op_3
$op''_1 = IT(op_1, op_3)$
$op''_2 = IT(op_2, op'_3)$

As SOCT4 requires only $TP1$ property, the above sequences are equivalent in the sense that they produce the same RDF graph. The operations are stored in the log according to the timestamp order but they may be executed in different orders at different sites.

It should be noted that SOCT4 has been used successfully in the development of a File Synchronizer [9] distributed with the industrial collaborative development environment, LibreSource Community[4], proposed by ARTENUM Company. LibreSource is a platform for hosting virtual teams. Users can register and create channels for synchronizing shared data. On a single server, LibreSource can host several projects, several groups of users, and grant fine grain access to the resources.

Although SOCT4 ensures causality and convergence properties, it degrades the responsiveness of the system as all messages are exchanged via a server. Moreover, it does not scale because it is based on a single point of failure.

[4] `http://dev.libresource.org`

3.4 Decentralized Solution

Integration algorithms based on $TP2$ property enable concurrent operations to be synchronized in a decentralized way. Thus, they avoid a single point of failure. Nevertheless, these algorithms have limited scalability with the number of users. Indeed, all proposed OT frameworks rely on a fixed number of users during collaboration sessions. This is due in the fact that they use vector timestamps to enforce causality dependency. The vector timestamps do not scale well, since each timestamp is a vector of integers with a number of entries equal to the number of users.

In [5], we proposed a new framework for collaborative editing to address the weakness of previous OT works. The features of our framework are as follows:

1. It supports an unconstrained collaborative editing work (without the necessity of central coordination). Using optimistic replication scheme, it provides simultaneous access to shared data.
2. Instead of vector timestamps, we use a simple technique to preserve causality dependency. Our technique is minimal because only direct dependency information between operations is used. It is independent on the number of users and it provides high concurrency in comparison with vector timestamps.
3. Using OT approach, reconciliation of divergent copies is done automatically in decentralized fashion.
4. Our framework can scale naturally thanks to our minimal causality dependency relation. In other words, it may be deployed easily in Peer-to-Peer (P2P) networks.

Example 4. Consider the scenario given in Example 3. In our framework, operations op_1 and op_2 will be related by a dependency. This is due in the fact that their added triples are adjacent (positions 0 and 1) and created by the same user. Thus, op_1 must be executed before op_2 at all sites. This dependency relation is minimal in the sense that when op_2 is broadcast to all sites it holds only the identity of op_1 as it depends on directly.

1. At site 1, op_3 is considered as concurrent. It is then transformed against op_1 and op_2. The following sequence is executed and logged in site 1:

$op_1 = Ins(0, t_1)$
$op_2 = Ins(1, t_2)$
$op_3'' = IT(IT(op_3, op_1), op_2) = Ins(2, t_3)$

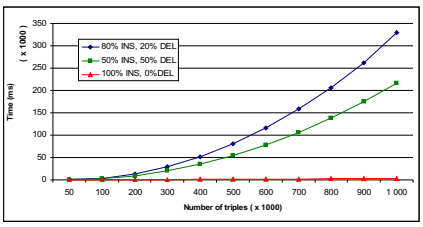
2. At site 2, op_1 and op_2 are concurrent with respect to op_3. They must be transformed before to be executed after op_3 according to their dependency relation. Thus, the following sequence is executed and logged in site 2:

$op_3 = Ins(0, t_3)$
$op'_1 = IT(op_1, op_3) = op_1$
$op'_2 = op_2$

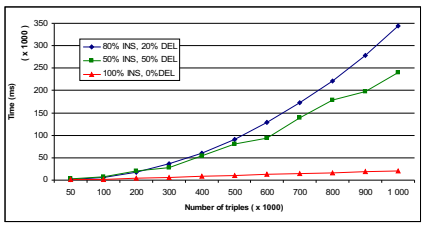
Unlike the others OT-based integration algorithms, we minimize the transformation steps when integrating a remote operation depending on another operation. Indeed, at site 2, the new form of op_2 is deduced from the executed form of op_1 (without transformation as in Example 3). On the other hand, the sequences of sites 1 and 2 are not identical but equivalent.

4 Performance Evaluation

Our experimentation consists to compare the response time of generating and integrating a sequence of remote triples over a local ones. We use two sites (Site1 and Site2), initially the log of each sites is empty. Each site generates locally a sequence of operations; the sites communicate the generated operations to be integrated. The sizes of the sequence are varied from 50 000 to 1 000 000 triples. The percentage of insertions in the sequence and the log are variants from 50%, 80% to 100%.



(a) Generation + Integration time of a sequence of triples over an empty RDF document



(b) Generation + Integration time of a sequence of triples over a RDF document containing 10 000 triples

Fig. 4. Updating RDF document

We implement a prototype of Optic [5] in java, compiled by NetBeans 6.8 with JVM heap size 1GB, and executed on a computer running Windows XP SP2 with an Intel (R) Core (TM) 2 CPU E7400 @ 2.80 GHz and 2 GB RAM. We calculate the sum of the generation time of the sequence in the Site1 with the time of integration of the same sequence in the Site2. For every generation and integrating sequence three times are executed and the average time is recorded.

The Figure 4(a) present the time of generation and integration of a varied sequences of triples over an empty RDF document. When the percentage of insertions in the sequence is 100% the performance of our algorithm increases. This

is due to the minimal causality dependency between insertions operations computed during the local generation of triples. The Figure 4(b) illustrate the time of generation and integration of a varied sequences of triples over a RDF document containing 10 000 triples. The performance decreases when the percentage of deletion increases. This degradation of performance is caused by the canonizing of the log [5] (tidy insertion operation before deletion operations). The rate of deletion operations in the log has a direct impact on the performance of the Optic algorithm.

5 Conclusion

In this paper, we have dealt with the problem of the real-time co-authoring of LDW. In this respect, we have suggested two solutions based on OT approach.

In centralized and decentralized solutions we propose in this paper, the shared RDF graph is serialized into a sequence of triples that can be altered by simple operations: insertion and deletion of triples. Mapping RDF graph into sequence of triples is given in order to reuse state-of-the art collaborative editing techniques including some systems in which we participated [9, 6]. This mapping is simple. But, if the RDF graph must satisfy some requirements based on semantic aspects (*e.g.*, graph connectiveness), preconditions must be added to operations. For example, we can state that the delete operation $Del(p, t)$ is enabled iff the p exits and the object of t is not a subject of another triple. It is not sure that this delete operation will be still enabled when it is integrated in another site which has added concurrently triple t' whose the subject is the object of t. Two solutions are possible: either writing another IT algorithm based on new constraints, or tolerating the violation of some requirements during some periods with the possibility to stabilize in correct state (by undoing some operations).

The question of adapting these solutions in existing semantic web browsers remains open in this paper. It will be interesting to plug these solutions in a given browser in order to evaluate the cost of mapping a RDF graph into a sequence. Using this implementation, we can also make measurements to experimentally validate the impact of OT approach on real-timeliness and scalability. On the other hand, designing a new IT algorithm for shared RDF graphs based on updates proposed in the recent version of SPARQL/Update [12] is an exciting and challenging problem.

References

1. K. Aslan, P. Molli, H. Skaf-Molli, and S. Weiss. C-set : a commutative replicated data type for semantic stores. In *Fourth International Workshop on REsource (RED)*, 2011.

2. T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, J. Dhanaraj, R.and Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *SWUI06 Workshop at ISWC06,*, Athens, Georgia, USA, 2006.
3. T. Berners-Lee, J. Hollenbach, L. Kanghao, E. Presbery, J. andPru d'ommeaux, and M. Schraefel. Tabulator redux: Writing into the semantic web. `http://eprints.ecs.soton.ac.uk`, 2008.
4. C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.
5. A. Imine. *Conception Formelle d'Algorithmes de Réplication Optimiste. Vers l'Edition Collaborative dans les Réseaux Pair-à-Pair*. Phd thesis, University of Henri Poincaré, Nancy, France,, December 2006.
6. A. Imine, M. Rusinowitch, G. Oster, and P. Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theoretical Computer Science*, 351(2):167–183, 2006.
7. R. Li, D. Li, and C. Sun. A time interval based consistency control algorithm for interactive groupware applications. In *IEEE ICPADS'2004*, pages 420–429, Los Alamitos, CA, USA, 2004.
8. F. Manola and E. Miller. Rdf primer. `http://www.w3.org/TR/rdf-primer/`, 2004.
9. P. Molli, G. Oster, H. Skaf-Molli, and A. Imine. Using the transformational approach to build a safe and generic data synchronizer. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 212–220. ACM Press, 2003.
10. M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauser. An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors. In *ACM CSCW'96*, pages 288–297, Boston, USA, November 1996.
11. O. Sacco, M. Collina, G. Schiele, G. E. Corazza, J. G. Breslin, and M. Hauswirth. Fine-grained access control for RDF data on mobile devices. In *WISE 2013*, pages 478–487, 2013.
12. A. Seaborne and G. Manjunath. Sparql/update: A language for updating rdf graphs. `http://jena.hpl.hp.com/~afs/SPARQL-Update.html`, 2008.
13. C. Sun and Agustina. Exhaustive search of puzzles in operational transformation. In *CSCW '14*, pages 519–529, 2014.
14. C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *ACM CSCW'98*, pages 59–68, 1998.
15. 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 Trans. Comput.-Hum. Interact.*, 5(1):63–108, March 1998.
16. N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *ACM CSCW'00*, Philadelphia, USA, December 2000.
17. X. Yi, S. Chengzheng, and L. Mo. Achieving convergence in operational transformation: Conditions, mechanisms and systems. In *CSCW '14*, pages 505–518, 2014.
18. S. Zander and B. Schandl. Context-driven RDF data replication on mobile devices. *Semantic Web*, 3(2), 2012.